SOFTWARE PROCESS MODELING in SOCCA

student : H.G.Brugman
studentnr : 9105506

version : 0.10
date : May 1998

Brief contents

Contents

SOFTWARE PROCESS MODELING in SOCCA

student : H.G.Brugman
studentnr : 9105506
version : 0.10

1. Introduction

This master thesis concludes the study of Computer Science at the University of Leiden. The mentor of this project is dr. L.P.J. Groenewegen of the Software Engineering and Information Systems (SEIS) group of the Department of Computer Science. The central theme of this thesis is 'process modeling in SOCCA'.

The acronym SOCCA stands for 'Specifications of Coordinated and Cooperative Activities'. SOCCA is a object-oriented process modeling language. It has been developed by the Software Engineering and Information Systems (SEIS) group of the Department of Computer Science of the University of Leiden. SOCCA is currently being refined and extended by the SEIS group.

Already there is experience within the SEIS group with large SOCCA models. Certain ideas to accomplish still larger SOCCA models are being contemplated by the SEIS group. This thesis looks into two of these ideas and gives a partly quantitative assessment of their usefullness in the construction of very large SOCCA models. Another topic of interest of the SEIS group is the use of a SOCCA model as a process description instead of the usual textual description. This thesis will also look into this issue.

The three topics addressed in this thesis are :

- investigate the use of often recurring SOCCA 'constructs' to limit the size of the model
- investigate if the SOCCA modeling language can be scaled up (i.e. can a larger SOCCA model be constructed from indepently developed sub-models)
- investigate the usefullness of a SOCCA model as a process description

The first question was handled as follows. A very large SOCCA model was developed. The size of this model is 29 classes, 86 operations and 1041 state transition diagrams. During the modeling recurring 'constructs' were identified, named and used. This resulted in the fact that only 310 state transition diagrams out of the 1041 had to be explicitily shown in the model.

The modeled process is the software process of the software development organization 'WBU' of the Dutch Ministry of Defense. Two processes of this organization were modeled using the SOCCA modeling language. These processes are the 'Software Configuration Management'-process and part of the 'Software Project Planning'-process. The SOCCA model of the 'Software Configuration Management'-process is given in chapter 4. The SOCCA model of the process fragment 'writing project management documents' (i.e. part of the 'Software Project Planning'-process) is given in chapters 5 and 6.

The identified SOCCA 'constructs' are discussed in chapter 3.

To look into the question of 'scaleability' of SOCCA, the modeling of the process fragment 'writing project management documents' was done by splitting the process fragment into four smaller process fragments. These smaller process fragments were modeled indepently of each other. Their SOCCA sub-models are described in chapter 5. Then the sub-models of these four smaller process fragments were integrated into one SOCCA model of the process fragment 'writing project management documents'. This integration is described in chapter 6.

The integration of sub-models introduced the concept of a 'control-object'. The 'integration' algorithm and the 'control-object' concept are described in chapter 3.

Lastly the usefullness of a SOCCA model as a process description was investigated. This was done by checking if the SOCCA models of the 'Software Configuration Management'-process and the 'Software Project Planning'-process could be used as input for a process audit. As audit method was chosen the 'Capability Maturity Model'-assessment.

The Capability Maturity Model is described (briefly) in chapter 2. Both chapter 4 (the SOCCA model of the 'Software Configuration Management'-process) and chapter 5 (the SOCCA model of part of the 'Software Project Planning'-process) include a CMM assessment checklist. The entries in this list are checked against the SOCCA process model.

SOFTWARE PROCESS MODELING in SOCCA

student : H.G.Brugman
studentnr : 9105506
version : 0.10

2. Capability Maturity Model

2.1 Introduction

To investigate the usefullness of a SOCCA model as a process description, a process audit is performed on the SOCCA models of the 'Software Configuration Management'-process and the 'Software Project Planning'-process. The audit method used is the 'Capability Maturity Model'-assessment.

This chapter gives a brief description of the 'Capability Maturity Model'. The CMM assessment checklists are included in chapter 4 (the SOCCA model of the 'Software Configuration Management'-process) and chapter 5 (the SOCCA model of the 'Software Project Planning'-process).

2.2 Description

Informally the total of software development and maintenance activities of an organization constitute its 'software process'. These activities include the traditional activities of the software project life cyle 'requirements analysis', 'system design', 'software design', 'coding', 'testing' and 'maintenance'. They also include 'project management', 'configuration management', 'software quality activities' and 'outsourcing'.

A more abstract definition is given in [LON] : 'A software process is a set of partially ordered process steps, with sets of related artifacts, human and computerized resources, organizational structures and constraints, intended to produce and maintain the requested software deliverables'.

The measure in which an organization controls its software process is an indication for the 'maturity' of the organization. A mature organization is more likely to deliver a software product according to the specifications, on time and within budget.

The Capability Maturity Model (CMM) is a framework that will help organizations manage and improve their software process. The CMM was developed by the Software Engineering Institute (SEI) of the Carnegie Mellon University. The CMM evolved from a SEI-project to establish guidelines for the assessment of the capabilities of software contractors of the American Air Force [HUM].

As such the CMM provides a set of generic rules for the software process. It prescribes regular measurement of the software process. Against these measurements the implementation of the rules is verified. On the basis of this feedback the software process can be better managed and improved.

The rules and guidelines of the CMM are generic. They form a famework. Organizations can choose their own methods and procedures to comply with these rules.

Levels of maturity

The Capability Maturity Model distinguishes between five levels of maturity [CMM] :

- level 1, Initial :

  the software process is characterized as 'ad hoc'. Few processes are defined. Success depends on individual effort of software engineers. (Every organization starts at level 1)

- level 2, Repeatable :

  basic project management processes are established. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

- level 3, Defined :

the software process for both management and engineering activities is documented, standardized and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software.

- level 4, Managed :

detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.

- level 5, Optimizing :

continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies. (The number of organizations on this level is very small)

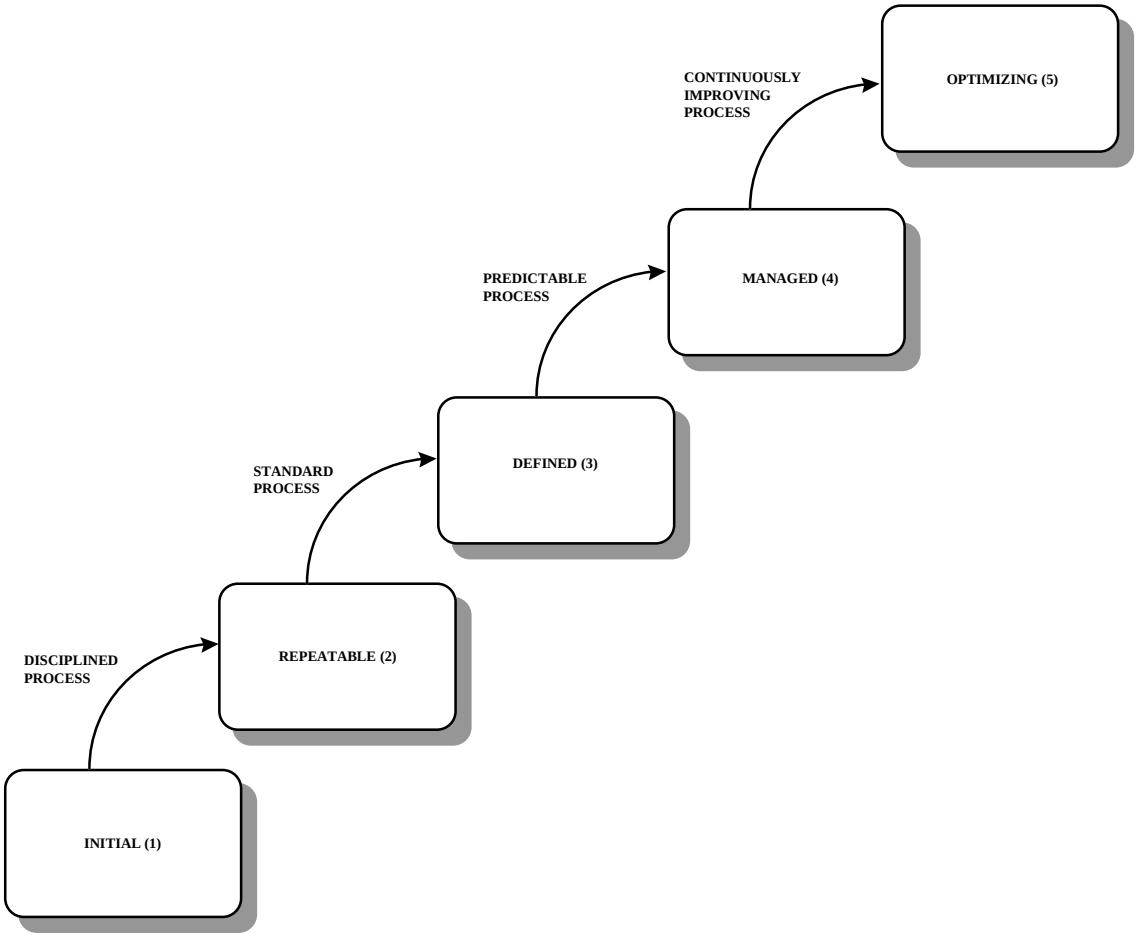


figure 2.1 progressing levels of software process maturity

An organization must progress from one maturity level to the next. Since each level is the foundation for the following level is it not possible to 'skip' a maturity level.

Key Process Area

Each maturity level is decomposed into several Key Process Areas (KPA). An organization must focus on these areas to improve its software process. Key Process Areas identify the issues that must be addressed to achieve a certain maturity level.

Within the five maturity levels the following Key Process Areas are recognized :

- level 1, Initial :

    - KPA : none

- level 2, Repeatable :

    - KPA : Requirements Management
    - KPA : Software Project Planning
    - KPA : Software Project Tracking and Oversight
    - KPA : Software Subcontract Management
    - KPA : Software Quality Assurance
    - KPA : Software Configuration Management

- level 3, Defined :

    - KPA : Organization Process Focus
    - KPA : Organization Process Definition
    - KPA : Training Program
    - KPA : Integrated Software Management
    - KPA : Software Product Engineering
    - KPA : Intergroup Coordination
    - KPA : Peer Reviews

- level 4, Managed :

    - KPA : Quantative Process Management
    - KPA : Software Quality Management

- level 5, Optimizing :

    - KPA : Defect Prevention
    - KPA : Technology Change Management
    - KPA : Process Change Management

For every KPA a general description (its purpose) will be given. From this description CMM distills the more specific ‘goals’ of each KPA. When the goals of a Key Process Area are accomplished on a continuing basis across projects, an organization can be said to have institutionalized the process capability characterized by that KPA. When all KPAs of a maturity level have been satisfied, an organization has reached that maturity level.

This thesis is concerned with the maturity level 2. For this reason, only the purpose and goals of the KPAs of level 2 are described below. For a description of the other KPAs the reader is referred to [CMM].

- KPA : Requirements Management :

    the purpose of Requirements Management is to establish a common understanding between the customer and the software project of the customer’s requirements that will be addressed by the software project.
    This agreement with the customer is the basis for planning (as described in KPA ‘Software Project Planning’) and managing (as described in KPA ‘Software Project Tracking and Oversight’) of the software project.
    Control of the relationship with the customer depends on following an effective change control process (as described in KPA ‘Software Configuration Management’).

    - Goal 1 :
    System requirements allocated to software are controlled to establish a baseline for software engineering and management use.

    - Goal 2 :
    Software plans, products, and activities are kept consistent with the system requirements allocated to software.

- KPA : Software Project Planning :

the purpose of Software Project Planning is to establish reasonable plans for performing the software engineering and for managing the software project. These plans are the necessary foundation for managing the software project (as described in the KPA 'Software Project Tracking and Oversight').

- Goal 1 :
Software estimates are documented for use in planning and tracking the software project.

- Goal 2 :
Software project activities and commitments are planned and documented.

- Goal 3 :
Affected groups and individuals agree to their commitment related to the software project.

- KPA : Software Project Tracking and Oversight :

the purpose of Software Project Tracking and Oversight is to establish adequate visibility into the actual progress of a software project so that management can take corrective actions when the project's performance deviates significantly from the software plans.

- Goal 1 :
Actual results and performance are tracked against the software plans.

- Goal 2 :
Corrective actions are taken and managed to closure when actual results and performance deviate significantly from the software plans.

- Goal 3 :
Changes to software commitments are agreed to by the affected groups and individuals. (This goal is strongly related to the KPA 'Software Project Planning' as a change in the software commitments results in a change in the planning.)

- KPA : Software Subcontract Management :

the purpose of Software Subcontract Management is to select qualified software subcontractors and manage them effectively. It combines the concerns of Requirements Management, Software Project Planning and Software Project Tracking and Oversight for basic management control, along with necessary coordination of Software Quality Assurance and Software Configuration Management, and applies this control to the subcontractor as appropriate.

- Goal 1 :
The prime contractor selects qualified software subcontractors.

- Goal 2 :
The prime contractor and the software subcontractor agree to their commitment to each other.

- Goal 3 :
The prime contractor and the subcontractor maintain ongoing communications.

- Goal 4 :
The prime contractor tracks the software subcontractor's actual results and performance against its commitments.

- KPA : Software Quality Assurance :

the purpose of Software Quality Assurance is to provide management with appropriate visibility into the process being used by the software project and of the products being build. Software Quality Assurance is an integral part of most software engineering and management processes.

- Goal 1 :
Software quality assurance activities are planned.

- Goal 2 :
Adherence of software products and activities to the applicable standards, procedures, and requirements is verified objectively.

- Goal 3 :
Affected groups and individuals are informed of software quality activities and results.

- Goal 4 :
Noncompliance issues that cannot be resolved within the software project are addressed by senior management.

- KPA : Software Configuration Management :

the purpose of Software Configuration Management is to establish and maintain the integrity of the products of the software project throughout the project's software life cycle. Software Configuration Management is an integral part of most software engineering and management processes.

- Goal 1 :
Software configuration management activities are planned.

- Goal 2 :
Selected software work products are identified, controlled, and available.

- Goal 3 :
Changes to identified software products are controlled.

- Goal 4 :
Affected groups and individuals are informed of the status and content of software baselines.

Key Practice

Because the 'goals' of a Key Process Area are still rather general, CMM indicates a way to asses if these goals are met. CMM describes each Key Process Area in terms of the Key Practices that contribute to satisfying the goals of a KPA. The Key Practices describe the infrastructure and activities that contribute most to the effective implementation of a KPA.

A Key Practice describes at the lowest level 'what' has to be done, but not 'how' it should be done. For a correct fulfillment of the goals of the KPA all its Key Practices have to be satisfied by the software development organization.

Key Practices are organized into five groups (processes). A group (process) is called 'common features' in CMM. The five groups are : 'Commitment to perform', 'Ability to perform', 'Activities performed', 'Measurement and analysis' and 'Verifying implementation'.

1. 'Commitment to perform' describes the actions an organization must take to ensure that the process is established and will endure. Typically this involves the codifying of organizational policies (in manuals) and senior management commitment to this policies.

2. 'Ability to perform' describes the preconditions that must exists in an organization to implement the process correctly. Typically this involves resources, organizational structures and training.

3. 'Activities performed' describes the roles and procedures necessary to implement the Key Process Area. Typically this involves establishing plans and procedures, performing the work, tracking it and taking corrective actions as necessary.

4. 'Measurement and analysis' describes the need to measure the process and analyzes the measurements. Typically this involves measurements that could be taken to determine the status and effectiveness of the 'Activities performed'.

5. 'Verifying implementation' describes the steps to ensure that the activities are performed in compliance with the process that has been established. Typically this involves reviews and audits by management and software quality assurance.

This thesis is concerned with the KPAs 'Software Configuration Management' and 'Software Project Planning' . For this reason only the Key Practices of these KPAs are described below. For a description of the other Key Practices the reader is referred to [CMM].

- Key Practices 'Software Configuration Management (SCM)' :

Commitment to perform

1. The project follows a written organizational policy for implementing software configuration management

Ability to perform

1. A board having the authority for managing the project's software baselines exists
2. A group that is responsible for coordinating and implementing SCM for the project exists
3. Adequate resources and funding are provided for performing the SCM activities
4. Members of the SCM group are trained in the objectives, procedures, and methods for performing their SCM activities
5. Members of the software engineering group are trained to perform their SCM activities

Activities performed

1. A SCM plan is prepared for each software project according to a documented procedure.
2. A documented and approved SCM plan is used as the basis for performing the SCM activities.
3. A configuration management library system is established as a repository for the software baselines
4. The software work products to be placed under configuration management are identified
5. Change requests and problem reports for all configuration items are initiated, recorded, reviewed, approved, and tracked according to a documented procedure.
6. Changes to baselines are controlled according to a documented procedure
7. Products from the software baseline library are created and their release is controlled according to a documented procedure
8. The status of configuration items is recorded according to a documented procedure
9. Standard reports documenting the SCM activities and the contents of the software baseline are developed and made available to affected groups and individuals
10. Software baseline audits are conducted according to a documented procedure

Measurement and analysis

1. Measurements are made and used to determine the status of the SCM activities

Verifying implementation

1. The SCM activities are reviewed with senior management on a periodic basis
2. The SCM activities are reviewed with the project manager on a periodic and event-driven basis
3. The SCM group periodically audits software baselines to verify that they conform to the documentation that defines them
4. The software quality assurance group reviews and/or audits the activities and work products for SCM and reports the results

- Key Practices 'Software Project Planning (SPP)' :

Commitment to perform

1. A software manager is designated to be responsible for negotiating commitments and developing the project's software development plan
2. The project follows a written organizational policy for planning a software project

Ability to perform

1. A documented and approved statement of work exists for the software project
2. Responsibilities for developing the software development plan are assigned
3. Adequate resources and funding are provided for planning the software project
4. The software managers, software engineers, and other individuals involved in the software project planning are trained in the software estimating and planning procedures applicable to their areas of responsibility.

Activities performed

1. The software engineering group participate on the project proposal team
2. Software project planning is initiated in the early stages of, and in parallel with, the overall project planning.
3. The software engineering group participates with other affected groups in the overall project planning throughout the project's life.
4. Software project commitments made to individual groups external to the organization are reviewed with senior management according to a documented procedure.
5. A software life cycle with predefined stages of manageable size is identified or defined.
6. The project's software development plan is developed according to a documented procedure
7. The plan for the software project is documented
8. Software work products that are needed to establish and maintain control of the software project are identified.
9. Estimates for the size of the software work product (or change of the size of software work product) are derived according to a documented procedure.
10. Estimates for the software project's effort and costs are derived according to a documented procedure.
11. Estimates for the project's critical computer resources are derived according to a documented procedure.
12. The project's software schedule is derived according to a documented procedure.
13. The software risks associated with the cost, resource, schedule, and technical aspects of the project are identified, assessed, and documented.
14. Plans for the project's software engineering facilities and support tools are prepared
15. Software planning data are recorded

Measurement and analysis

1. Measurements are made and used to determine the status of the software planning activities

Verifying implementation

1. The activities for software project planning are reviewed with senior management on a periodic basis
2. The activities for software project planning are reviewed with the project manager on both a periodic and event-driven basis
3. The software quality assurance group reviews and/or audits the activities and work products for software project planning and reports the results

SOFTWARE PROCESS MODELING IN SOCCA

student : H.G.Brugman
studentnr : 9105506
version : 0.10

3. SOCCA

3.1 Introduction

SOCCA, Specifications of Coordinated and Cooperative Activities, is a object-oriented process modeling language. It has been developed by the Software Engineering and Information Systems (SEIS) group of the Department of Computer Science of the University of Leiden. SOCCA is currently being refined and extended by the SEIS group.

Paragraph 3.2 of this chapter describes the SOCCA methodology. In paragraph 3.3 some basic SOCCA concepts are aggregated into higher level 'constructs' (3.3.1 until 3.3.5). Also in paragraph 3.3 some new 'constructs' are introduced (3.3.6 until 3.3.14). By using these 'constructs' in a SOCCA model, the modeling process can be speeded up.

Paragraph 3.4 describes two methods to construct a 'view' on a STD.

During the writing of this thesis, an 'integration' algorithm was developed. When using this algorithm SOCCA sub-models can be integrated into one bigger SOCCA model. The integration algorithm is described in paragraph 3.5.

3.2 Methodology

SOCCA models the static and dynamic aspects of a process and includes the human agents as objects in the model. SOCCA's focus is primarily on the modeling of the dynamic behavior of the process, including the communication between the different objects in the model.

The consecutive design steps in SOCCA are :

1. model the static structure
2. model the dynamic behavior
3. model the communication

3.2.1 Static structure

The static structure is modeled by a class diagram, using the Extended Entity Relationship technique (EER).
It is not unusual to construct four subdiagrams of the class diagram. The first one shows the classes, subclasses and aggregation associations. The second one shows the general associations, the third one shows the classes with their operations and attributes and the last one shows the 'uses' associations (the import-export diagram).

The first three subdiagrams are equivalent to the class diagrams as constructed by many other Object Oriented design methodologies. Like for example the 'Object Modeling Technique' [RUM] or the 'Unified Modeling Language' [UML] or [FOW].

The import-export diagram is SOCCA specific. This diagram identifies which operations are imported by which classes. Within the importing classes the importing operations are identified. This is done by the SOCCA specific binary association 'uses'. This association has the attribute 'import_list' that has as its domain a list of imported operations together with the operations that import them. The style guideline for this association is a solid line with an arrow at one end. The arrow indicates the exporting class.

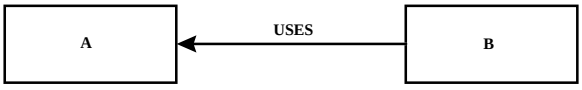


figure 3.1 example : import-export diagram

In the example above, some operation(s) of the class B are importing some operation(s) of the class A. The specification of the uses-association and its 'import-list' attribute is :

uses : imported operation imported by

operation_x operation_y

The import-export diagram is showing the communication between the classes at the highest level and is constructed as a step towards the constructing of the State Transition Diagrams in the next phase of the design process.

3.2.2 Dynamic behavior

The dynamic behavior is modeled by State Transition Diagrams, STD's. An STD is a nondeterministic finite automaton described by the 5-tuple $(Q, \Sigma, \delta, q_0, F)$. Q is the set of states (nodes), Σ is the input alphabet (set of labels on the transitions), δ is the set of transitions (edges), q_0 is the starting state and F is the set of final states.

The behavior of the objects of a particular class is modeled by two kinds of STDs, 'external' STDs and 'internal' STDs. The external STDs model the visible behavior of the objects. The internal STDs model the functionality of the objects (i.e. they model the operations of an object).

3.2.2.1 External STD

First an external STD is constructed. This STD shows the observable behavior of a class. It consists of a set of states (Q) an object of a class can be in. It also shows the possible sequences in which the exported operations of a class, as shown in the class diagram, can be started. I.e. it shows the possible sequences in which calls to the exported operations are serviced. The starting-sequences appear as the consecutive transitions in the STD. The transitions are labeled with the exported operations or are unlabeled. The transitions are the set δ. The exported operations are the input alphabet Σ. In standard SOCCA there is one external STD per class and every object of the class shows this behavior. The use of an STD to model the observable behavior of a class is done also by other Object Oriented design methodologies. Like for example the use of 'state charts' for this purpose in the 'Object Modeling Technique' [RUM] or the 'Unified Modeling Language' [UML] or [FOW].

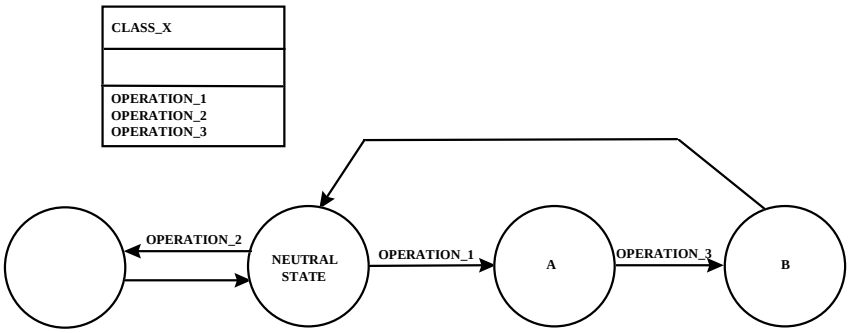


figure 3.2 example : External STD of class_x

Example : an object of the class_x (see figure) can service a call to operation_3 only when it is in state A. If the object arrives in state A and a call has been made to operation_3 at some earlier time, the object can service this call. The object can start operation_3 and can transit to state B. If the object arrives in state A and no call to operation_3 has been made yet, the object must wait in state A until such a call is made before it can proceed to state B.

3.2.2.2 Internal STD

Secondly the behavior of each operation of a class is shown in an internal STD. This amounts to one internal STD per operation in standard SOCCA. Consequently a class has as many internal STDs as it has operations. (NB : An object can have more internal STDs than the class it belongs to. This is caused by the fact that an object can execute multiple instances of the same internal STD concurrently. This phenomenon will be explained later on.)

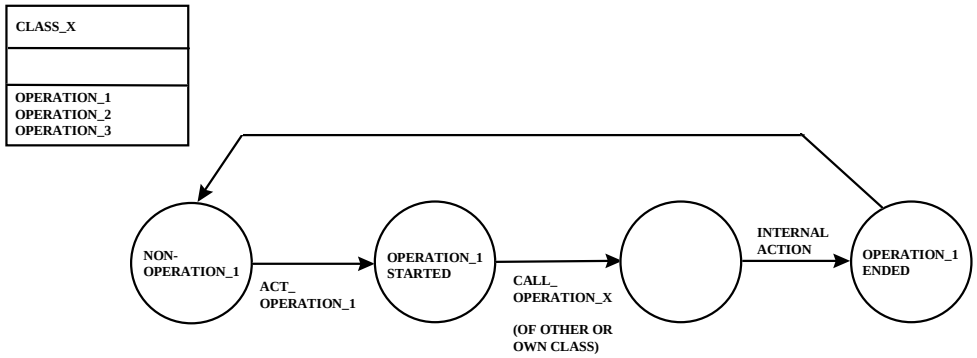


figure 3.3 example int-operation_1 : Internal STD of operation_1 of class_x

The name of an internal behavior is the name of the operation prefixed by 'int-'. In the state 'non-operation' the internal behavior is waiting to be started. The transition with label 'act-operation name' indicates the start of the execution of the STD. This transition is supposed to happen when the external STD of the class makes its transition labeled with 'operation name'; this will be enforced by the communication.

The behavior of an operation can be generally divided into two kinds of actions. It can call operations of other classes (or of its own class), and it can perform some 'internal action'.

The prefix 'call-' to an operation name 'x' means that the internal behavior calls the operation 'x'. This operation 'x' is either exported by another class or exported by its own class. A label without a prefix means that the behavior executes some internal action of its own without interaction with another object.

The meaning of the labels in SOCCA is the same as in UML. A transition-label in UML consists of an 'event'-part, a 'condition'-part and an 'operation'-part. Its syntax is 'event [condition]/action'. A label can be any combination of these three parts. The notation of an automatic transaction is '- [condition]/action' or '[condition]/action'. An automatic transition takes place 'automatically' after the activity of the state it leaves, is finished.

A transition with an event-part, leaving the current state, takes place (fires) when the corresponding event occurs. If the event occurs when the current state has no outgoing transition labeled with this event, the event is being ignored. If the transition has a condition-part, then the transaction can only take place if the condition is satisfied. If the transition has an action-part, then this action is performed when the transaction takes place.

The label 'act_operation_1' can be read as '[call to operation_1 has been made]/activate_operation_1' : the transition can take place (and eventually will take place) if the condition is satisfied. The action 'activate_operation_1' is performed when the transition takes place. The label 'call_operation_x' can be read as '-/call_operation_x' : the action 'call_operation_x' is performed when the transition takes place. The label 'internal_operation' can be read as '-/internal_operation' : the action 'internal_operation' is performed when the transaction takes place.

3.2.3 Communication

Communication between objects consists of calling exported internal operations of one class from within an internal operation of another (or the same) class. The called operations are the 'callees' and the internal operations performing the call are the 'callers'. The callers and the callees can be seen as threads of their respective objects.

This communication between objects is modeled using concepts of the 'Paradigm'-formalism. The Paradigm-formalism is used to describe parallel phenomena in general. A detailed description of this formalism can be found in [GRO]. Of the Paradigm concepts that are incorporated in SOCCA, that of the 'manager' process (described by an STD) and 'employee' process (also described by an STD) are described below. Employees are processes that communicate which each other and a manager is a process that synchronizes the communication between these employees.

In using the Paradigm-formalism all behavior STD's, internal and external, are seen as parallel processes. Conceptually this means that each internal and external STD is presumed to execute on its own dedicated processor.

3.2.3.1 Manager & Employee STD

Applying these concepts in a structured way to SOCCA, the external STD of a class becomes the manager STD of that class and the internal STD's become employee STD's.

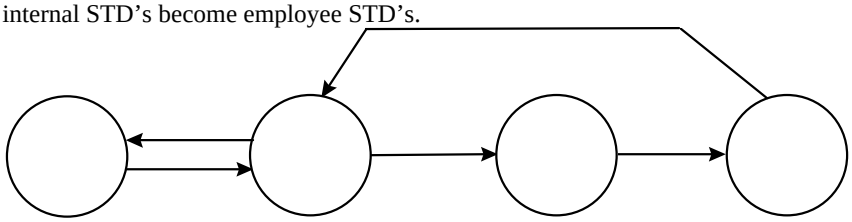


figure 3.4 example : Manager STD of class_x is the same as external STD of class_x

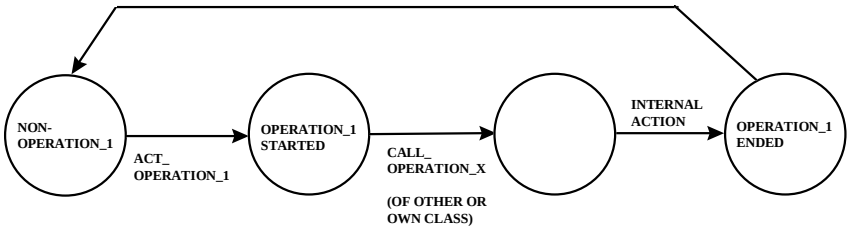


figure 3.5 example : Employee STD of operation_1 is the same as internal STD of operation_1

The manager STD synchronizes every two of its employees who need to communicate with each other (i.e. every caller-callee pair). The manager STD does this by allowing these two employees only to execute specific parts of their internal behavior STD at any one time. When at the end of this partial execution the two employees reach their respective 'synchronization points' the communication (call) takes places. Thereafter the manager STD allows both employees to proceed with the execution of the next part of their internal behavior STD.

3.2.3.2 Subprocess & Trap

These partial internal STDs which the manager prescribes to its employees are called 'subprocesses'. Such a partial STD is a temporary behavior restriction of that STD. A subprocess is decribed by at 3-tuple (Q', Σ', δ'). Q' is a subset of the set Q of the total internal behavior STD. Σ' is a subset of the set Σ of the total internal STD. δ' is a subset of set δ of the total internal STD.

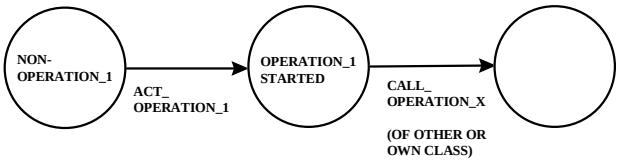


figure 3.6 example : Subprocess S1 of employee STD of operation_1

The synchronization points are called 'traps'. A trap is a subset of the set of states Q' of a subprocess. The states in a trap may not have transitions to states outside the trap. Once an employee enters a trap in a subprocess, it can not leave it as long as this subprocess remains the current behavior restriction.

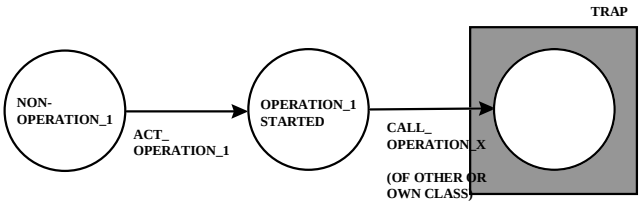


figure 3.7 example : Trap in subprocess S1 of employee STD of operation_1

Manager prescribes subprocesses

The manager STD prescribes in its states the subprocesses for all of its employees. The employees can only execute their currently prescribed subprocess. I.e they can only execute a part of their total behavior. The manager can make the transition to another state, where it prescribes other subprocesses, when the relevant employees have all entered their relevant traps. The employees then can execute the next part of their behavior, i.e their next subprocess. The states of the manager STD show the names of the prescribed subprocesses and the transitions are labeled with the names of the traps which the employees have to enter to cause the transitions to happen. So the traps are the conditions for the manager transitions.

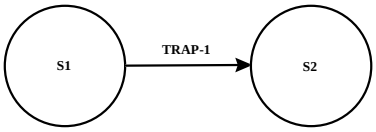


figure 3.8 example : manager STD prescibing subprocesses to its employee

The states in the trap of one subprocess must also be in the next subprocess for the switching from one subprocess to the next to work.

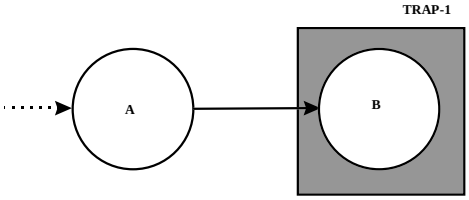


figure 3.9 example : employee in prescribed subprocess S1

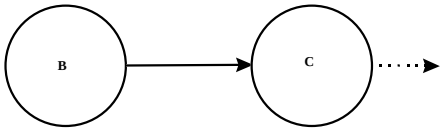


figure 3.10 example : employee in next prescribed subprocess S2

The number of employees that a manager STD of a class manages, varies. If every exported operation is called by a different caller, it is two times the number of exported operations. It can be more if there are operations that are being called by more than one caller. It can be less of some caller calls more than one exported operation.

An employee STD can have one or more managers. If an internal STD does not call any operations it has only one manager, namely the manager STD of its own class. If an internal STD performs one or more calls, the number of managers it has equals the number of different classes the called operations belong to plus one (the manager of its own class).

3.2.3.3 Intersection of subprocesses

When an employees has more than one manager it has more than one subprocess prescribed to it at any point in time. The actual subprocess (actual behavior restriction) the employee is executing is constructed by taking the intersection of all its prescribed subprocesses.

The trap(s) of the actual subprocess are constructed as follows. Suppose the number of prescribed subprocesses is n. Determine every possible combination of n traps, where every trap in the combination belongs to a different subprocess. The trivial traps of the subprocesses are also taken into account in this process. The trivial trap of a subprocess is the trap wich encompasses all states of that subprocess. Then take for every combination that was found the intersection of the traps in that combination. The intersection results are the traps of the actual subprocess.

The construction of the actual subprocess and its traps is illustrated by the next example.

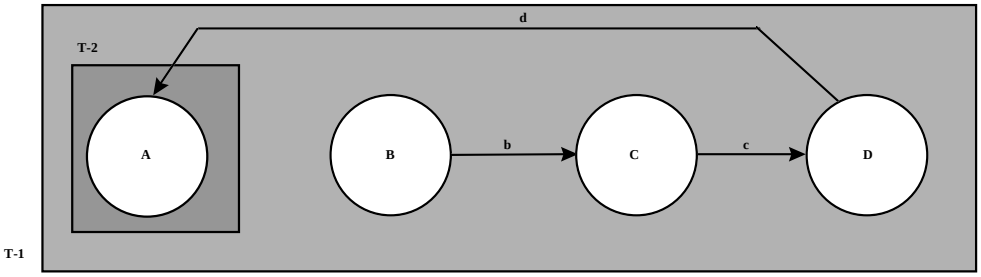


figure 3.11 example : prescribed subprocess S2 by manager 1

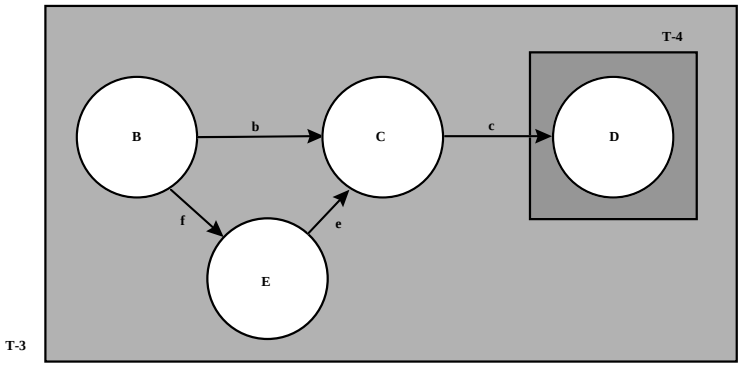


figure 3.12 example : prescribed subprocess S4 by manager 2

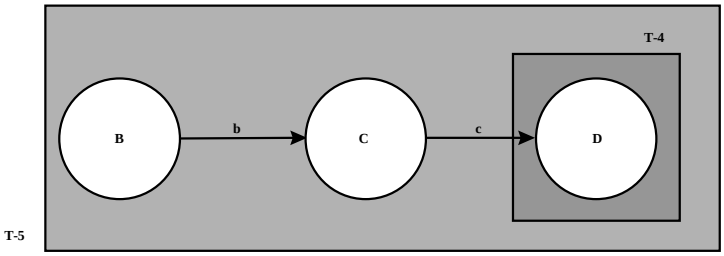


figure 3.13 example : actual behavior restriction and traps

Example :

Prescribed subprocess S2 consists of the states A, B, C and D and the transitions b, c and d. It has the traps T-1 (trivial trap) and T-2. Trap T-1 consists of the states A, B, C and D. Trap T-2 consists of the state A.

Prescribed subprocess S4 consists of the states B, C, D and E and the transitions b, c, e and f. It has the traps T-3 (trivial trap) and T-4. Trap T-3 consists of the states B, C, D and E. Trap T-4 consists of the state D.

The states of the actual subprocess (actual behavior restriction) are found by taking the intersection of the states of subprocess S2 and the states of subprocess S4. This results in the states B, C and D ($\{A,B,C,D\} \cap \{B,C,D,E\}$).
The transitions of the actual subprocess are found by taking the intersection of the transitions of subprocess S2 and the transitions of subprocess S4. This results in the transitions b and c ($\{b,c,d\} \cap \{b,c,e,f\}$).

The traps of the actual subprocess are found by first determining all possible combinations of traps from S2 and S4. This yields the combinations {T-1,T-3}, {T-1,T-4}, {T-2,T-3} and {T-2,T-4}. Secondly the intersection of the traps of each combination is determined.

For T-1 $\cap$ T-3 this results in T-5. T-1 has the states {A,B,C,D}. T-3 has the states {B,C,D,E}. The intersection of the states of both traps results in {B,C,D}. So the result is the trap T-5, the trivial trap of the actual subprocess.

For T-1 ∩ T-4 this results in T-4. T-1 has the states {A,B,C,D}. T-4 has the state {D}. The intersection of the states of both traps results in {D}. So the result is the trap T-4.

T-2 ∩ T-3 does not produce a trap. T-2 has the state {A}. T-3 has the states {B,C,D,E}. The intersection of the states of both traps results in the empty set.

T-2 ∩ T-4 does not produce a trap. T-2 has the state {A}. T-4 has the state {D}. The intersection of the states of both traps results in the empty set.

So the traps of the actual subprocess are T-4 and T-5.

3.3 Constructs

In SOCCA there are a certain number of often recurring combinations of syntactical elements. These are called 'constructs'. In a SOCCA model these constructs can either be modeled explicitly or alternatively they can be used on a higher aggregation level (without showing the underlying details) or they can simply be only referred to.

3.3.1 Activate-construct

The activate-construct describes the starting of an internal behavior STD by its manager STD. If the internal operation is called by another operation, then the act-construct is part of the caller_callee-construct. In case of autonomous behavior (no external caller) the act-construct is used 'stand-alone'. The act-construct only covers the callee. The caller behavior (e.g. whether the caller proceeds right after the call or waits for the result of the internal operation) is dealt with in the caller_callee-construct description.

Initially the manager is in its state 'neutral'. Here it prescribes the subprocess S1 for its internal employee.

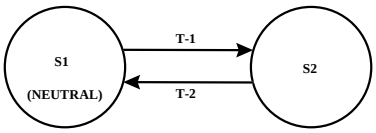


figure 3.14 example : act-construct, manager STD

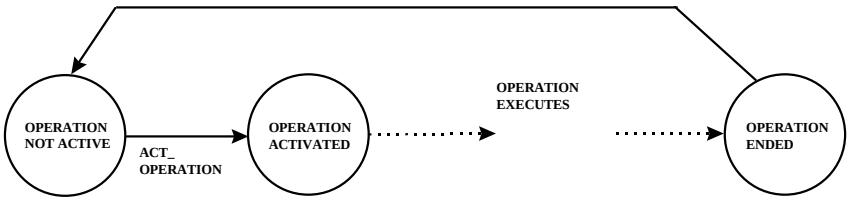


figure 3.15 example : act-construct, employee STD

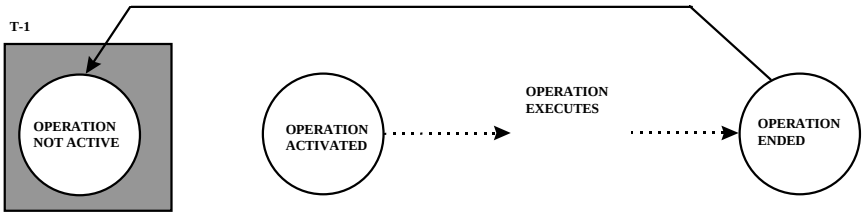


figure 3.16 example : act-construct, subprocess S1 of internal employee

When the employee has entered trap T-1 (is inactive, ready to be called) and the operation is called (i.e. the caller STD has entered a trap, say T-x, expressing this call), the manager can make the transition to its next state. In this state it prescribes the subprocess S2 for its internal employee.

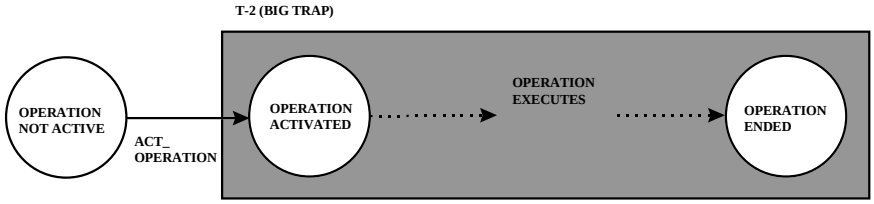


figure 3.17 example : act-construct, subprocess S2 of internal employee, big trap

The employee now can (and will) make the transition ‘act_operation’, i.e. it starts executing. When the employee now enters the trap T-2 (and also after the caller has entered some relevant trap, say T-y), the manager can make the transition back to its initial state. In general it arrives there before the internal STD has finished executing. In the initial state the manager prescribes again the subprocess S1 for its internal employee. The manager is ready to service a new call to the internal operation (and also calls to its other internal operations).

If the called operation must perform (part of) its actions before any other internal operation may be started, the transition of the manager back to the initial state must be delayed until these actions are performed. This is done by choosing the trap T-2 accordingly smaller (excluding the actions which must be performed).

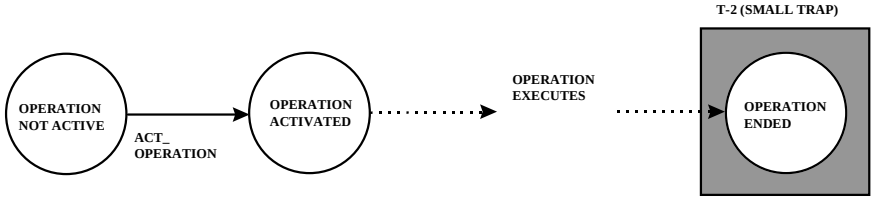


figure 3.18 example : act-construct, subprocess S2 of internal employee, small trap

When the trap T-2 is chosen smaller in subprocess S2, the subprocess S1 changes accordingly to reflect this. It has as its states only the state ‘operation non active’ plus the states that are in trap T-2 of the subprocess S2. In the event that the trap T-2 only encompasses the state ‘operation ended’, the matching subprocess S1 is shown in the next figure.

figure 3.19 example : act-construct, subprocess S1 of internal employee, matching S2 with small trap

3.3.2 Caller_Callee-construct

The caller_callee-construct describes the calling mechanism. The callee part is the starting of the called operation and is described by the act-construct. The caller part introduces two subprocesses and two traps in the calling employee.

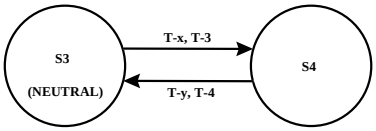


figure 3.20 example : caller_callee-construct, manager STD

The manager prescribes initially subprocess S3 for the calling employee, it is waiting for the call. When the caller enters its trap T-3, i.e. executes the call (and the callee is ready in its trap T-x) the manager can make the transition to the next state. Here it prescribes S4 for the caller, thereby allowing it to proceed in its next subprocess. This has the effect that the caller does not wait for the result of the called operation but proceeds right away after the manager has started the called operation.

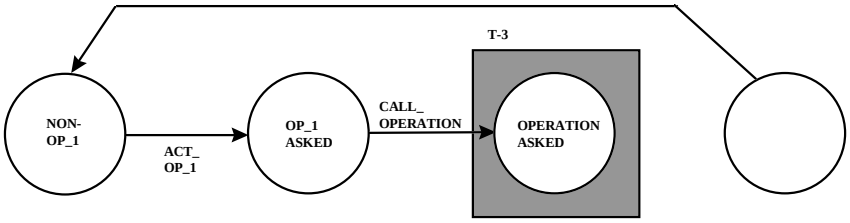


figure 3.21 example : caller_callee-construct, caller subprocess S3

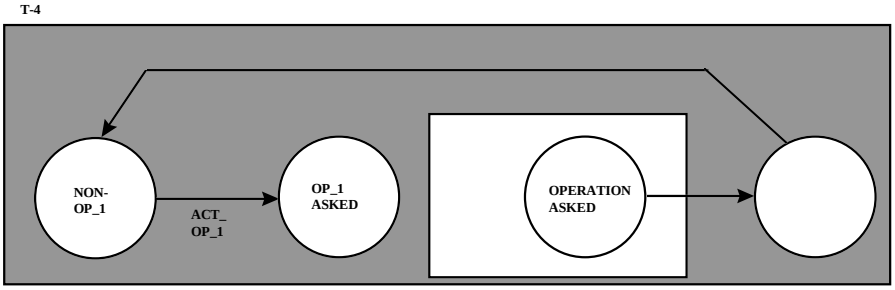


figure 3.22 example : caller_callee-construct, caller subprocess S4

The trap T-4 in the caller is chosen as big as possible to make the communication as asynchronous as possible. Also the trap T-3 can chosen bigger than one state, so as to allow more asynchronism before (and during) the execution of the called operation.

An important variant of the caller_callee-construct is the case when the caller has to wait for the return of a result of the callee. This 'caller waits'-construct can be modeled in two ways. These are called 1^e and 2^e variant of the 'caller waits-construct.

The caller waits-construct as described here is the 1^e variant of the caller waits-construct. The 2^e variant of the caller waits-construct is described in the paragraph 'waiting_caller_proceed-construct' of this chapter.

In the case of the 1^e variant of the caller waits-construct, the manager and the traps of the callee are a little more complicated then those of the normal caller_callee-construct ('normal' means that the caller does not wait).

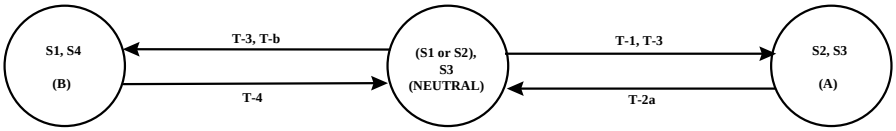


figure 3.23 example : caller_callee-construct, manager STD

figure 3.24 example : caller_callee-construct, callee subprocess S1

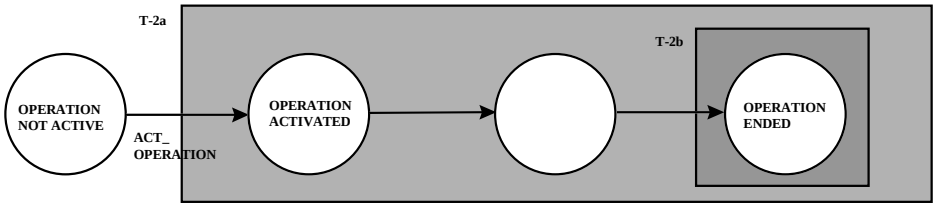


figure 3.25 example : caller_callee-construct, callee subprocess S2

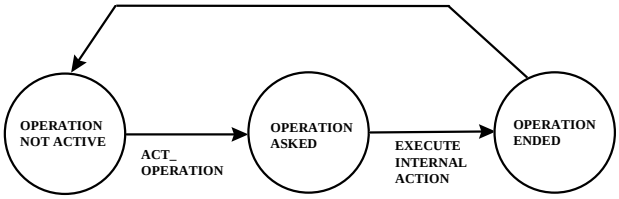
In its neutral state the manager prescribes initially subprocess S1 for the callee and S3 for the caller. When the caller enters its trap T-3, i.e. executes the call (and the callee is ready in its trap T-1) the manager can make the transition to the state A. Here it prescribes S2 for the callee and S3 for the caller (i.e the caller stays in subprocess S3). When the callee has entered trap T-2a the manager can transit back to its neutral state. Now it prescribes S2 (instead of S1) for the callee and S3 for the caller. If the callee has finished its action, i.e it has entered trap T2-b, the manager can transit to state B. Here it prescribes S1 for the callee and S4 for the caller (i.e. the caller is allowed to proceed). When the caller enters T-4, the manager can transit back to its neutral state. Here it prescribes S1 again for the callee.

So the manager prescibes in its neutral state either S1 or S2 for the callee, depending on which state it was in before entering its neutral state.

The act-construct and the caller_callee-construct can be used in different combinations. The trap sizes of the caller and the callee are chosen to suit the process that is modeled. Also the choice of the ‘caller wait’-variant or the ‘caller does not wait’-variant depends on the actual process.

3.3.3 Only internal action

When an internal STD calls no other operation during its execution it is modeled by the ‘only-internal_action’-construct. The only_internal_action-construct STD is a generic STD, with the states ‘operation not active’, ‘operation asked’ and ‘operation ended’ and the transition labels ‘act_operation’ and ‘execute internal action’. When this construct is used during actual modeling, the generic STD is shown in combination with a table that specifies the operation names and the internal actions that they perform.



OPERATION	INTERNAL ACTION
CH_ST_TO_SCB	UPDATE STATUS ATTRIBUTE TO SCB
CH_ST_TO_CCB	UPDATE STATUS ATTRIBUTE TO CCB
CH_ST_TO_REJECTED	UPDATE STATUS ATTRIBUTE TO REJECTED

figure 3.26 only_internal_action-construct : generic STD with specifying table

The subprocesses and traps in the only_internal_action-construct are according to the normal act-construct. When the internal action updates an attribute then the internal action has to be finished before the operation may be called again thereby guaranteeing the correct sequential updating of this attribute. This forces a small trap in the second subprocess.

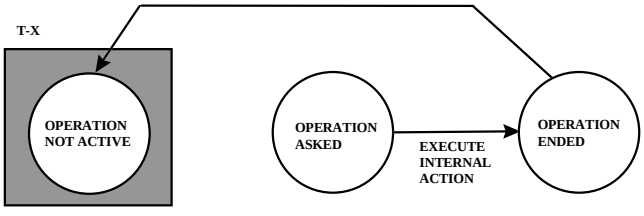


figure 3.27 only_internal_action-construct : subprocess SX

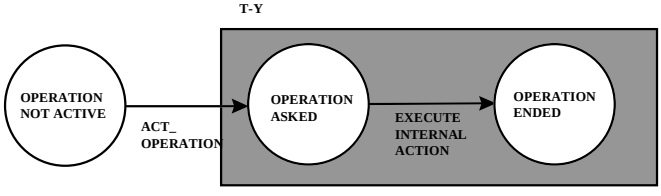


figure 3.28 only_internal_action-construct : subprocess SY, large trap

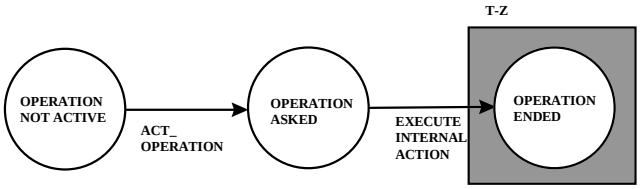


figure 3.29 only_internal_action_construct : subprocess SZ, small trap

3.3.4 No-operation (nop)

When an operation has only an internal action which performs no function, the operation is called a ‘no-operation’ (nop).

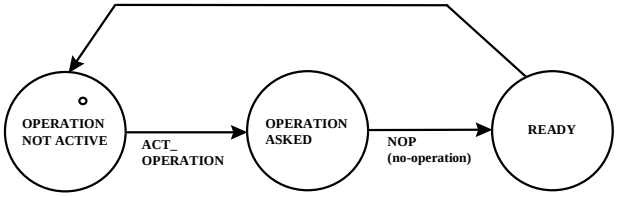


figure 3.30 no-operation : internal STD

The sole purpose of such an operation is to enable the corresponding transition of the external STD within the existing SOCCA framework.

3.3.5 Autonomous behavior

When an internal behavior of a class is initiated by the object itself instead of being called from the outside, this is called ‘autonomous behavior’ by the class. This can be interpreted as a special case of the caller_callee-construct. The internal STD is started with only the callee part of the caller_callee-construct (this being the act-construct). The caller part of the caller_callee-construct is simply left out.

3.3.6 Consolidated Prescribed Subprocesses & Traps Logical Formula

The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps.

When modeling a larger process, the convention to show all the prescribed subprocesses and all the traps in the actual figure of the manager STD can no longer be followed. The reason is that there is simply not enough space in the figure. Therefore a shorthand notation is used. The states of the manager STD are provided with the text CPS1, CPS2 etc. The transitions of the manager STD get the labels TLF1, TLF2 etc.

The notation CPSx in the STD stands for Consolidated Prescribed Subprocesses and is a set of CCx’s. The notation CCx in the STD stands for ‘Caller(s)-Callee’ and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for ‘Traps Logical Formula’ and describes the combination of traps that enables the transition.

The CPSs, CCs, and TLFs are then defined in the text following the figure of the manager STD.

This idea can be taken one step further. Since the manager STD is always equal to the external STD, there is no need to make a separate figure of the manager STD. The external STD now ‘doubles’ as the manager STD.

The CPSx’s now get the same name as the state for which they are valid. E.g. if ‘disc_starting_pr_count_two_phase_ ended’ is the name of a state in some external STD, it will be also the name of the CPS for that state in the manager STD. The TLF-x’s will get the same name as the operation that is started when the transition is taken. E.g. if ‘pr_count_two_phase_ ended’ is the name of the operation that is started (indicated in the external STD), it is also the name of the TLF that is guarding the transition in the manager STD. Transitions with no TLF (with no name in the

external STD) are automatic transitions (unless otherwise indicated in the TLF-definitions). Traps that are not mentioned in the TLF are don't cares.

The name of CCx's will include the full names of the caller and callee operations separated by a tilde (~) character. E.g. the CC of the caller operation 'cu_project_life_cycle' and the callee 'pr_project_life_cycle' will be named 'cu_project_life_cycle ~ pr_project_life_cycle'.

3.3.7 Subprocess and Trap-naming conventions

Subprocesses and traps will be given as much of their full name as is necessary to avoid ambiguity. This full name is constructed with the 'dot'-notation.

E.g. 'cu_project_life_cycle.S1' is subprocess S1 of the operation 'cu_project_life_cycle'. If there are more subprocesses with the same name prescribed to an operation, the manager prescribing each subprocess will be added. E.g. 'cu_project_life_cycle.S1_wrt_project'. This is the subprocess S1 with respect to (wrt, prescribed by) the manager of the class 'project'. In the same way the naming of the traps is handled. E.g. 'cu_project_life_cycle.S1.T1_wrt_project' is the trap T-1 prescribed by the 'project' manager STD in the subprocess S1 (which is also prescribed by the 'project' manager STD) of the operation 'cu_project_life_ cycle'. If the class of an operation is not clear, the class will be prefixed to the operation name. E.g. 'customer.cu_project_ life_cycle' is the operation 'cu_start_project_life_cycle' of the class 'customer'.

3.3.8 Multiplicity of concurrent STDs

Each class has one external STD and one or more internal STDs. The external STD describes the sequence in which the operations of the class can be started. The internal STDs describe the behavior of the operations of the class. These external and internal STDs are on 'class'-level. They are 'templates' that are used by the objects of that class.

An object has an external STD (equivalent to the class-external STD) and internal STDs (equivalent to the class-internal STDs). These external and internal STDs are on 'object'-level.

As already mentioned in the paragraph 'communication' of this chapter, all behavior STD's, internal and external, are seen as parallel processes. Conceptually this means that each internal and external STD (on the 'object'-level) is presumed to execute on its own dedicated processor.

When an object is created, its external STD starts executing. When a call to an operation of an object is serviced (by the external STD), the appropriate internal STD starts executing.

It is possible for an object to have multiple instances of the same internal STD executing concurrently. It is also possible for an object to have multiple instances of its external STD executing concurrently.

3.3.8.1 Internal STDs

When an internal STD is executing, and another call (of the operation it represents) is about to be serviced (by the external STD) then there are two possibilities. What happens depends on the 'multiplicity of concurrent executing STD instances' of that internal STD.

a. The 'multiplicity of concurrent executing STD instances' is zero or one. In this case the call is not serviced. It will only be serviced when the current instance of the internal STD finishes its execution.

b. The 'multiplicity of concurrent executing STD instances' is zero or more. In this case the call is serviced. Another instance of the internal STD starts executing. This instance will execute concurrently with the first instance.

The 'multiplicity of concurrent executing STD instances' is indicated in the 'non-active' state of the internal STD. A multiplicity of zero or more is indicated by solid circle inside the non-active state. A multiplicity of zero or one is indicated by hollow circle inside the non-active state. The multiplicity can also be a definite number. E.g. there can be at most 5 instances executing concurrently. In this case the number (5) is indicated next to the solid circle in the non-active state.

The default multiplicity of an internal STD is zero or more. An internal STD with no multiplicity indication in its starting state has this default multiplicity.

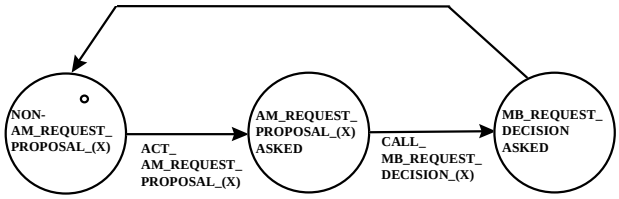


figure 3.31 example : internal STD, multiplicity of concurrency 'zero or one'

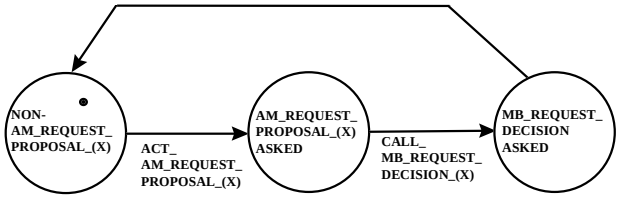


figure 3.32 example : internal STD, multiplicity of concurrency 'zero or more'

3.3.8.2 External STDs

Normally only one instance of an external STD will be executing. This is indicated by the hollow little circle in the start state of the external STD. Since the multiplicity of zero or one is the default multiplicity of an external STD, the hollow circle-indication is normally not shown in an external STD. If the multiplicity of concurrency is zero or more, a solid little circle is shown in the start state of the external STD.

The rule for concurrently executing external STDs is as follows. If no instance of the external STD is currently executing, and the operation labeling the (a) transition leaving the start state is called, the external STD starts executing (and it starts the called operation). If there is currently executing an instance of the external STD and the operation labeling the (a) transition leaving the start state is called, two possiblities exist.

a. The 'multiplicity of concurrent executing STD instances' is zero or one. In this case nothing will happen.

b. The 'multiplicity of concurrent executing STD instances' is zero or more. In this case another instance of the external STD starts executing (and the called operation is started (if possible)). This instance will execute concurrently with the first instance. Both external STDs are valid for the object at the same time.

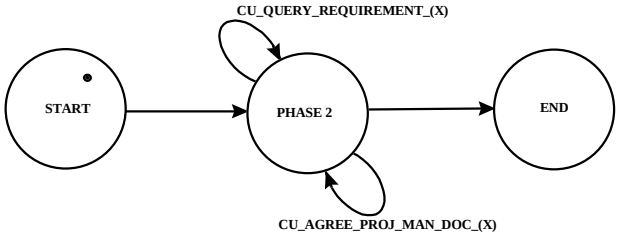


figure 3.33 example : external STD, multiplicity of concurrency 'zero or more'

3.3.9 Simultaneous_call-construct

It is possible to model one-to-many communication in SOCCA by introducing the 'simultaneous_call'-construct. The prefix for the transition label in the internal STD is 'sim_'. This indicates the simultaneous calling of the operation 'operation' of x objects of the same class (or of different classes).

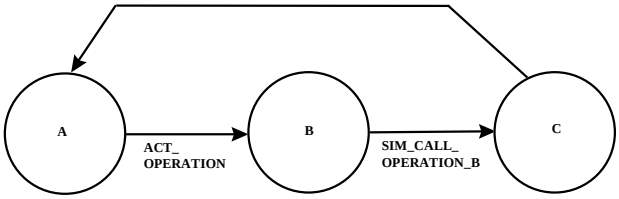


figure 3.34 simultaneous_call : internal STD of an operation that places a sim_call

The functioning of a sim_call will be explained in an example. In the example the internal operation of the above figure places a sim_call. The sim_call will be to the operation ‘operation_b’ of three other objects. These objects are of the same class and have the object-identification ‘obj9’, ‘obj21’ and ‘obj137’.

First the internal operation of the caller is shown with an ‘exploded’ view of the states ‘B’ and ‘C’.

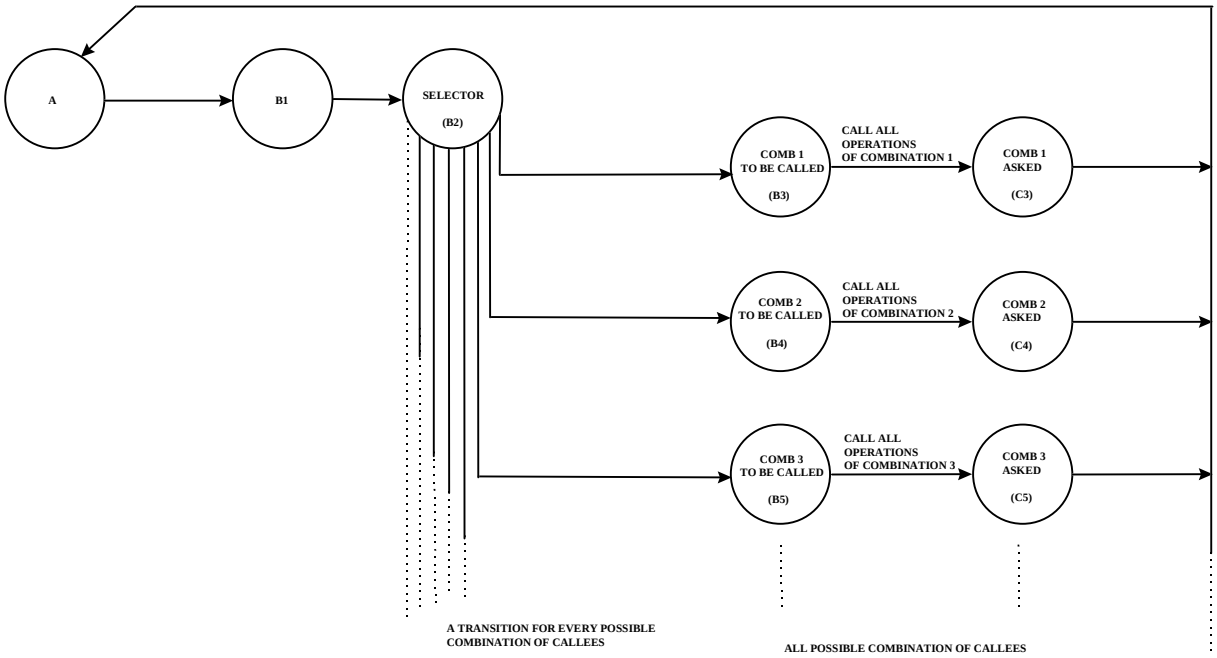


figure 3.35 simultaneous_call : internal STD of an operation that places a sim_call (exploded view)

The state ‘B’ has the sub-states ‘B1’, ‘B2’ (the ‘selector’ state) and ‘B3’ until ‘Bn’ (‘combination x to be called’-states). There are as many ‘combination x to be called’-states as there are possible combinations of calls.

The state ‘C’ has the sub-states ‘C3’ until ‘Cn’ (‘combination x asked’-states). There are as many ‘combination x asked’-states as there are possible combinations of calls.

When the internal STD arrives in state ‘B1’ it transits to state ‘B2’, the selector state. Here it determines which combination of calls it has to perform. It then goes to either ‘B3’ or ‘B4’ or .. or ‘Bn’ according to the combination to be called. From there it transits to the corresponding ‘C3’ until ‘Cn’ state. When it makes the transition it calls all the operations in the combination.

In this example it is assumed that ‘combination 3’ consists of ‘obj9’, ‘obj21’ and ‘obj137’. So in the example the internal STD of the caller goes from the selector state to the ‘B3’ state. It then makes the transition to the ‘C3’ state. While making the transition it calls ‘obj9.operation_b’ and ‘obj21.operation_b’ and ‘obj137.operation_b’.

Because of the fact that the internal STD of the caller can call the operation ‘operation_b’ of all possible objects that export that ‘operation_b’, the internal STD of the caller is an employee of all the manager STDs of the exporting objects. All these manager STDs prescribe subprocesses to the internal STD of the caller. The actual subprocess of the internal STD of the caller is (at any point in time) the intersection of all these prescribed subprocesses.

On a higher level this results in the ‘normal’ caller subprocesses S3 and S4 of the caller_callee-construct. Also the ‘normal’ traps T-3 and T-4 are the result of the intersection.

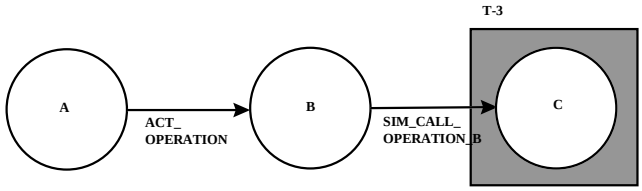


figure 3.36 simultaneous_call : ‘normal’ subprocess S3 and trap T-3 for the caller operation

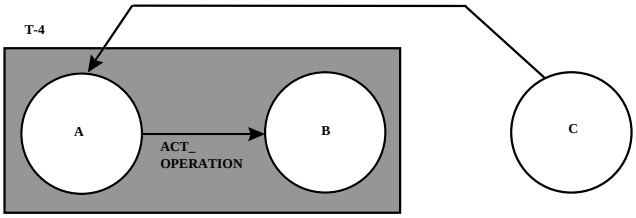


figure 3.37 simultaneous_call : ‘normal’ subprocess S4 and trap T-4 for the caller operation

To see that this is indeed the case, the subprocesses S3 and S4 will be looked at with the states ‘B’ and ‘C’ in an exploded view. The subprocesses prescibed by the manager STD of ‘obj21’ will be looked at in detail. It was assumed that ‘obj21’ was part of combination 3. It is further assumed that ‘obj21 is part of combination 1. Also it is assumed that ‘obj21’, ‘obj9’ and ‘obj137’ are not part of combination 2. The other possible combinations (4 until n) are not part of the example.

When the sim_call has not yet been made, then the manager STD of object x prescribes the subprocess S3_wrt_objx.
In this subprocess S3_wrt_objx some of the states ‘combination asked’ are traps. Namely precisely those states ‘combination asked’ for which ‘objx’ is part of that combination.

The subprocess S3_wrt_obj21 prescribed by the manager STD of ‘obj21’, which is part of combination 1 and combination 3, is shown in the next figure. It has two traps T-3_comb1_wrt_obj21 and T-3_comb3_wrt_obj21.

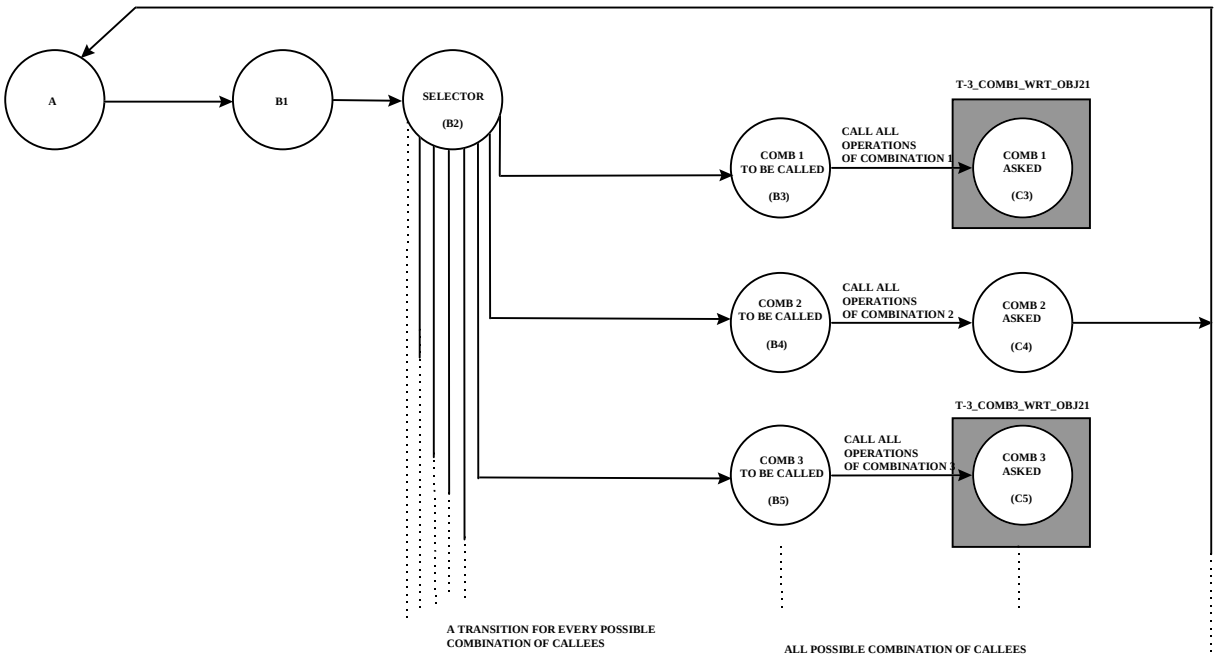


figure 3.38 simultaneous_call : prescribed subprocess S3_wrt_obj21

The sim_call in this example is a sim_call to ‘obj9’, ‘obj21’ and ‘obj137’. That is to say combination 3. After this sim_call has been made, and after obj21 has serviced it, the manager STD of obj21 will prescribe S4_wrt_obj21.

The manager STD of obj21 knows that the sim_call was for combination 3, because the caller STD has entered the trap T-3_comb3_wrt_obj21. The subprocess S4_wrt_obj21 that the manager STD of obj21 now prescribes contains the state ‘comb3 asked’ (‘C3’) plus the states ‘A’, ‘B1’ until ‘Bn’. It has the trap T4_wrt_obj21, which contains the states ‘A’, ‘B1’ until ‘Bn’.

N.B. if the caller would have entered ‘T-3_comb1_wrt_obj21 (if it made the sim_call to combination 1), then the manager STD of obj21 would have prescribed another S4_wrt_obj21. Namely instead of the state ‘comb3 asked’ it would contain the state ‘comb1 asked’.

The next figure shows the prescibed subprocess S4_wrt_obj21 when the sim_call has been made to combination 3.

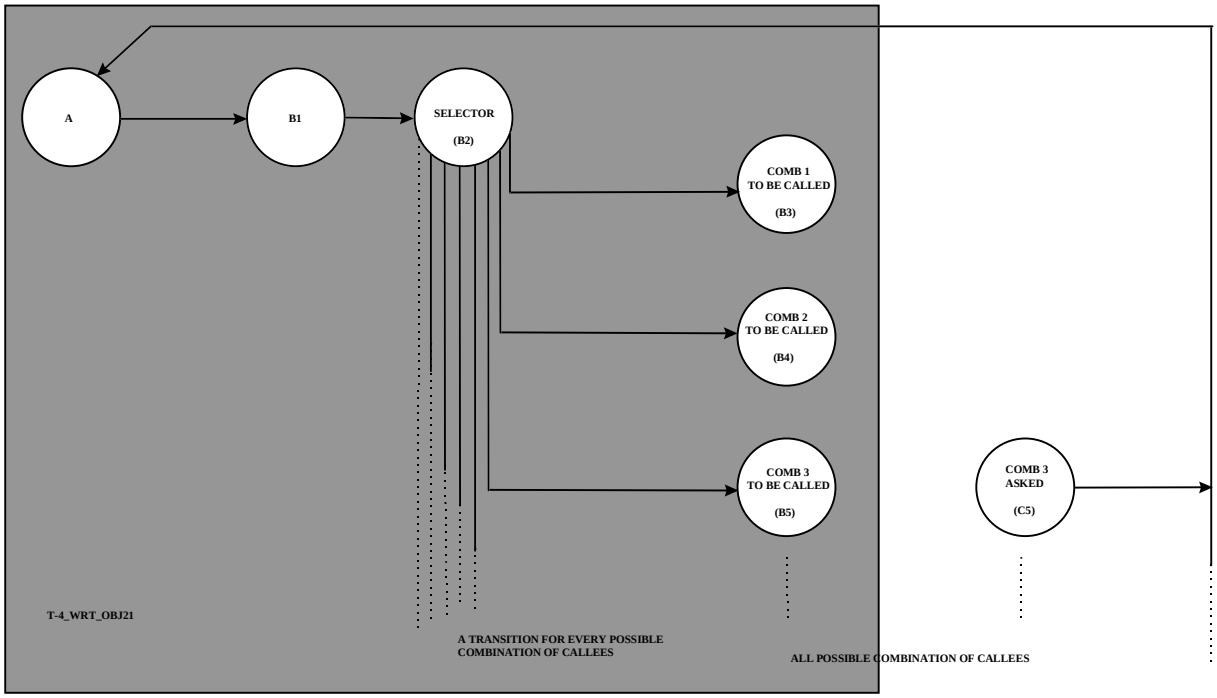


figure 3.39 simultaneous_call : prescribed subprocess S4_wrt_obj21 after sim_call to combination 3

Now the prescribed subprocesses by the manager of one object (obj21) are known, the result of the intersection of all the prescribed subprocess can be shown.

Going back to the point in the example where the sim_call has not been placed. The manager STD of obj21 prescribes S3_wrt_obj21. All other manager STD (of all other objects) also prescribe an equivalent subprocess S4_wrt_objx. When the intersection of all these subprocesses is taken, the actual subprocess as shown in the figure below is the result. All the states ‘combination x asked’ are now traps. The sim_call can take place.

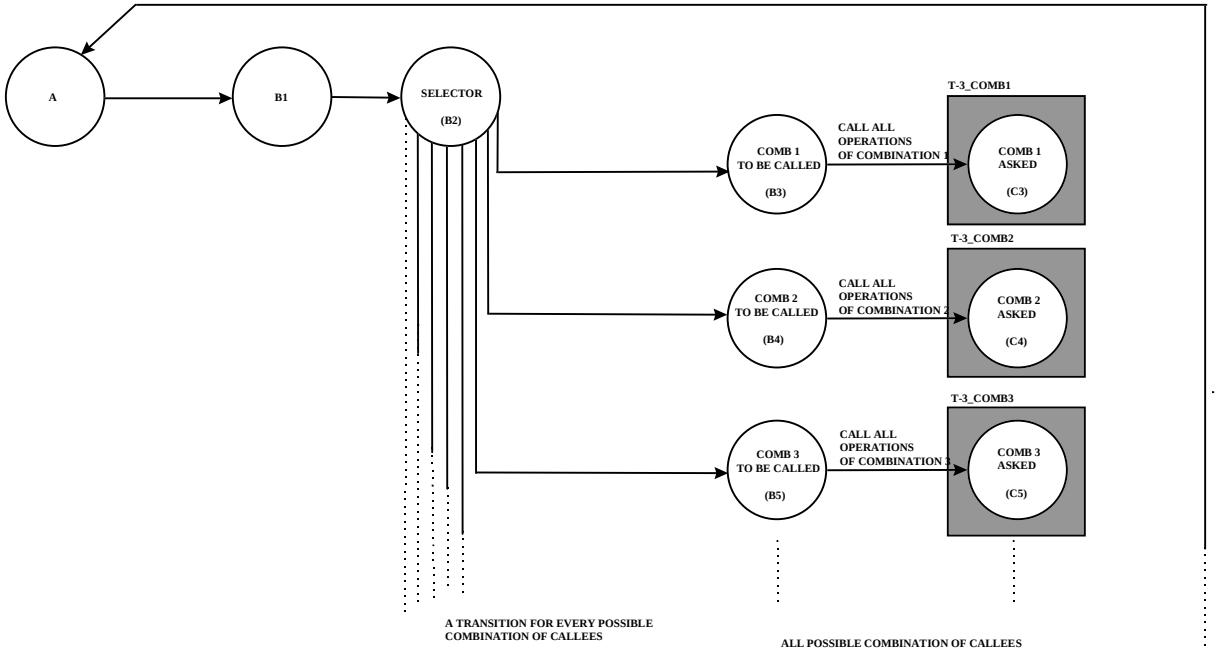


figure 3.40 simultaneous_call : actual subprocess S3

On a higher level this actual subprocess S3 is indeed the ‘normal’ caller subprocess S3 of the caller_callee-construct. The traps T-3_comb1 until T-3_combn are, on a higher level, the ‘normal’ trap T-3 of the caller S3 subprocess.

Now the sim_call takes place. The sim_call in this example is a sim_call to ‘obj9’, ‘obj21’ and ‘obj137’. That is to say combination 3. The three managers STD (of ‘obj9’, ‘obj21’ and ‘obj137’) will not react all at the same time. For the example it is assumed that the manger STD of obj21 reacts first. It will prescribed S4_wrt_obj21. The other two managers still prescribe S3_wrt_obj9 and S3_wrt_obj137. All other manager STD, of objects not in combination 3, will still prescribe their S3_wrt_objx.

The actual subprocess of the caller internal STD is again the result of the intersection of all these prescribed subprocesses. It is shown in the next figure.

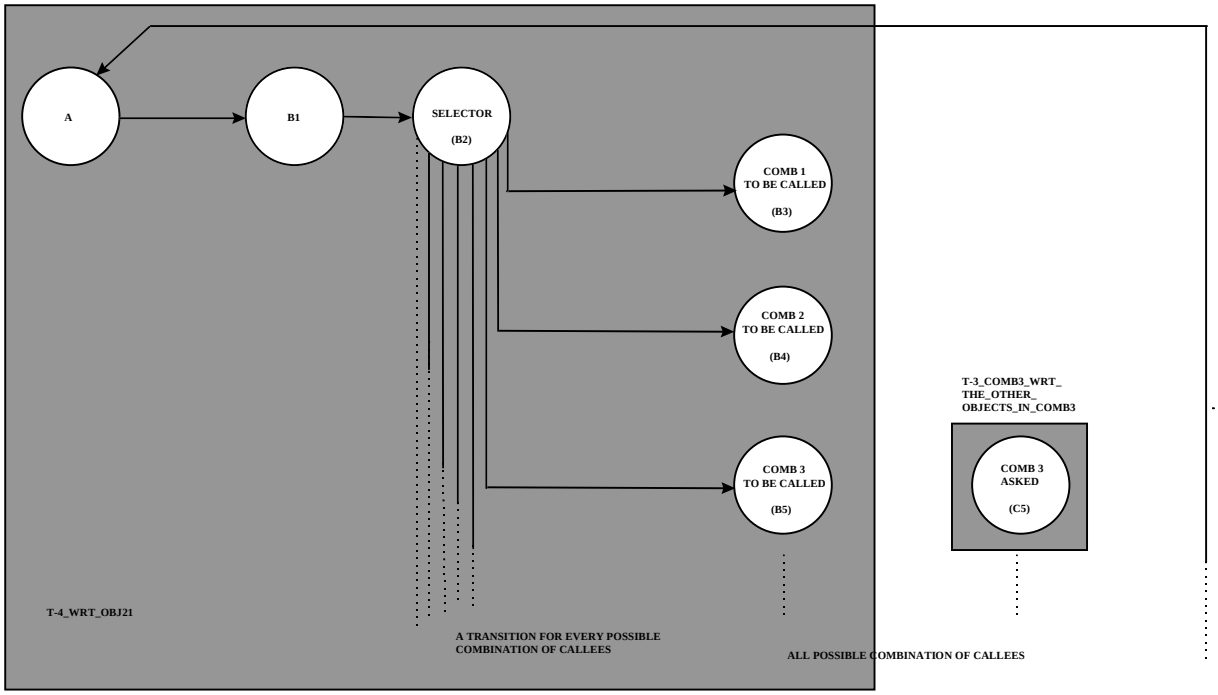


figure 3.41 simultaneous_call : intermediate actual subprocess, after manager of obj21 has reacted

The actual subprocess is an ‘intermediate’ one. The manager STD of obj21 prescribes S4_wrt_obj21, so trap T-4_wrt_obj21 is in the actual subprocess. The managers of obj9 and obj137 still prescribe S3_wrt_obj9 and S3_wrt_obj137, so trap T-3_comb3_wrt_other_objects_in_comb3’ is also in the actual subprocess. The internal STD of the caller has placed the sim_call. It is in the state ‘comb3 asked’. It can not leave this state (yet). It has to wait until the managers of obj9 and obj137 have serviced the sim_call also.

This intermediate actual subprocess is valid until both the manager of obj9 and of obj137 have serviced the sim_call and have prescribed S4_wrt_obj9 and S4_wrt_obj137.

When this has happened, the intersection of the then prescribed subprocesses will result in a new actual subprocess. This (new) actual subprocess is shown in the next figure.

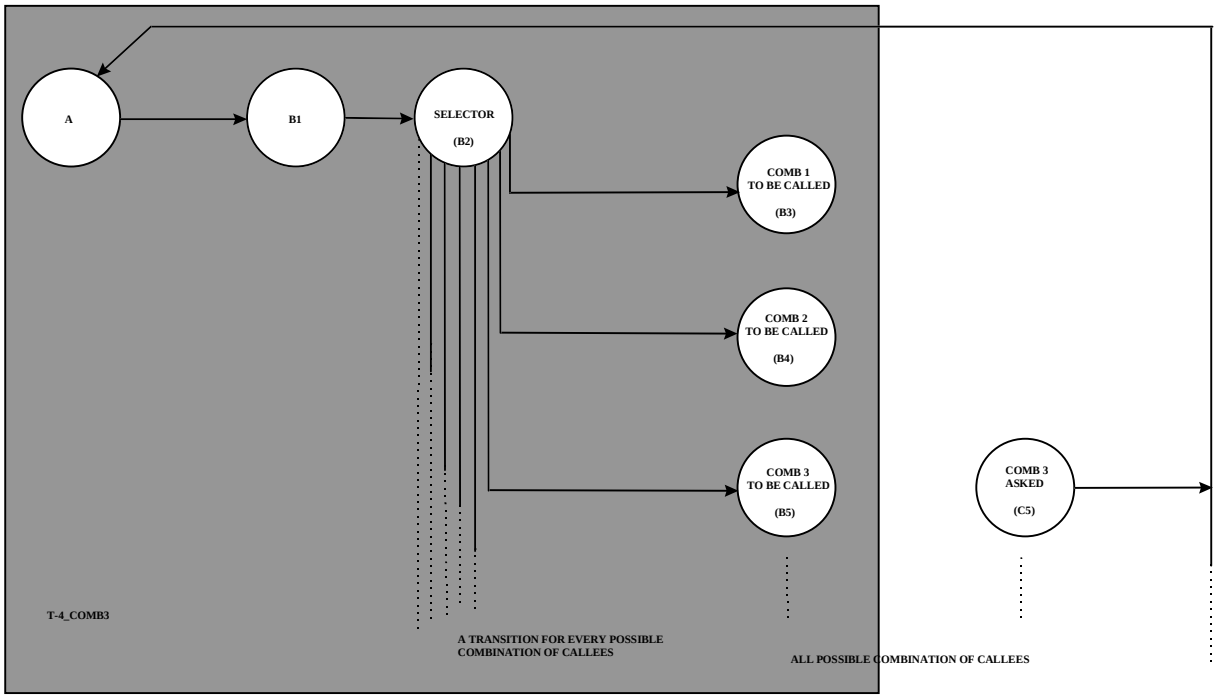


figure 3.42 simultaneous_call : actual subprocess 4

On a higher level this actual subprocess S4 is indeed the ‘normal’ caller subprocess S4 of the caller_callee-construct. The trap T-4_comb3 is, on a higher level, the ‘normal’ trap T-4 of the caller S4 subprocess.

Now the internal STD of the caller can leave the state ‘comb3 asked’. When it does so and enters the trap T-4_comb3, the three manager STDs (of ‘obj9’, ‘obj21’ and ‘obj137’) will again prescribe the subprocesses S3_wrt_obj9, S3_wrt_obj21 and S3_wrt_obj137. All other managers (of objects not in combintion 3) are still prescribing their S4_wrt_objx subprocesses. So, at this point the actual subprocess is again S3. The internal STD of the caller is ready again to place another sim_call. The example has come full circle.

It can readily be seen that every ‘normal’ call in an internal STD is a special case of a simultaneous call. In the selector state the ‘combination’ consisting of only that one call is chosen. Because a call to one object is the most frequent occurence of the simultaneous call, the prefix ‘sim_’ is omitted in this case.

3.3.10 Discriminator-construct

It is possible that one internal operation can be called by more than one caller. This is handled in SOCCA by a ‘discriminator’-construct in the external (and manager) STD. The prefix for the state name in the external STD is ‘disc_’.

This indicates that the state can discriminate between more than one caller object. The external STD can be described (at a higher abstraction level) by the following figure.

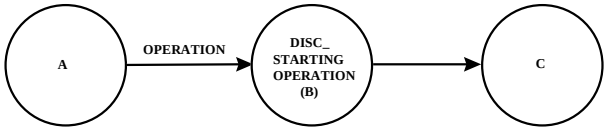


figure 3.43 discriminator_state : external STD

If there is only one caller of the operation ‘operation’ (see figure above) when the STD is in state A, then this caller is serviced. When there is more than one caller, then only one of these callers is serviced. The other callers have to wait until the STD returns again in state A.

The state A of the external STD (see figure below) can refined in the substates A1 and A2 (the ‘discriminator’ state). The state B (‘disc_starting_operation’) can be refined in the substates B1 until Bn (‘starting operation’). There are as many substates (B1, B2, .., Bn) as there are possible callers.

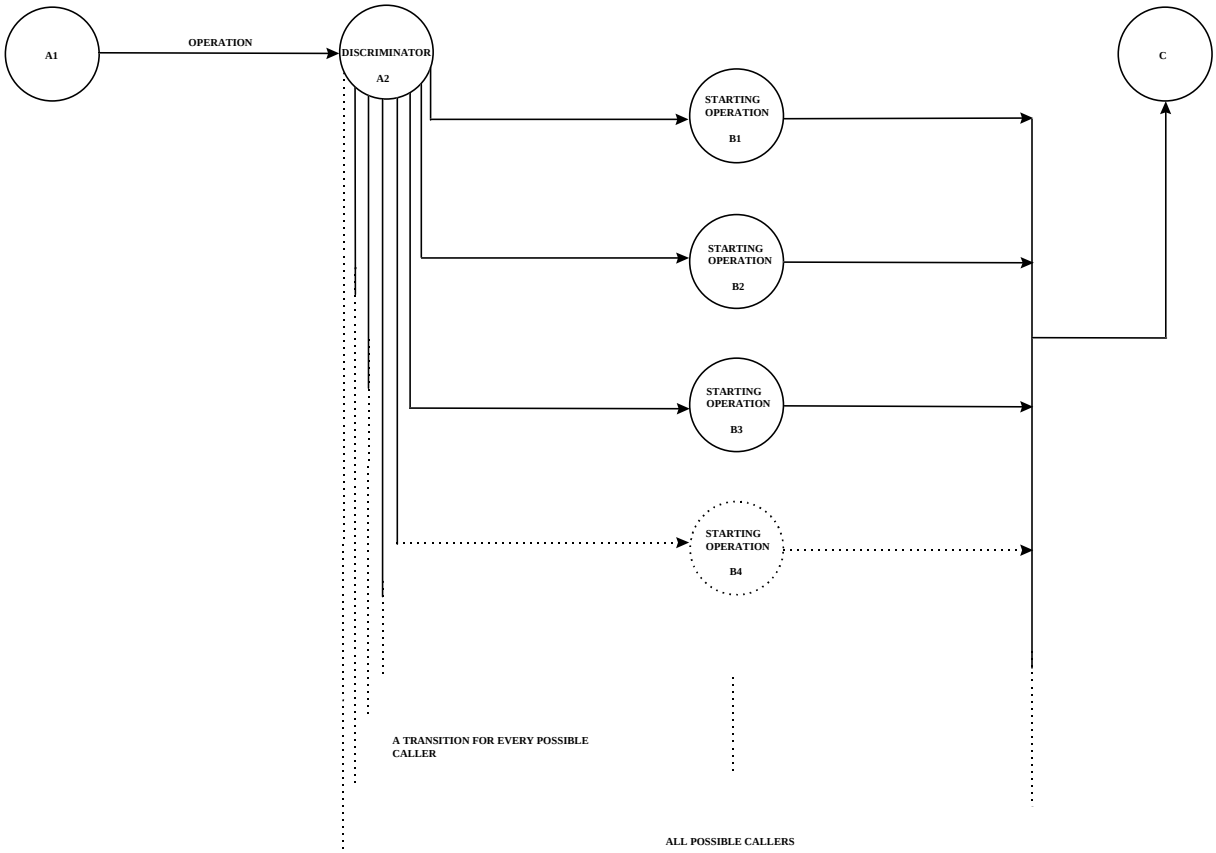


figure 3.44 discriminator_state : external STD, refined

The substates B1 until Bn are all ‘starting’-states in which the internal operation ‘operation’ is started. The discriminator state is a ‘switching’ state. From the discriminator state exits a transition to every substate B1 until Bn. Which transition is taken depends on the caller. Every caller corresponds with one particular transition.

The manager STD containing a discriminator state is manager of all possible calling objects. In servicing a call the manager will allow only one caller to proceed with its next behavior restriction. All other callers stay in their current behavior restriction. The manager can be described (at a higher abstraction level) by the following figure (this is an example of ‘caller does not wait’).

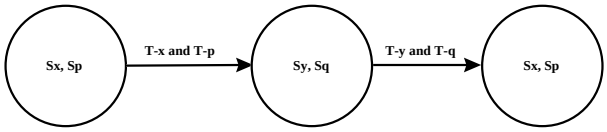


figure 3.45 discriminator_state : manager STD

The states of this manager STD can be refined in the same way as those of the external STD. Suppose that the callee has two behavior restrictions. The first behavior restriction is Sx. When the callee is in trap T-x it can be started (again). The second behavior restriction of the callee is Sy. Trap T-y indicates that the callee is started. Suppose that all callers also have two behavior restrictions. The first behavior restriction of all callers is Sp. They enter the trap T-p after executing their call to the operation 'operation'. The second behavior restriction of all callers is Sq. The entering of the trap T-q indicates that a caller is proceeding with it second behavior restriction.

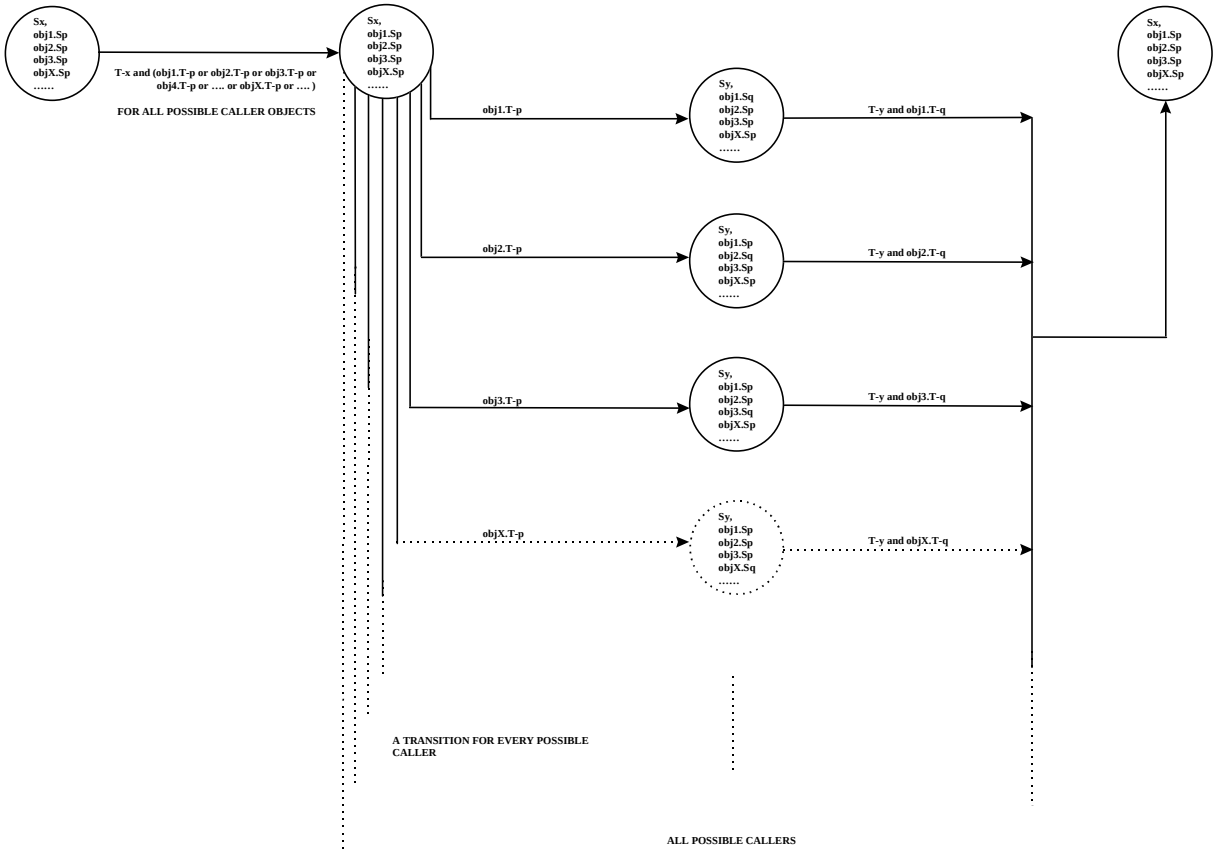


figure 3.46 discriminator_state : manager STD

Initially in state 'A1' the manager prescribes Sx for the internal operation and obj1.Sp, obj2.Sp, obj3.Sp, …, objxX.Sp (call detection) for all the possible callers. The manager can make the transition to the discriminator state when the internal operation has entered T-x and some object has made a call (i.e if [T-x and (obj1.T-p or obj2.T-p or obj3.T-p or obj4.T-p or ……)] is true). In the discriminator state the manager prescribes still Sx for the internal operation and obj1.Sp, obj2.Sp, obj3.Sp, …, objxX.Sp for the callers. Suppose object 3 has made the call. The manager then transits out of the discriminator state taking the transition labeled with the trap obj3.T-p. It arrives in the 'starting'-state B3. Here the manager prescribes Sy for the internal operation thereby starting it. It further prescribes obj1.Sp, obj2.Sp, obj3.Sq, obj4.Sp, …., objXSp for the calling objects. So it allows the calling object to proceed with its next subprocess while not changing the prescribed subprocesses of the not-calling objects. When the internal operation has entered trap T-y and the caller has entered its trap obj3.T-q the manager can transit to the state 'C' where the same subprocesses as in state 'A1' are again prescribed.

It can readily be seen that every 'normal' starting state in an external STD is indeed a discriminator state. Because the discriminating between different calling objects belonging to the same class is the most common occurrence of the discriminator state, the prefix 'disc_' is omitted in this case. But when the calling objects belong to different classes the prefix 'disc_' is applied to the state name. So a state named 'disc_..' discriminates explicitly between callers belonging to different classes as wel as implicitly between calling objects belonging to the same class within those different classes.

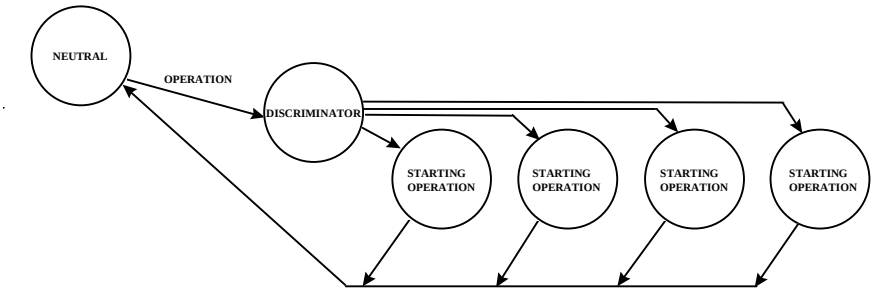


figure 3.47 example : manager STD discriminating between callers of 4 different classes

The figure above shows the explicit discriminating between callers of 4 different classes. There is a starting state per class. Each starting state is in itself again an (implicit) discriminator state that discriminates between different callers of that class.

3.3.11 Waiting_caller_proceed-construct

The situation that the caller has to wait (after it has placed the call) for the callee to return some result, can be modeled by two variants of the caller_callee-construct. The 1^{e} variant of the caller waits-construct is already described in the paragraph 'caller_callee-construct' of this chapter.

In this paragraph the 2^{e} variant of the caller waits-construct will be described. In this 2^{e} variant no extra trap is needed in the subprocesses of the employees. Still some additional information is needed in this 2^{e} variant. The manager has to know if a caller has already placed a call to the callee. In general the manager has to know which caller has called which callee. This information is called the 'caller_callee-relation' (see also the 'caller_callee-relation' paragraph).

The waiting_caller-proceed-construct (= 2^{e} variant of caller waits) is described using an example in which the caller 'operation_b' places a call to the callee 'operation_a' and has to wait for a result. It is assumed that the callee is the only operation of its class.

The external STD of the class of the callee will have a neutral state, a starting state in which the callee is started and a 'waiting caller proceed' state in which the caller is allowed to proceed.

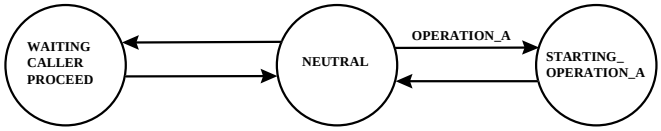


figure 3.48 example : external STD of the callee

The corresponding manager STD of the callee class reflects these states.

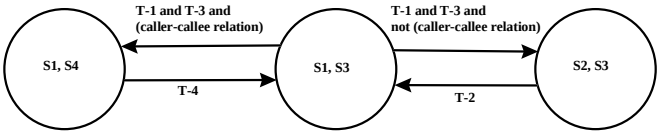


figure 3.49 example : manager STD of the callee

It can be seen from the figure of the manager STD that the 'trap information' guarding the transition from 'neutral' to the starting state is the same as that guarding the transition from 'neutral' to the state 'waiting caller proceed'. The manager needs extra information to decide which transition to make. This extra information is the 'caller_callee'-relation. This relation exists between a caller and a callee when the caller has already placed a call to the callee. When the caller has not placed a call, the relation does not exist.

The subprocesses and traps in the caller and callee are just according to the normal caller_callee-construct. They are S1, S2, S3 and S4 and T-1, T-2, T-3 and T-4. They are shown in the next four figures.

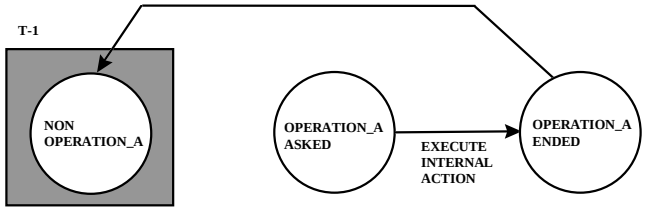


figure 3.50 example : subprocess S1 of the callee (operation_a)

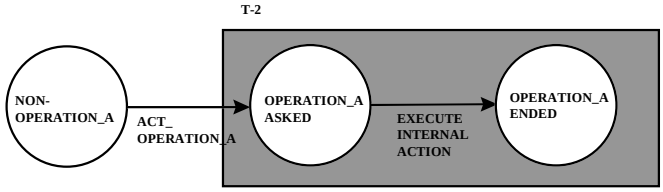


figure 3.51 example : subprocess S2 of the callee (operation_a)

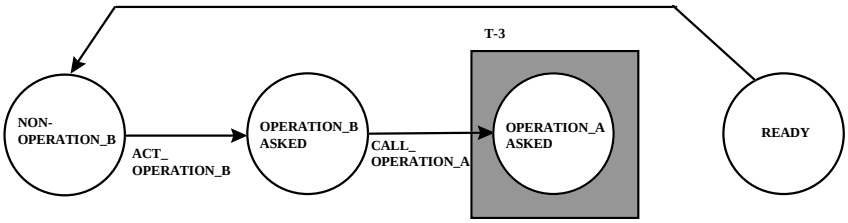


figure 3.52 example : subprocess S3 of the caller (operation_b)

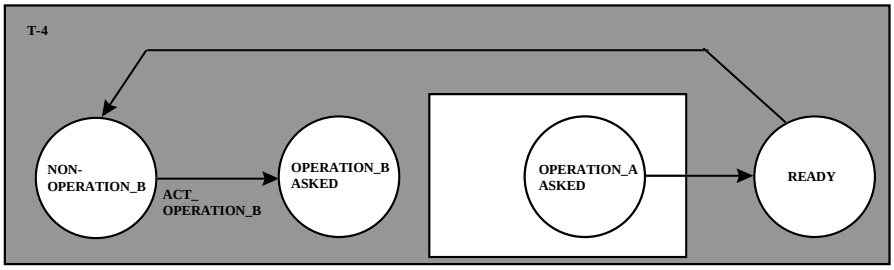


figure 3.53 example : subprocess S4 of the caller (operation_b)

Initially the manager is in its state ‘neutral’. Here it prescribes S1 for the callee and S3 for the caller. When the callee is ready in its trap T-1 and the caller enters in its trap T-3 (places the call), and there is no caller_calllee-relation (the caller has not yet called the callee), then the manager can (and eventually will) make the transition to its state ‘starting operation_a’. Here it prescribes S2 for the callee and still S3 for the caller (i.e. the caller has to wait in its trap T-3). When the callee enters its trap T-2 (it starts executing) the manager can and will transit back to ‘neutral’. Here it prescribes S1 for he callee and S3 for the caller (i.e. the caller still has to wait in its trap T-3).

When the callee finishes executing (after having past some result to the waiting caller), it enters its trap T-1. Now the callee is in T-1 and the caller is in T-3 and there exists a caller_callee-relation. The manager can and will make the transition to its state ‘waiting caller proceed’. Here it prescribes S1 for the callee and S4 for the caller (i.e. he caller is allowed to proceed in its next subprocess). When the caller enters now its trap T-4, the manager can and will make the transition back to ‘neutral’.

discriminator_waiting_caller_proceed-construct

When there are more then one operation in a class for which the callers have to wait for a result, they all have a corresponding ‘waiting caller proceed’-state in the manager STD of this class. These ‘waiting caller proceed’-states are aggregated into one discriminator state ‘disc_waiting_caller_proceed’. In this discriminator state the manager determines

which caller had called some terminated callee. This caller is then allowed to proceed. (Due to time limitations no generalization of this construct can be given in this thesis. An example of the use of the ‘disc_waiting_caller_proceed’-construct can be found in the SOCCA model of the KPA ‘Software Project Planning’, phase 2 of the process fragment ‘writing project management documents’, class ‘head production section’.)

3.3.12 Caller_Callee-relation

The ‘waiting_caller_proceed-construct’ uses the caller_callee-relation (as explained in the paragraph ‘waiting_caller _proceed-construct’).

This relation exists between a caller and a callee when the caller has placed a call to the callee. When the caller has not placed a call, the relation does not exist. This information comes from some internal bookkeeping of the manager. If an operation is started by the manager on behalf of a caller, a caller-callee relation is initiated. If both the callee and the caller have terminated, the particular caller-callee relation is cancelled in the internal administration of the manager.

The caller_callee-relation includes information down to the STD-instance level. It keeps track which instance of a callee internal STD is executing on behalf of which instance of the caller internal STD.

3.3.13 Counting-construct

A common situation in a real life process is ‘counting’. To model this in SOCCA, the counting-construct can be used.

The counting-construct is incorporated in the external STD of a class. It works in conjunction with a ‘nop’ (no-operation) internal operation. The example below shows a ‘two counting’-construct. The operation ‘pr_count_two _phase_ended’ is a ‘nop’.

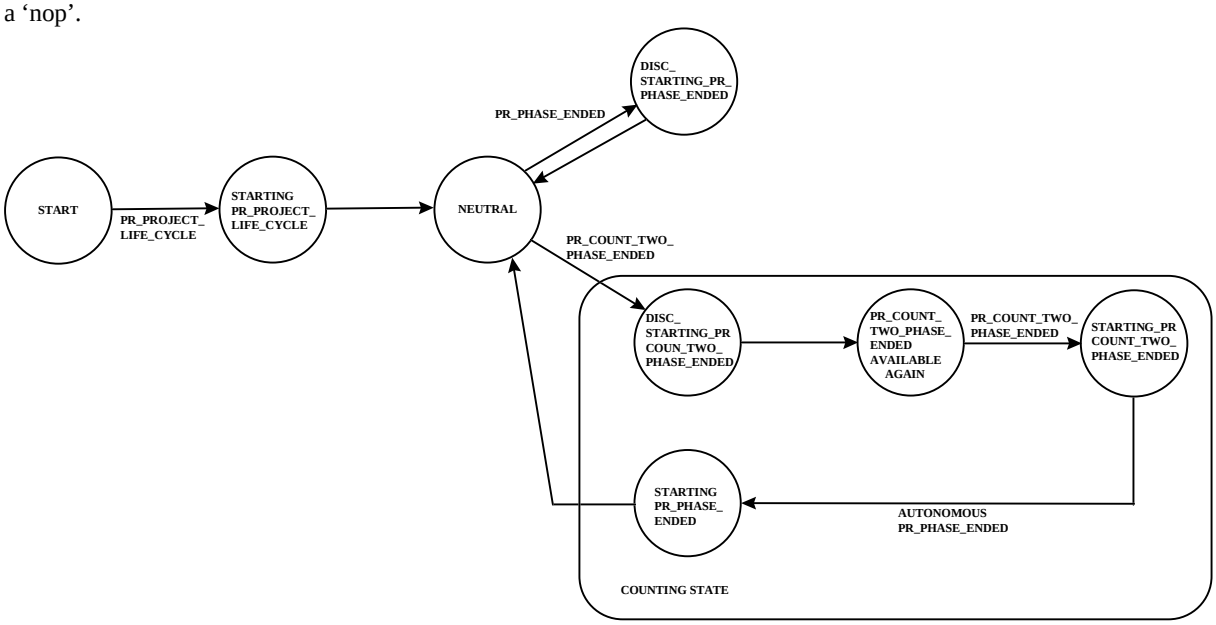


figure 3.54 example : counting-construct in external STD

The counting state counts the number of events (modeled by calls to its (nop) operation ‘pr_count_two_phase_ended’) that have to occur before the external STD has to take some action. The action it takes is modeled by the autonomous call of the external STD to one of its own operations. In the example this is ‘autonomous pr_phase_ended’.

(Due to time limitations no generalization of this construct can be given in this thesis. An example of the use of the ‘counting’-construct can be found in the chapter ‘Integration of process fragment ‘writing project management documents’’, class ‘project (control class)’.)

3.3.14 Finishing state-indicator / finishing state-construct

The finishing state indicator is used in the integration of SOCCA sub-models into one big SOCCA model. It is applied to the ‘finishing’ state of the operation(s) of a sub-model. A ‘finishing’ state is a state that indicates that a submodel has progressed far enough in its execution and that it can be terminated (by the ‘control’ object). The notation convention for a ‘finishing’ state-indicator is an asterix (*).

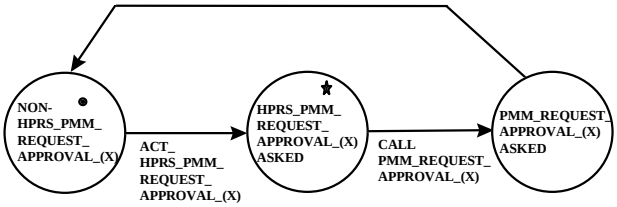


figure 3.55 example : finishing state-indicator in internal STD

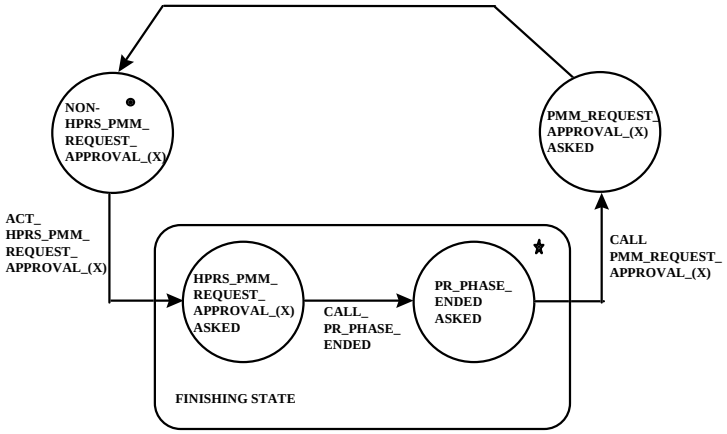


figure 3.56 example : finishing state-construct (internal STD-part)

The ‘finishing’ state is in fact an aggregate state. In this state a signal is given to the ‘control’ object that the sub-model can be terminated. This signal is modeled by a call to an operation of the ‘control’ class. In the example above ‘call pr_ phase_ended’.

(Due to time limitations no generalization of this construct can be given in this thesis. An example of the use of the ‘finishing state’-construct can be found in the chapter ‘Integration of process fragment ‘writing project management documents’’, class ‘project (control class)’.)

3.4 Views

A view of an STD is itself also an STD. In this view-STD some parts of the original STD are shown and some parts are not shown (are hidden). The parts that are shown are those that are of interest for a particular viewer (user of the STD).

So many views on an STD are possible depending on the needs of the user of the STD.

If, for example, in an external STD all the states and transitions are shown that are necesseray to model the communication (between objects), this is called a 'communicative view'. This view is necessary to complete the SOCCA modeling of the communication.

But when using this same external STD to present the SOCCA model to the 'end-users' of the modeled process, another situation arises. These 'end-users' are not interested in the lower level communication details. They only want (need) a clear picture of the modeled process. Also they are only interested in detail in the model of their own workfield, if they are specialists, or in a global model on a higher abstraction level, if they are managers.

For these 'end-users' the communication details are 'filtered' out of the external STD and the abstraction level of the STD is raised. Such an external STD is called an 'organizational view'.

Another example, on a more technical level, is the view that a manager STD has on its callee-employee STDs. All the manager STD is interested in is the 'act-transition' in the callee employee. It 'sees' only two states in the callee, either it is 'non-active' or it is executing. So the 'manager-STD' view of a callee employee STD consists of only two states, 'non-active' and 'active' and the 'act-transition' and 'return-transition' between them.

Two construction methods are possible to construct a view STD from another STD. These are the 'homomorphic picture'-construction and the 'aggregate state'-construction.

3.4.1 Homomorphic picture-construction

A homomorphic picture is a structure preserving mapping from one STD to another STD. The mapping function is called a homomorphism 'h'.

When a homomorphic function 'h' can be found from one STD 'A' to an STD 'B' then STD 'B' is a view of STD 'A'. This concept is given in [EBE]. The rules for constructing the function 'h' are also given in [EBE].

An example of the use of the 'homomorphic picture'-construction can be found in the chapter 'KPA 'Software Configuration Management', class 'configuration item'.

3.4.2 Aggregate state-construction

This construction method has two steps :

- The first step in the aggregate state-construction consists of taking together certain states to form 'aggregate' states.

- The second step in the construction is to no longer show the states inside an aggregate state.

The inverse construction is also easily performed. To show more detail in an STD, some states are 'exploded'. That is, the sub-states within that state are shown again.

An example of the use of the 'aggregate state'-construction can be found in the chapter 'KPA 'Software Configuration Management', class 'configuration item'.

3.5 Integration of sub-models

The intention of the integration process is not to influence the sub-models that are being integrated. That is to say to make the modeling of the constituent smaller process fragments independent of their integration. This allows for a separate modeling of the sub-models (by separate engineers) in a big project. It also facilitates the update of the total model if there are any changes required during the life time of the model (maintainability). This is according to the accepted software engineering principle of low coupling between software modules.

In this paragraph the general principles involved in the integration will be discussed. A more detailed description and the application of the principles to a specific case, can be found in the chapter 'Integration of process fragment 'writing project management documents''.

3.5.1 Algorithm

The algorithm involves the 'linking' together of the external STDs of the classes participating in the sub-models into one 'total' external STD for each class for the integrated model. The states of this 'total' external STD are the states of the separate external STDs plus intermediate states in between.

A control operation (of the control object) then 'switches' the total external STD from one (no longer separate) external STD (of sub-model 1) to the next (no longer separate) external STD (of sub-model 2). The behavior of a class is thus first according to the external STD of sub-model 1 and then according to the external STD of submodel 2.

The control operation 'switches' the total external STDs of all participating classes at the same time. It 'switches' from sub-model to sub-model. The sub-models will execute sequentially.

The above 'linking' of the external STDs takes place when the sub-models are sequential in the integrated model. When the sub-models are parallel in the integrated model, the linking does not take place. The external STDs stay separate. The control operation will then take care that the separate external STDs run concurrently (both external STDs will be valid at the same time). The sub-models will now execute in parallel.

The separate steps of the algorithm are :

1. construct the total external STD for each class for :
    - sequential integration or
    - parallel integration or
    - a combination of both

2. define a control class

3. define the control operation

These steps are illustrated using a simple example. The example involves just one class, class_a, that participates in two sub-models, sub-model_1 and sub-model_2. The sub-models will be integrated first sequentially into one bigger model. Thereafter the parallel integration of the two sub-models will be shown.

3.5.2 Sequential integration

3.5.2.1 Total external STD

Class_a has an external STD for sub_model_1 and an external STD for sub-model_2. These external STDs are shown in a view that abstracts from all irrelevant detail. They are shown as just one state.

figure 3.57 example : external STDs of class_a for sub-model_1 and for sub-model_2

As an intermediate step in the construction of the total external STD of class_a, the two external STDs are first ‘extented’ with a start state and a final state. The start state has a transition leaving it and entering the sub-model external STD. The final state has a transition coming into it from the sub-model external STD. The transition leaving the start state gets the label ‘change_to_sub-model_x’. The transition entering the final state gets the label ‘sub-model_ended’.

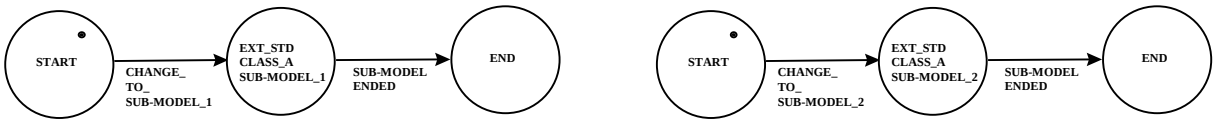


figure 3.58 example : extended external STDs of class_a for sub-model_1 and for sub-model_2

Since in this example the sub-models are sequential in the integrated model, the next step is to connect the two extended external STDs with each other in such a way that the final state of one sub-model external STD coincides with the start state of the next sub-model external STD. This ‘coinciding’ state between two phase-external STDs is called an ‘intermediate’ state.

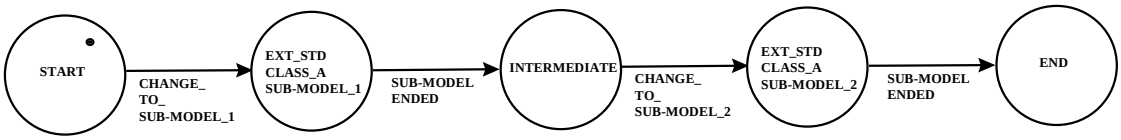


figure 3.59 example : total external STD of class_a for integrated model (sequential)

The operations ‘change_to_sub-model_1’, ‘change_to_sub-model_2’ and ‘sub-model_ended’ are ‘nops’ (no-operations). They are added to the operations of class_a.

This concludes the construction of the total external STDs of class_a for the integrated model.

Next a control class must be defined. This depends on the process that is being modeled. Sometimes a ‘natural’ candidate can be found among the existing classes. At other times a control class will have to be specially defined.
For the sake of this example the existence of a control class is assumed.

Next the control operation has to be defined.

3.5.2.2 Control class

The control object models the (higher level) control flow of the integrated process (fragment). It does this by manipulating the total external STD of the class_a that participates in the constituent (smaller) process fragments. For this purpose the control object has an internal operation (control operation) which calls the ‘sub-model-changing’-operations of a total external STD in sequence.

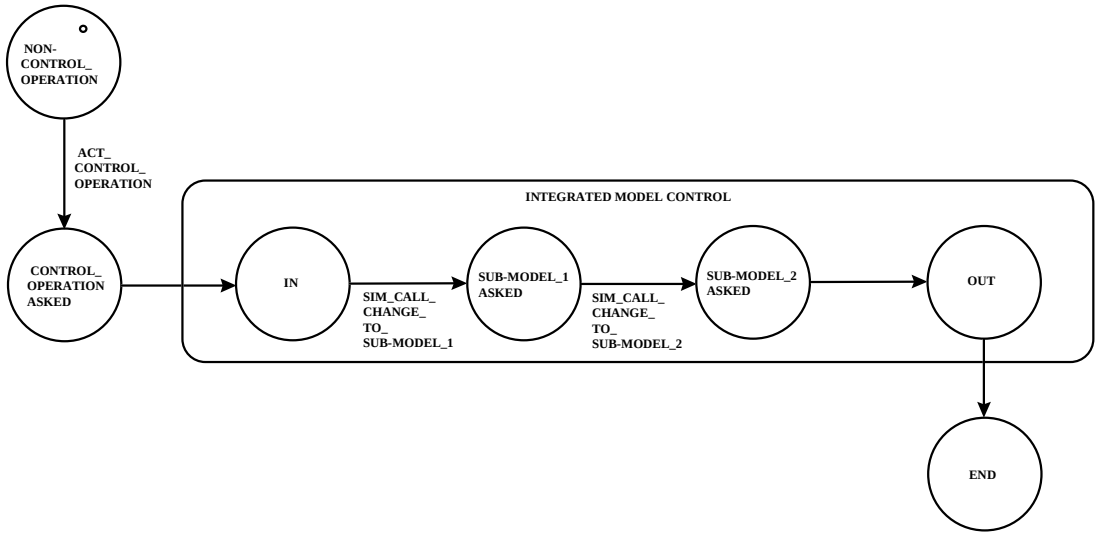


figure 3.60 example : control_operation, internal operation of control class

As can be seen from the figure, the control operation calls the ‘change_to_sub-model_1’ and ‘change_to_sub-model_2’ in sequence. It uses for this purpose a simultaneous call. In this example there is only one participating class, class_a. Only the operations of this class are called by the sim_call. When there are more participating classes, the sim_call is placed to all participating classes. So all classes will change to the next sub-model simultaneously.

The total external STD of the ‘class_a’ starts in its state ‘start’. Now the control operation calls ‘change_to_sub-model_1’. The total external STD of ‘class_a’ can and will transit to its state ‘ext_STD class_a sub-model_1’. I.e. the ‘class_a’ will perform its ‘sub-model_1’-behavior. Sub-model_1 is executing.

The control operations will continue with its execution and calls ‘change_to_sub-model_2’. This call is not (yet) serviced by the total external STD of ‘class_a’ because it is not (yet) in a state to do so.

The responsibility of knowing when sub-model_1 is ready with all its actions, rests with the sub-model itself. Only it can know when it is ready. In the ‘finishing state’ of its last operation, the ‘class_a’ calls the operation ‘control_sub-model_ended’ of the control object. (The ‘finishing state’ is explained in the paragraph ‘finishing state’-indicator / ‘finishing state’-construct of this chapter).

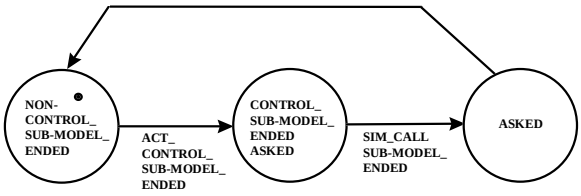


figure 3.61 example : control_sub-model_ended, internal operation of control class

This ‘control_sub-model_ended’ operation then ‘sim_calls’ the operation ‘sub-model_ended’ of the ‘class_a’ (and if there are more participating classes, of all those classes). The total external manager of the ‘class_a’ can and will transit to its state ‘intermediate’. I.e. its sub-model_1 behavior will end.

Now is the total external STD of ‘class_a’ in a state where it can service the call to its operation ‘change_to_sub-model_2’ which the control operations has already placed.

The total external STD of ‘class_a’ can and will transit to its state ‘ext_STD class_a sub-model_2’. I.e. the ‘class_a’ will perform its ‘sub-model_2’-behavior. Sub-model_2 is executing.

Sub-model_2 is ended by a call to the operation ‘control_sub-model_ended’ of the control object. This call is placed by the sub-model itself in the ‘finishing state’ of its last operation.

In this way are the two sub-models integrated sequentially into one bigger model. When they are to be integrated parallel into one bigger model, the algorithm is as follows.

3.5.3 Parallel integration

The extended external STDs of ‘class_a’ are constructed the same way as for the sequential integration. Also the control_operation is the same as for the sequential integration. The difference lies in the total external STD and in the control_sub-model_ended operation.

When constructing the total external STD of ‘class_a’ the extended external STDs are not linked one after the other, but they stay separately. The total external STD consists now of two ‘not-connected’ external STDs. Both are valid at the same time.

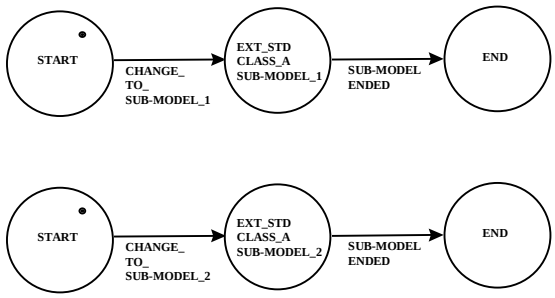


figure 3.62 example : total external STD of class_a for integrated model (parallel)

The total external STD of the ‘class_a’ starts now in both ‘start’ states. Now the control operation calls ‘change_to_sub-model_1’. The one part of total external STD of ‘class_a’ can and will transit to its state ‘ext_STD class_a sub-model_1’. I.e. the ‘class_a’ will perform its ‘sub-model_1’-behavior. Sub-model_1 is executing.

The other part of total external STD of ‘class_a’ still stays in its ‘start’ state.

The control operation will continue with its execution and calls ‘change_to_sub-model_2’. The other part of total external STD of ‘class_a’ can and will transit to its state ‘ext_STD class_a sub-model_2’. I.e. the ‘class_a’ will perform its ‘sub-model_2’-behavior. Sub-model_2 is executing.

Now both sub-models execute in parallel.

The terminating of either sub-model causes some changes in the ‘control_sub-model_ended’ operation of the control class. The changes are also depending on the process that is being modeled. It can be that both sub-models have to finish before they can be permitted to go to their ‘end’ state. This is conceivable when the integrated model itself is part of a still larger model. In this case the ‘counting’-construct can be used by the control class to ascertain that both sub-models have called the ‘control_sub-model_ended’ operation in their ‘finishing state’.

It is also possible that when one sub-model finishes it can be terminated and that the other sub-model is allowed to continue. In this case the ‘control_sub-model_ended’ operation has to be parameterized. The finishing sub-model calls the operation with the appropriate parameter. The operation ‘control_sub-model_ended_(x)’ then knows which sub-model to terminate and it calls the ‘sub-model_ended’ operation of total external STD of that sub-model.

Then again it may be the case that when one sub-model finishes, the other sub-model is no longer of interest and may also be terminated. In this case it seems that the operation ‘control_sub-model_ended’ does not have to be changed. When it is called it sim_calls ‘sub-model_ended’ of both sub-models. The total external STD of the sub-model that called ‘control_sub-model_ended’, can and will transit to its ‘end’ state. But the total external STD of the other sub-model may or may not transit to its ‘end’ state. If can not be forced to do so. It can just go on executing. Even up to the point where it calls ‘control_sub-model_ended’ when it finishes. This is a problem. And when it does transit to its ‘end’ state before it is finished, there is the possibility that it will leave some objects in the sub-model in an incorrect state.

So in this case the control class must use a counting-construct and the second sub-model must be allowed to finish in a normal way.

3.5.4 Scaleability

In the example above only one class participated in the two sub-models. This can be easily extended to more participating classes. This is taken care of by the 'simultaneous' calls in the 'control operation' and in the 'control_sub-model_ended operation'. The example showed only two sub-models. This can easily be extended to more sub-models. This is done by just putting more (sequential) states in the 'control operation' and incorporating the external STDs of the extra sub-models into the appropriate total external STDs (either sequentially or parallel).

So, three kinds of integration of SOCCA sub-models are possible :

- sequential integration of sub-models
- parallel integration of sub-models
- mixed sequential and parallel integration of sub-models

This constitutes the first step in the 'scaling up' of a SOCCA model. The next step involves the integration of sub-models that are themselves integrated models. I.e. the sub-models to be integrated are the result of the first step. They already incorporate a 'control' class. Now the integration algorithm is applied to the 'control' classes of the sub-models. The 'control' classes are then managed by a 'master' control class.

So, also three kinds of integration of SOCCA integrated sub-models are possible :

- sequential integration of integrated sub-models
- parallel integration of integrated sub-models
- mixed sequential and parallel integration of integrated sub-models

In this way successive bigger models can be build. To prevent the bigger models to become to unwieldy (physically to big), the external STDs must be aggregated to a higher level view before they are incorporated into their total external STD. In this way, using the integration algorithm in combination with the view concept, the model can be scaled up in a transparant manner.

SOFTWARE PROCESS MODELING in SOCCA

student : H.G.Brugman
studentnr : 9105506
version : 0.10

4. Key Process Area 'Software Configuration Management'

4.1 Introduction

In this chapter the 'Software Configuration Management'-process of the software development organization 'Waco Business Unit' (WBU) of the Dutch Ministry of Defense is modeled using the SOCCA process modeling language. The SOCCA model is described in paragraph 4.3. Paragraph 4.3.1 describes the 'class diagrams' of the model and paragraph 4.3.2 describes the 'state transition diagrams' of the model.

Also the usefullness of a SOCCA model as a process description is investigated in this chapter. This is done by checking if the SOCCA model of the 'Software Configuration Management'-process can be used as input for a process audit. As audit method is chosen the 'Capability Maturity Model'-assessment. The process audit is described in paragraph 4.2.

4.2 CMM Assessment

In this paragraph the usefullness of a SOCCA model as a process description is investigated This is done by checking the implementation of the CMM 'Key Practices' by the WBU organization while using the SOCCA model of the 'Software Configuration Management'-process as a reference instead of the real process.

The purpose of Software Configuration Mangement (SCM) is to establish and maintain the integrity of the products of the software project throughout the project's software lifecycle [CMM].

Software Configuration Management (SCM) involves identifying the configuration of the software, controlling the changes to the configuration and maintaining the traceability of the configuration. The work products placed under SCM include the software products to be delivered to the customer (e.g. requirements documents, software design documents and the executable program(s)) and the items that are required to create these software products (e.g. source code, compiler/linker and compile/link command-files).

A software baseline library is established containing the software baselines. Changes to baselines and the release of software products built from the software baseline library are controlled via change control and configuration auditing functions.

The Capability Maturity Model (CMM) states the following goals for this Key Process Area (KPA) :

Goal 1 : Software Configuration Management activities have to be planned
Goal 2 : Selected software work products have to be identified, controlled (e.g. via version numbering) and made available
Goal 3 : Changes to identified software products have to be controlled (e.g. via numbered problem and change reports)
Goal 4 : Affected groups and individuals have to be informed of the status and content of software baselines

Key Process Areas in CMM are divided into Key Practices. A Key Practice describes at the lowest level 'what' has to be done, but not 'how' it should be done. For a correct fulfillment of the goals of the KPA all its Key Practices have to be satisfied by the software development organization.

Key Practices are organized into five groups (processes). A group (process) is called 'common features' in CMM. The five groups are : 'Commitment to perform', 'Ability to perform', 'Activities performed', 'Measurement and analysis' and 'Verifying implementation'.

1. 'Commitment to perform' describes the actions an organization must take to ensure that the SCM process is established and will endure. Typically this involves the codifying of organizational policies (in manuals) and senior management commitment to this policies. This codifying process is not modeled in this thesis. The result of this process is the 'Manual for Technical Project Management' now in use in the Waco Business Unit (WBU) organization.

2. 'Ability to perform' describes the preconditions that must exists in an organization to implement the SCM process correctly. Typically this involves resources, organizational structures and training. The organizational structure as applied to the SCM process is modeled by the class diagrams of the next chapter.

3. 'Activities performed' describes the roles and procedures necessary to implement the Key Process Area. Typically this involves establishing plans and procedures, performing the work, tracking it and taking corrective actions as necessary. The SCM process is modeled by the state transition diagrams of the next chapter.

4. 'Measurement and analysis' describes the need to measure the SCM process and analyzes the measurements. Typically this involves measurements that could be taken to determine the status and effectiveness of the 'Activities performed'. This process is not implemented in the Waco Business Unit (WBU).

5. 'Verifying implementation' describes the steps to ensure that the activities are performed in compliance with the process that has been established. Typically this involves reviews and audits by management and software quality assurance. This process takes only place at an 'ad-hoc'-basis in the Waco Business Unit (WBU). It is not modeled in this thesis.

The SOCCA model of the Software Configuration Management-process as given in the next paragraph, models the organizational structure with class diagrams. It models the implemented SCM-process with state transition diagrams.

The following table lists all the Key Practices of this KPA in the left column. In the right column is indicated if and how the Waco Business Unit (WBU) has satisfied a particular Key Practice.

Commitment to perform

no	**Key Practice**	**WBU implementation**
1	The project follows a written organizational policy for implementing software configuration management	The SCM process is described in the 'Manual for Technical Project Management'

Ability to perform

no	**Key Practice**	**WBU implementation**
1	A board having the authority for managing the project's software baselines exists	See the class diagram of the SOCCA model of this KPA. The board is modeled by the class 'configuration control board'.
2	A group that is responsible for coordinating and implementing SCM for the project exists	See the class diagram of the SOCCA model of this KPA. The group is modeled by the class 'software configuration board'.
3	Adequate resources and funding are provided for performing the SCM activities	An automatic configuration management tool (CMS) is available
4	Members of the SCM group are trained in the objectives, procedures, and methods for performing their SCM activities	not implemented. There exists no SCM group within the WBU.
5	Members of the software engineering group are trained to perform their SCM activities	Members of the software engineering group have to follow the in-house SCM training course.

Activities performed

no	**Key Practice**	**WBU implementation**
1	A SCM plan is prepared for each software project according to a documented procedure.	See the SOCCA model of the KPA 'Software Project Planning', Key Practice 'Activities performed' no 8.
2	A documented and approved SCM plan is used as the basis for performing the SCM activities.	The Software Configuration Management plan is part of the software development plan. See the sub-attribute 'SCM-plan' of the class 'software development plan' of the class diagram of the SOCCA model of the KPA 'Software Project Planning'.

no	Key Practice	WBU implementation
		As such the SCM is documented in the Software Development Plan (SDP) and approved as part of the SDP approval.
3	A configuration management library system is established as a repository for the software baselines	a configuration management-tool (CMS) is used by the projects.
4	The software work products to be placed under configuration management are identified	See the class diagram of the SOCCA model of this KPA. The identification is modeled by the attributes 'id' and 'version' of the class 'configuration_item'
5	Change requests and problem reports for all configuration items are initiated, recorded, reviewed, approved, and tracked according to a documented procedure.	The procedure is documented by the behavior part (STDs) of the SOCCA model of this KPA. More specifically by the operations of the class 'problem_and_change_report'.
6	Changes to baselines are controlled according to a documented procedure.	The procedure is documented by the behavior part (STDs) of the SOCCA model of this KPA. More specifically by the external STD of the class 'configuration item'.
7	products from the software baseline library are created and their release is controlled according to a documented procedure	The procedure is documented by the behavior part (STDs) of the SOCCA model of this KPA. More specifically by the operation (internal) STD of the class 'release_note'.
8	The status of configuration items is recorded according to a documented procedure	The procedure is documented by the behavior part (STDs) of the SOCCA model of this KPA. More specifically by the external STD of the class 'configuration item'.
9	Standard reports documenting the SCM activities and the contents of the software baseline are developed and made available to affected groups and individuals	not implemented
10	Software baseline audits are conducted according to a documented procedure	The procedure is documented in the behavior part (STDs) of the SOCCA model of the KPA 'Software Quality Assurance' (not part of this thesis)

Measurement and analysis

no	Key Practice	WBU implementation
1	Measurements are made and used to determine the status of the SCM activities	not implemented

Verifying implementation

no	Key Practice	WBU implementation
1	The SCM activities are reviewed with senior management on a periodic basis	not implemented
2	The SCM activities are reviewed with the project manager on a periodic and event-driven basis	only on a 'ad-hoc'-basis (event-driven)
3	The SCM group periodically audits software baselines to verify that they conform to the documentation that defines them	not implemented
4	The software quality assurance group reviews and/or audits the activities and work products for SCM and reports the results	only on a 'ad-hoc'-basis (event-driven)

4.3 SOCCA model

The SOCCA model depicts the Configuration Management-process that is in use within the Waco Business Unit (WBU). This process is described in the 'Manual for Technical Project Management', chapter 3 'Configuration management', sections 1 and 2, 'Promotion Configuration Item', and 'Procedure handling Problem and Change Report' [MTP].

Paragraph 4.3.1 describes the 'class diagrams' of the model and paragraph 4.3.2 describes the 'state transition diagrams' of the model.

4.3.1 Class Diagrams

The data perspective of CM-process is modeled by four class diagrams. The first one shows the classes, subclasses and aggregation associations. The second one shows the general associations, the third one shows the classes with their operations and attributes and the last one shows the 'uses' associations (the import-export diagram).

The figure below shows the classes and subclasses (via generalization/ specialization associations) and the aggregation associations between them. The notation convention of 'superclass/subclass', 'aggregation association' and 'the multiplicity of an association (cardinality ratio constraint)' in SOCCA is the same as in UML [UML], [FOW].

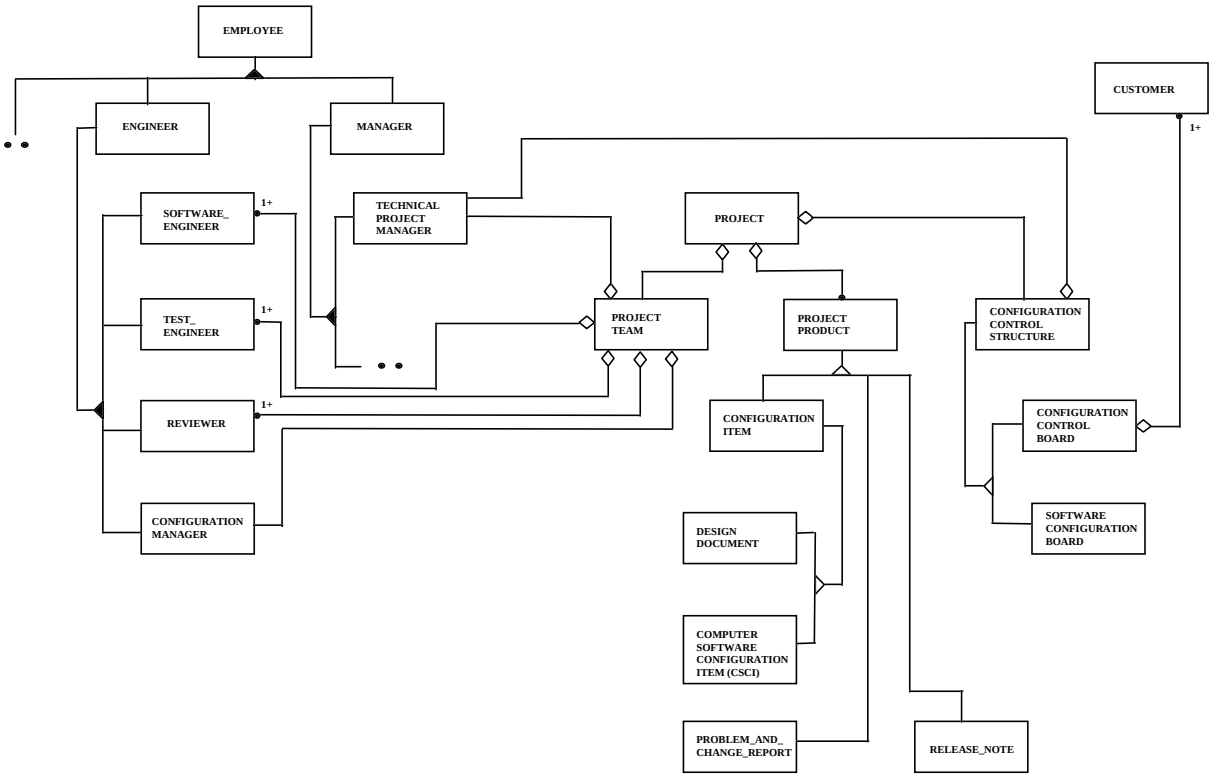


figure 4.1 Class diagram : classes, subclasses and aggregation associations

This class diagram shows the employees involved : SOFTWARE_ENGINEER, TEST_ENGINEER, REVIEWER and CONFIGURATION_MANAGER are overlapping subclasses of the superclass ENGINEER. TECHNICAL_PROJECT_ MANAGER is, among others, a subclass of MANAGER.

A PROJECT consists of one PROJECT_TEAM, zero or more PROJECT_PRODUCTs and one CONFIGURATION_ CONTROL_STRUCTURE.

The PROJECT_TEAM is an aggregation of the employees involved and consists of one TECHNICAL_PROJECT_ MANAGER, one or more SOFTWARE_ENGINEERs, one or more TEST_ENGINEERs, one or more REVIEWERs and one CONFIGURATION_MANAGER.

The superclass PROJECT_PRODUCTs is specialized by the disjoint subclasses CONFIGURATION_ITEM (with its own disjoint subclasses DESIGN_DOCUMENT and COMPUTER_SOFTWARE_CONFIGURATON_ITEM), PROBLEM_AND_CHANGE_REPORT and RELEASE_NOTE.

The CONFIGURATION_CONTROL_STRUCTURE is specialized by the disjoint subclasses CONFIGURATION_ CONTROL_BOARD and SOFTWARE_CONFIGURATION_BOARD which have both the TECHNICAL_PROJECT_ MANAGER as a member. Also at least one CUSTOMER is a member of the CONFIGURATION_CONTROL_ BOARD.

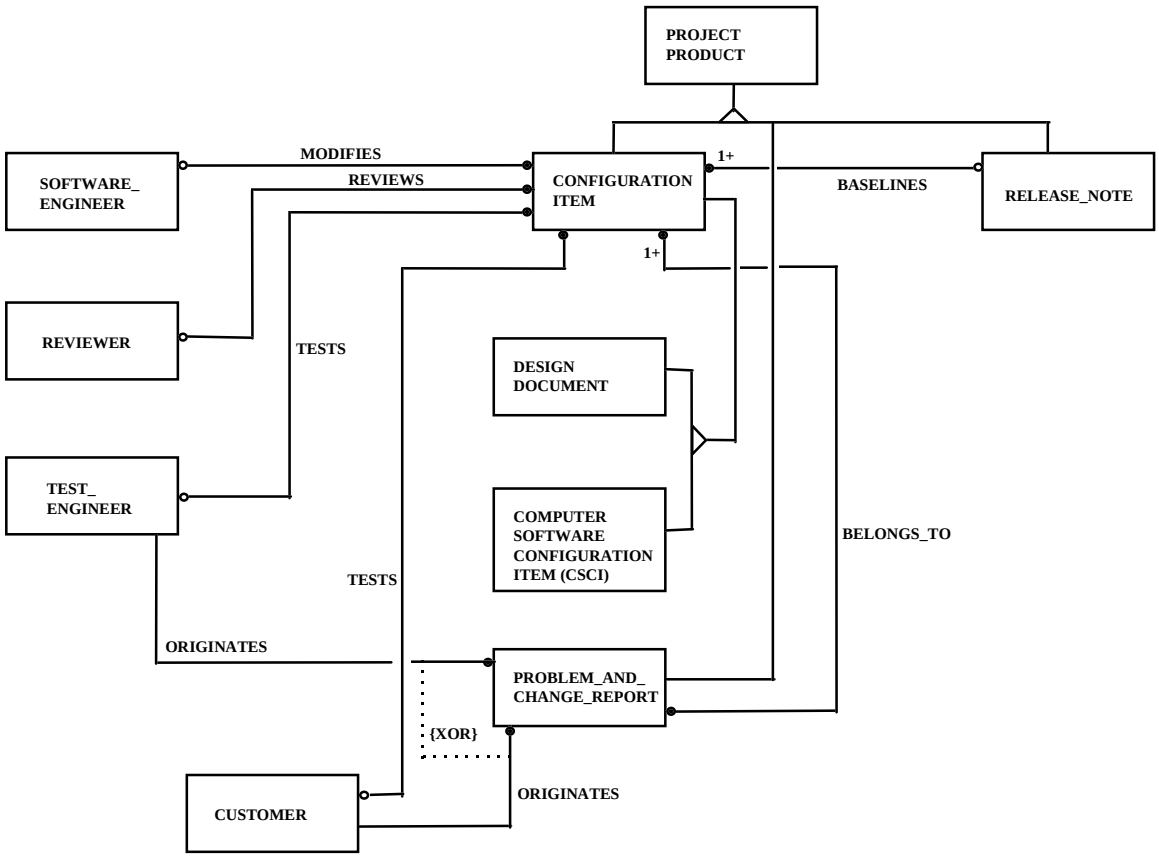


figure 4.2 Class diagram : classes, subclasses and general associations

The figure above shows the classes and subclasses with the general associations between them. The notation convention of '(general) association' is the same in SOCCA as it is in UML. In the notation convention of 'constraint between associations' there is some difference between SOCCA and UML. In SOCCA the constraints are given in a list accompanying the class diagram. In UML this is also allowed but alternatively a constraint can be shown in the diagram itself. A dotted line is drawn between the associations involved. The constraint is then placed in braces near the dotted line. In the figure this is done with the XOR-constraint. The deviation from SOCCA in using the dotted line-notation is justified by the fact that there is only one simple constraint in the diagram. In this case the constraint can easily be incorporated. When there are more or more complex constraints a seperate list of constraints is the better notation convention.

The names and the multiplicity of the general associations are also given in the above figure. An association is inherently bidirectional. I.e it can be traversed in both directions. The name of a binary association reads in a particular direction. This direction is called the forward direction. The opposite direction is called the inverse direction [RUM]. When an association is traversed in the inverse direction, the association name is 'inverted'. E.g. MODIFIES becomes IS_MODIFIED or BELONGS_TO becomes HAS_BELONGING_TO. The class diagram only shows one name per association.

A software_engineer MODIFIES zero or more configuration_items. And a configuration_item IS_MODIFIED by zero or one software_engineer.

A reviewer REVIEWS zero or more configuration_items. And a configuration_item IS_REVIEWED by zero or one reviewer.

A test_engineer TESTS zero or more configuration_items. And a configuration_item IS_TESTED by zero or one test_engineer. A test_engineer ORIGINATES zero or more problem_and_change_reports. And a problem_and_change_report IS_ORIGINATED by either one test_engineer or by one customer, but not by both. This is modeled by the XOR-constraint in the class diagram.

A customer ORIGINATES zero or more problem_and_change_reports. A customer TESTS zero or more configuration_items. And a configuration_item IS_TESTED by zero or one customer.

A problem_and_change_report BELONGS_TO one or more configuration_item. (The problem/change can involve more than one configuration_item.) And a configuration_item HAS_BELONGING_TO it zero or more problem_and_change_reports.

A release_note BASELINES one or more configuration_items. And a configuration_item IS_BASELINED by zero or one release_note.

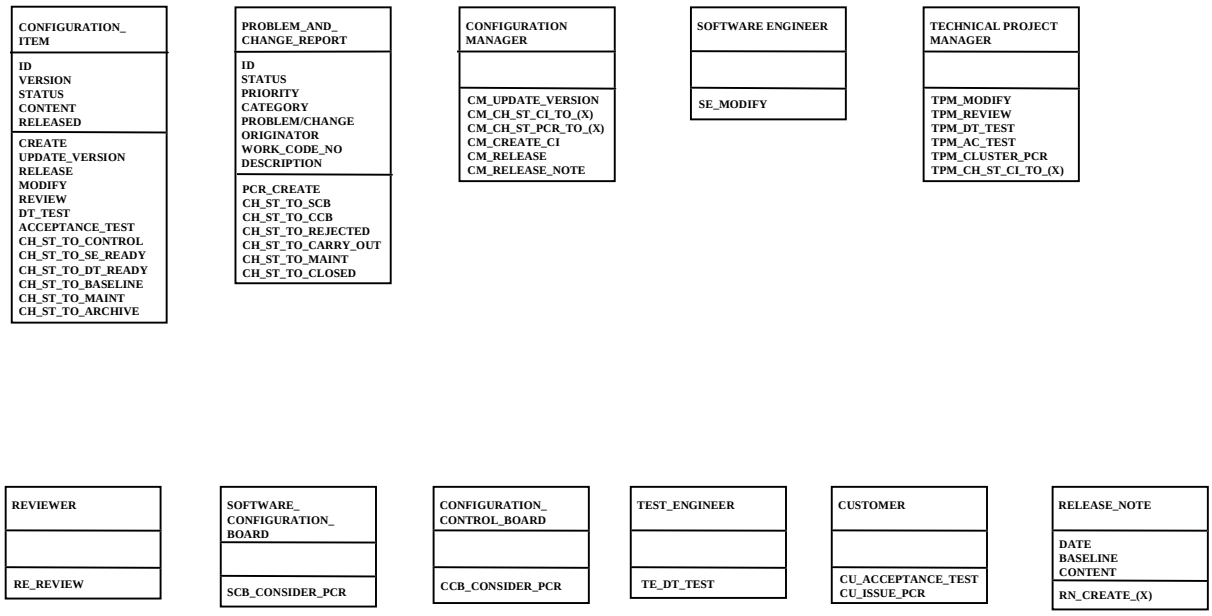


figure 4.3 Class diagram : classes, operations and attributes

The figure above shows the classes with their operations and attributes. The notation convention is the same in SOCCA as in UML. A class is depicted as a rectangle with three compartments. The top compartment holds the class name. The middle compartment shows the attributes. The bottom compartment shows the operations.

The operations will be explained in the ‘internal behavior STDs’-subparagraphs of the ‘State Transition Diagrams’-paragraph of this chapter. The meaning of most attributes can be readily deduced from their name. The following attributes warrant an explanation.

The attribute STATUS in configuration_item has the following domain : (control, se_ready, dt_ready, baseline, maint, archive).

The attribute RELEASED in configuration_item has the following domain : (yes, no). It indicates whether a configuration_item is under configuration management or is released for handling.

The attribute BASELINE in release_note indicates the baseline-type the release_note represents. This attribute has the following domain : (Functional, Allocated, Technical, Product, Operational). These baseline_types are defined in the 'Manual for Technical Project Management' [MTP].

The attribute CATEGORY in problem_and_change_report has the following domain : (A,B,C). Category A means that the problem_and_change_report belongs to a configuration_item with the status 'se_ready' or 'dt_ready'. Category B means that it belongs to a configuration_item which is part of a functional, allocated or operational baseline. Category C means that the configuration_item is part of a technical or a product baseline.

The figure below is the import-export diagram. This diagram identifies which operations are imported by which classes. Within the importing classes the importing operations are identified. This is done by the SOCCA specific binary association 'uses'. This association has the attribute 'import_list' that has as its domain a list of imported operations together with the operations that import them. The style guideline for this association is a solid line with an arrow at one end. The arrow indicates the exporting class.

UML uses the 'interface'-notation to describe the import-export relation between classes. In it a 'type' is connected to the exporting class. The importing class connects via a dashed arrow to this 'type'. A 'type' is a descriptor for objects with an abstract state, with concrete external operation specifications but no concrete operation implementations. A 'type' can be shown either as a small circle or as a class symbol. If the class symbol is used, then the exported operations are shown in the operations-compartment of the class symbol. This is equivalent to showing these operations in the 'import_list' attribute of the SOCCA uses-association.

The import-export diagram is showing the communication between the classes at the highest level and is constructed as a step towards the constructing of the state transition diagrams in the next paragraph.

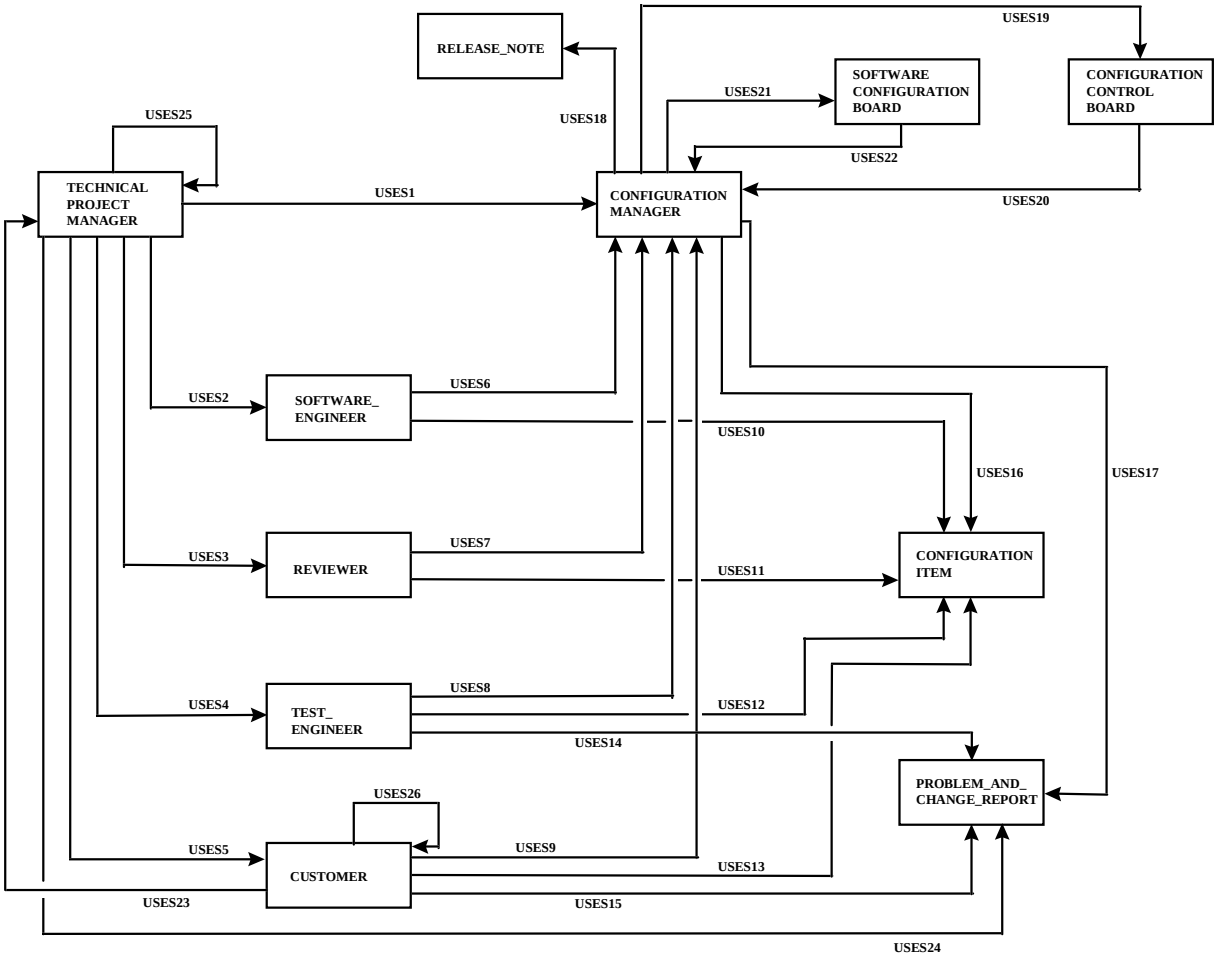


figure 4.4 Class diagram : classes and uses associations

The technical_project_manager is the one who initiates and guides the whole Configuration Management (CM) process by his autonomous behavior. The autonomous operations of the technical_project_manager, which he 'invokes' himself are : tpm_modify, tpm_review, tpm_dt_test, tpm_ac_test, tpm_cluster_pcr and tpm_ch_st_ci_to_(x).

He can direct the software_engineer to start work. The technical_project_manager will direct the configuration_manager to create the configuration_item first if it does not exists yet. Otherwise he will direct the configuration_manager to release the configuration_item.

The technical_project_manager can direct the reviewer to start reviewing on some released configuration_item. He can direct the test_engineer to start testing on some released configuration_item.

The technical_project_manager can decide to bypass the review or dt-test, or both, of some configuration_item. This is done when only minor adjustments are applied to the configuration_item. The technical_project_manager uses his operation tpm_ch_st_ci_to_(x) to direct the configuration_manager to change the status of a configuration_item directly from 'control' to 'se_ready' or 'dt_ready'.

The technical_project_manager informs the customer that the acceptance test can start on a released configuration_item.

Depending on the positive result of the ac_test the technical_project_manager gets informed by the customer that the status of the configuration_item can be changed to 'baseline'. The technical_project_manager directs the configuration_manager to do so. And if all the configuration_items of a particular baseline have reached the status 'baseline', he directs the configuration_manager to write up the release_note of that baseline.

Periodically the technical_project_manager clusters the problem_and_change_reports into one logical group to be solved together. He directs the configuration_manager to change the status of the affected configuration_items to 'maintenance' if they are not part of a baseline. If they are part of a baseline, then their status becomes 'archive' and the technical_project_manager will direct the configuration_manager to create new configuration_items and copy the contents of the old items into the new items.

The software_engineer modifies a configuration_item and when he is finished, he asks the configuration_manager to put it under configuration management (i.e. to change its status to 'control') and to update the version of the configuration_item.

The reviewer reviews a configuration_item and when he is finished he asks the configuration_manager to update the status of the configuration_item according to the result of the review. To 'maintenance' if the review was not ok and to 'se_ready' if the review was ok.

The test_engineer tests a configuration_item and when he detects a defect he writes a problem_and_change_report which he gives to the configuration_manager for further dispatch. If the test is concluded ok, i.e. their are no defects detected, the test_engineer asks the configuration_manager to change the status of the configuration_item to 'dt_ready'. If the test is concluded not ok, the test_engineer takes no 'status changing'-action. The status of the configuration_item will be changed to 'maintenance' later on as a result of the tpm_cluster_pcr operation of the technical_project_manager.

The 'Manual for Technical Project Management' states that a problem_and_change_report becomes closed when the configuration_item it belongs to again gets the status it had when the problem_and_change_report was originated. This procedure can not work. For example if the old status was 'se_ready' and the problem_and_change_report was originated in the dt-test, the configuration_item failed to get the status 'dt_ready'. When the software_engineer repairs the problem_and_change_report and the configuration_item again gets the status 'se_ready' (its old status) the tester still has to perform (part of) the dt-test to check whether this problem_and_change_report is correctly solved.

A solution to this problem is to let the test_engineer perform a check during the dt-test on all the problems_and_ change_reports on this configuration_item to see if they are correctly solved or not. The test_engineer must check all problem_and_change_reports. Also the ones which are originated by the customer in the acceptance-test and during the operational life of the system. If these are not solved, the configuration_item can never be offered to the customer for a (new) acceptance-test.

The customer tests a configuration_item and when he detects a defect he writes a problem_and_change_report which he gives to the configuration_manager for further dispatch. If the test is concluded ok, i.e. their are no defects detected, the customer informs the technical_project_manager that the status of configuration_item can be changed to 'baseline'. If the test is concluded not ok, the customer takes no 'status changing'-action. The status of the configuration_item will be changed to 'maintenance' later on as a result of the tpm_cluster_pcr operation of the technical_project_manager.

The customer also has some autonomous behavior. The operation cu_issue_pcr is an operation which the customer 'invokes' himself. The customer does this when he detects a problem during the operational use of a system or when he wants a functional change in system in operational use.

The configuration_manager creates the new (instances of) configuration_items, including copying the contents from the old item into the new item if directed so by the technical_project_manager. He performs all version- and status-changes of a configuration_item and the release of a configuration_item for review or test. The configuration_manager creates and writes the release_note for a baseline.

He also handles the status-changes of the problem_and_change_reports. If an problem_and_change_report is of the category A or C, he will offer it to the software_configuration_board. If it is of category B he will offer it to the configuration_control_board.

The software_configuration_board decides how to handle the problem_and_change_reports it is offered and directs the configuration_manager to change the status of the problem_and_change_reports according to its decision.

The configuration_control_board decides how to handle the problem_and_change_reports it is offered and directs the configuration_manager to change the status of the problem_and_change_reports according to its decision.

In terms of the values of the 'import-list'-attributes of the uses associations this amounts to the following :

	imported operation	imported by
uses1 :	cm_ch_st_ci_to_(x)	tpm_ch_st_ci_to_(x)
	cm_ch_st_pcr_to(x)	tpm_cluster_pcr
	cm_release	tpm_modify, tpm_review, tpm_dt_test, tpm_ac_test
	cm_release_note	tpm_ch_st_ci_to_(x)
	cm_create_ci	tpm_modify, tpm_ch_st_ci_to_(x)

	imported operation	imported by
uses2 :	se_modify	tpm_modify

	imported operation	imported by
uses3 :	re_review	tpm_review

	imported operation	imported by
uses4 :	te_dt_test	tpm_dt_test

	imported operation	imported by
uses5 :	cu_acceptance_test	tpm_ac_test

	imported operation	imported by
uses6 :	cm_update_version	se_modify
	cm_ch_st_ci_to_(x)	se_modify

	imported operation	imported by
uses7 :	cm_ch_st_ci_to_(x)	re_review

	imported operation	imported by
uses8 :	cm_ch_st_ci_to_(x)	te_dt_test
	cm_ch_st_prc_to_(x)	te_dt_test

	imported operation	imported by
uses9 :	cm_ch_st_prc_to_(x)	cu_acceptance_test, cu_issue_pcr

	imported operation	imported by
uses10 :	modify	se_modify

	imported operation	imported by
uses11 :	review	re_review

uses12 :	imported operation	imported by
	dt_test	te_dt_test

uses13 :	imported operation	imported by
	acceptance_test	cu_acceptance_test

uses14 :	imported operation	imported by
	pcr_create	te_dt_test

uses15 :	imported operation	imported by
	pcr_create	cu_acceptance_test, cu_issue_pcr

uses16 :	imported operation	imported by
	create	cm_create
	update_version	cm_update_version
	release	cm_release
	ch_st_to_control	cm_ch_st_ci_to_(x)
	ch_st_to_se_ready	cm_ch_st_ci_to_(x)
	ch_st_to_dt_ready	cm_ch_st_ci_to_(x)
	ch_st_to_baseline	cm_ch_st_ci_to_(x)
	ch_st_to_maint	cm_ch_st_ci_to_(x)
	ch_st_to_archive	cm_ch_st_ci_to_(x)

uses17 :	imported operation	imported by
	ch_st_to_scb	cm_ch_st_pcr_to_(x)
	ch_st_to_ccb	cm_ch_st_pcr_to_(x)
	ch_st_to_rejected	cm_ch_st_pcr_to_(x)
	ch_st_to_carry_out	cm_ch_st_pcr_to_(x)
	ch_st_to_maint	cm_ch_st_pcr_to_(x)
	ch_st_to_closed	cm_ch_st_pcr_to_(x)

uses18 :	imported operation	imported by
	rn_create_(x)	cm_release_note

uses19 :	imported operation	imported by
	ccb_consider_pcr	cm_ch_st_pcr_to_(x)

uses20 :	imported operation	imported by
	cm_ch_st_pcr_to_(x)	ccb_consider_pcr

uses21 :	imported operation	imported by
	scb_consider_pcr	cm_ch_st_pcr_to_(x)

uses22 :	imported operation	imported by
	cm_ch_st_pcr_to_(x)	scb_consider_pcr

uses23 :	imported operation	imported by
	tpm_ch_st_ci_to_(x)	cu_acceptance_test

uses24 :	imported operation	imported by
	pcr_read_status	tpm_cluster_pcr

uses25 :	imported operation	imported by
	tpm_modify	autonomous operation
	tpm_review	autonomous operation
	tpm_dt_test	autonomous operation
	tpm_ac_test	autonomous operation
	tpm_cluster_pcr	autonomous operation
	tpm_ch_st_ci_to_(x)	autonomous operation, tpm_cluster_pcr

uses26 :	imported operation	imported by

cu_issue_pcr	autonomous operation

4.3.2 State Transition Diagrams

The behavior perspective of CM-process is modeled in SOCCA by STDs (State Transition Diagrams). The behavior perspective comprises both the visible behavior of classes (and thus of objects of that classes) and the behavior of operations (methods) of the classes.

The visible behavior of a class is modeled by an external STD. This STD shows the possible sequences of starting the execution of the class' (object's) operations. Usually there is one external STD per class. The UML equivalent of an external STD is a statechart describing the states an object can be in and the transitions it can make in response to received stimuli..

The behavior of an operation is modeled by an internal STD. It describes the functionality of the operation. This comprises the operation's start and end, the calling of other operations by this operation, and 'local steps' (local functionality). In UML methods are also described by statecharts.

The communication between objects (calling of exported operations) is modeled by manager STDs and employee STDs. External STDs become manager STDs and internal STDs become employee STDs. This mechanism is explained in the SOCCA chapter of this document. This way of modeling communication is unique for SOCCA.

STD's can show different views of the same process. An STD is called a 'communicative view' if it models any communication details between one or more caller and its callee(s). An STD is called an 'organizational view' otherwise.

The behavior of the following classes is modeled for this Key Process Area 'Software Configuration Management':

- technical project manager
- configuration manager
- configuration item
- problem and change report
- software engineer
- reviewer
- software configuration board
- configuration control board
- test engineer
- customer
- release note

4.3.2.1 Technical_Project_Manager

4.3.2.1.1 Technical_Project_Manager : external behavior-STD

The STD of the external behavior consists of a neutral state in which the technical_project_manager waits for a call to one of its operations and of several states in which he starts his operations after they are called. Typically the technical_project_manager does not wait in this states until the called operation is finished, but returns as soon as possible to its neutral state to allow further calls to its operations. I.e. its operations can execute concurrently.

The technical_project_manager exihibits autonomous behavior by starting all his own operations without their being called from another object. When the operation 'tpm_ch_st_ci_to_(x)' is used autonomously, the state 'starting tpm_ch_st_ci_to_(x) (autonomous)' is used to start the operation. The operation 'tpm_ch_st_ci_to_(x)' can also be called by the operation 'cu_acceptance_test' of the customer or by the operation 'tpm_cluster_pcr' of the technical_project_manager itself. This is modeled by the 'disc_starting tpm_ch_st_ci_to_(x)' state that is entered after an external call from either one of these callers. The state 'disc_starting_tpm_ch_st_ci_to_(x)' is a 'discriminator' state. In it the external STD determines from which operation the call is coming and will start the operation on behalf of that caller.

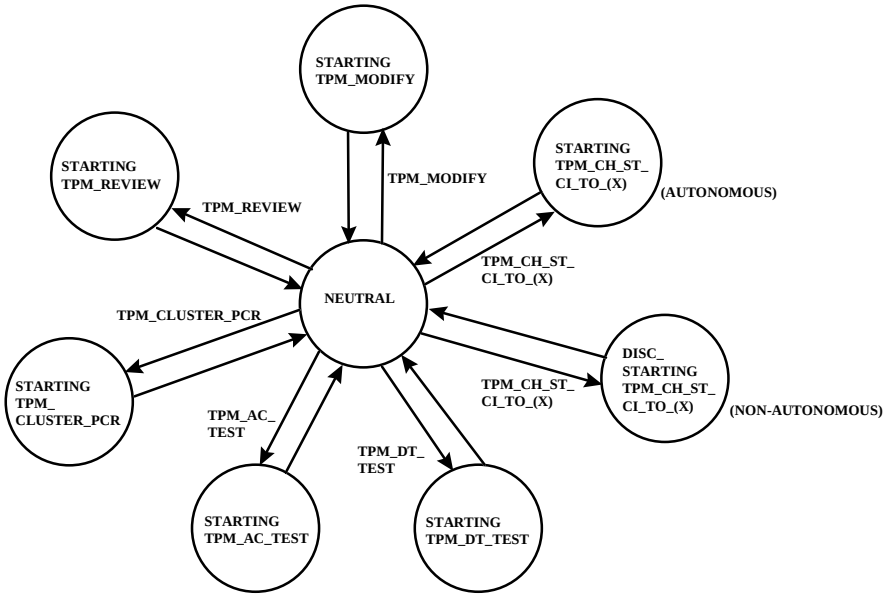


figure 4.5 technical_project_manager : external behavior STD

4.3.2.1.2 Technical_Project_Manager : internal behavior-STDs

The 6 operations of the technical_project_manager have the following internal behavior STDs.

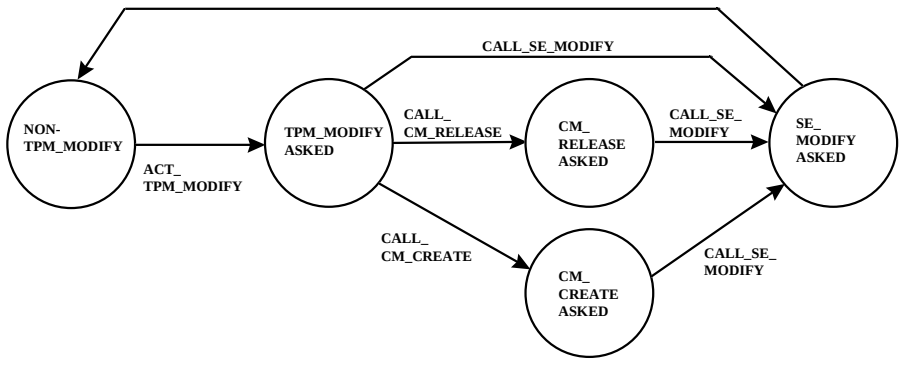


figure 4.6 int-tpm_modify : internal behavior STD

With the operation ‘tpm_modify’ the technical_project_manager (TPM) directs the software_engineer to start modifying a configuration_item (CI). The formal parameter of the operation is the id of the configuration_item that has to be modified. The status of a configuration_item that has to be modified, can be ‘maintenance’. In this case the TPM directs the configuration_manager (CM) to release the CI from configuration management. The configuration_item can also not yet exists. In this case the TPM directs the CM to create the configuration_item. The configuration_item can also exists, but not have a status yet. This is the case after a new CI is created when some (old) CI goes to the status ‘archive’. The software_engineer can start directly to work on such a configuration_item.

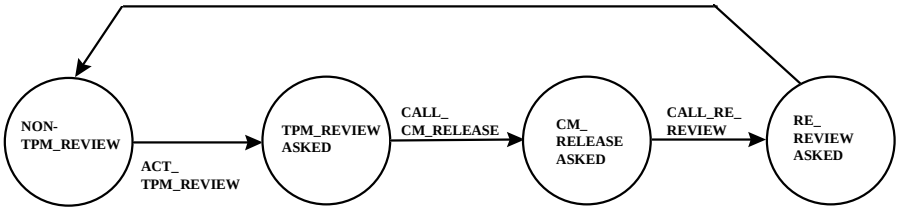


figure 4.7 int-tpm_review : internal behavior STD

With the operation ‘tpm_review’ the technical_project_manager directs the configuration_manager to release the configuration_item. Then the technical_project_manager directs the reviewer to start reviewing the configuration_item. The formal parameter of the operation is the id of the configuration_item that has to be reviewed.

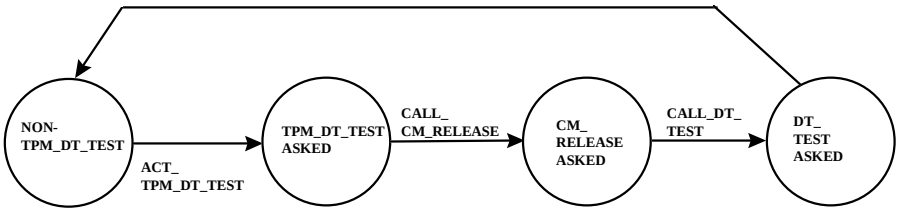


figure 4.8 int-tpm_dt_test : internal behavior STD

With the operation ‘tpm_dt_test’ the technical_project_manager directs the configuration_manager to release the configuration_item. Then the technical_project_manager directs the test_engineer to start testing the configuration_item. The formal parameter of the operation is the id of the configuration_item that has to be tested.

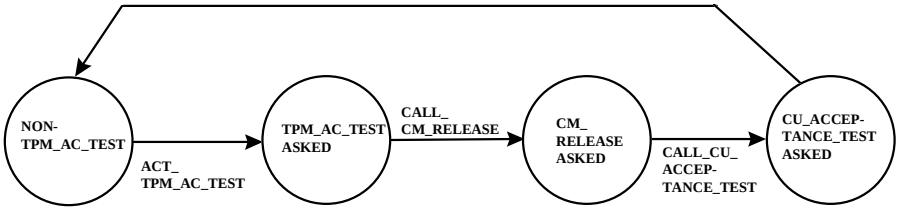


figure 4.9 int-tpm_ac_test : internal behavior STD

With the operation ‘tpm_ac_test’ the technical_project_manager directs the configuration_manager to release the configuration_item. Then the technical_project_manager informs the customer that he can start the acceptance test on the configuration_item. The formal parameter of the operation is the id of the configuration_item that has to undergo the acceptance test.

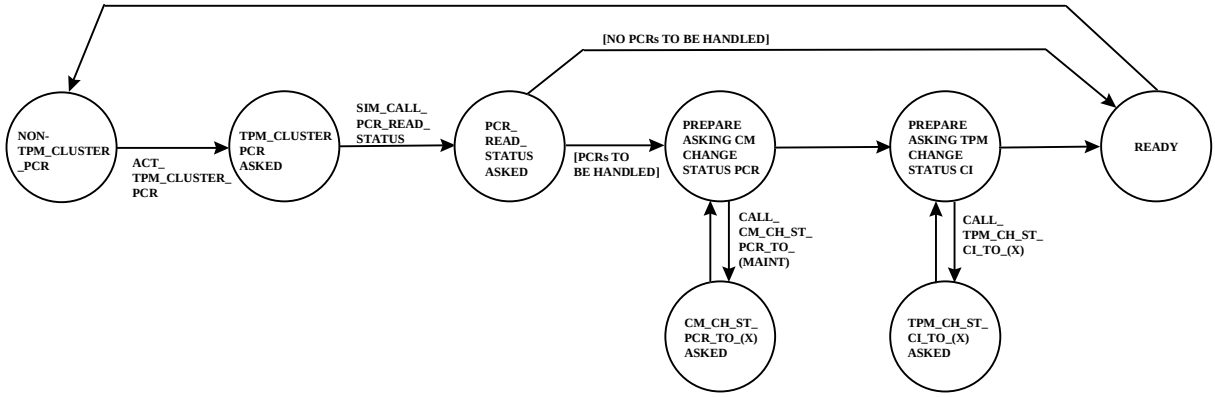


figure 4.10 int-tpm_cluster_pcr : internal behavior STD

With the operation ‘tpm_cluster_pcr’ the technical_project_manager (TPM) clusters the problem_and_change_reports (PCR) that have to be worked on in logical groups. The TPM performs this operation autonomously from time to time. He first reads the status of a number of PCRs. The reading (at the same time) of the status of a number of PCRs is modeled with the ‘simultaneous_call’-construct. All calls to the same operation of objects of the same (or different) class are executed concurrently by this construct.

If there are no PCRs to be handled, the TPM goes to the state ‘ready’. If there are PCRs to be handled (i.e. PCRs with the status ‘carry_out’ or ‘rejected’) the TPM take the following actions. For each PCR with the status ‘carry_out’ the TPM instructs the configuration_manager (CM) the change the status of that PCR to ‘maintenance’ (repetitive calls to cm_ch_st_pcr_to_(maint)). The TPM then transists to its next state. Here he instructs the CM to change the status of each configuration_item that has now PCRs belonging to it, to CI-status ‘maintenance’ or to ‘archive’ if it is a CI that has the status ‘baseline’. The TPM instructs the CM (indirectly) via repetitive calls to the TPM’s own operation tpm_ch_st_ci_to_(x).

For all the PCRs with the status ‘rejected’, the TPM checks the CIs that belong to those PCRs. If there are no other PCRs open on this CI, the TPM instructs the configuration_manager (again via repetitive calls to tpm_ch_st_ci_to_(x)) to their next status. This can be the status ‘dt_ready’ if the PCR was found during ‘dt-test’, ‘baseline’ if the PCR was found during ‘acceptance-test’. If the PCR was found during the operational use of the system, the status of the CI just stays ‘baseline’ and is not changed.

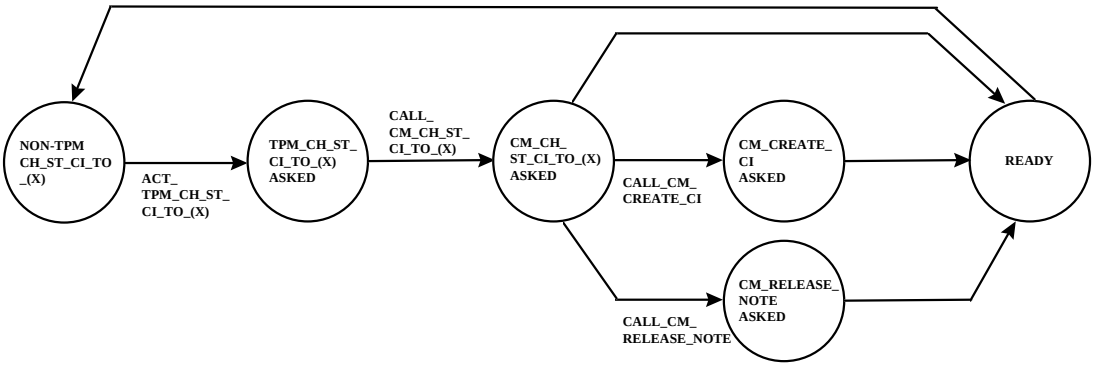


figure 4.11 int-tpm_ch_st_ci_to_(x) : internal behavior STD

With the operation ‘tpm_ch_st_ci_to_(x)’ the technical_project_manager directs the configuration_manager to change the status of the configuration_item. After the call ‘cm_ch_st_ci_to_(x)’ the technical_project_manager has the choice out of three actions. The choice depends on the status change that was asked.

If the status change was to ‘archive’ then the technical_project_manager directs the configuration_manager to create a new configuration_item object. The configuration_manager create a new configuration_item and copies the contents of the archived configuration_item to the new object. (The configuration_manager does this by calling the operation ‘create’ of the configuration_item with the appropriate parameters. The operation ‘create’ will create a new object and optionally copy the contents of an old configuration_item to a new configuration_item.)

If the status changes was to ‘baseline’ then the technical_project_manager checks if all configuration_items for a particular baseline have reached the status ‘baseline’ (not modeled here) and directs the configuration_manager to issue a release_note. In the case of other status changes, the technical_project_manager will go directly to its state ‘ready’.

4.3.2.1.3 Technical_Project_Manager : manager-STD

The communication between the technical_project_manager’s operations (callees) and their callers is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

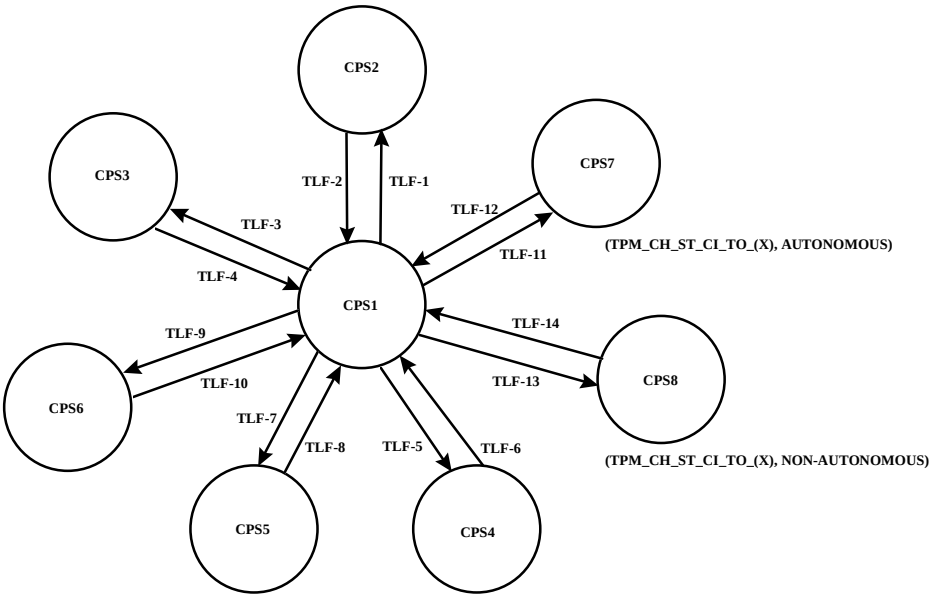


figure 4.12 technical_project_manager : manager STD

The notation CPSx in the STD stands for Consolidated Prescribed Subprocesses and is a set of CCx’s. The notation CCx in the STD stands for ‘Caller(s)-Callee’ and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for ‘Traps Logical Formula’ and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

In the state ‘neutral’ the CPS and the TLFs for the transitions leaving the state are :

CPS1 = {CC1-1, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1}
TLF-1 = T-1 (corresponds with ‘tpm_modify’ transition in extern STD)
TLF-3 = T-3 (corresponds with ‘tpm_review’ transition in extern STD)
TLF-5 = T-5 (corresponds with ‘tpm_dt_test’ transition in extern STD)
TLF-7 = T-7 (corresponds with ‘tpm_ac_test’ transition in extern STD)
TLF-9 = T-9 (corresponds with ‘tpm_modify’ transition in extern STD)
TLF-11= T-11 (corresponds with ‘tpm_ch_st_ci_to_(x)’ transition, autonomous)
TLF-13= T-11 and (T-13 or T-15) (corresponds with ‘tpm_ch_st_ci_to_(x)’ transition, non-autonomous)

In the state ‘starting tpm_modify’ the CPS and the TLF’s for the transitions leaving the state are :

CPS2 = {CC1-2, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1}
TLF-2 = T-2

In the state ‘starting tpm_review’ the CPS and the TLF’s for the transitions leaving the state are :

CPS3 = {CC1-1, CC2-2, CC3-1, CC4-1, CC5-1, CC6-1}
TLF-4 = T-4

In the state ‘starting tpm_dt_test’ the CPS and the TLF’s for the transitions leaving the state are :

CPS4 = {CC1-1, CC2-1, CC3-2, CC4-1, CC5-1, CC6-1}
TLF-6 = T-6

In the state ‘starting tpm_ac_test’ the CPS and the TLF’s for the transitions leaving the state are :

CPS5 = {CC1-1, CC2-1, CC3-1, CC4-2, CC5-1, CC6-1}
TLF-7 = T-7

In the state ‘starting tpm_cluster_pcr’ the CPS and the TLF’s for the transitions leaving the state are :

CPS6 = {CC1-1, CC2-1, CC3-1, CC4-1, CC5-2, CC6-1}
TLF-10= T-10

In the state ‘starting tpm_ch_st_ci_to_(x)’ (autonomous) the CPS and the TLF’s for the transitions leaving the state are :

CPS7 = {CC1-1, CC2-1, CC3-1, CC4-1, CC5-1, CC6-2}
TLF-12= T-12

The state ‘disc_starting tpm_ch_st_ci_to_(x)’ (non-autonomous) is a discriminator state. The possible callers are ‘cu_acceptance_test’ and ‘tpm_cluster_pcr’. In the discriminator state the manager STD will determine which caller has placed the call and will prescribe the required subprocesses for his caller and the callee. The prescribed subprocess of the other caller does not change. The CPS and the TLF’s for the transitions leaving the state are :

CPS8 = {CC1-1, CC2-1, CC3-1, CC4-1, CC5-1, CC6-3}
TLF-14= T-12 and (T-14 or T-16)

The caller-callee combinations for ‘tpm_modify’ consist only of prescribed subprocesses of the callee ‘tpm_modify’. This is because ‘tpm_modify’ is an autonomous operation.

CC1-1 = {S1}
CC1-2 = {S2}

The caller-callee combinations for ‘tpm_review’ consist only of prescribed subprocesses of the callee ‘tpm_review’. This is because ‘tpm_review’ is an autonomous operation.

CC2-1 = {S3}
CC2-2 = {S4}

The caller-callee combinations for ‘tpm_dt_test’ consist only of prescribed subprocesses of the callee ‘tpm_dt_test’. This is because ‘tpm_dt_test’ is an autonomous operation.

CC3-1 = {S5}
CC3-2 = {S6}

The caller-callee combinations for ‘tpm_ac_test’ consist only of prescribed subprocesses of the callee ‘tpm_ac_test’. This is because ‘tpm_ac_test’ is an autonomous operation.

CC4-1 = {S7}
CC4-2 = {S8}

The caller-callee combinations for ‘tpm_cluster_pcr’ consist only of prescribed subprocesses of the callee ‘tpm_cluster_pcr’. This is because ‘tpm_cluster_pcr’ is an autonomous operation.

CC5-1 = {S9}
CC5-2 = {S10}

The operation ‘tpm_ch_st_ci_to_(x)’ can be started autonomously and it can be called by ‘cu_acceptance_test’ or ‘tpm_cluster_pcr’. The CCs are therefore :

CC6-1 = {S11, S13, S15}
CC6-2 = {S12, S13, S15}
CC6-3 = {S12, S14, S15} or
{S12, S13, S16}

4.3.2.1.4 Technical_Project_Manager : employee-STDs

Internal operations are normally started with the caller_callee-construct. In the case of autonomous operations the caller_callee-construct consists only of the callee-part (i.e. the act-construct). The caller_callee-construct, the act-construct and autonomous behavior are described in more detail in the chapter explaining the SOCCA concepts.

This manager STD has 8 employees. The operations ‘tpm_modify’, ‘tpm_review’, ‘tpm_cluster_pcr’, ‘ tpm_ac_test’ and ‘tpm_dt_test’ are autonomous operations. They are started with the act-construct. The operation ‘tpm_ch_st_ci_ to_(x)’ can be autonomous, in which case it is started with the act-construct. It can can also be called by ‘cu_acceptance_test’ and ‘tpm_cluster_pcr’. In that case the act-construct which starts the callee ‘tpm_ch_st_ci_to_(x)’ is part of a caller_callee-construct. The caller ‘cu_acceptance_test’ or ‘tpm_cluster_pcr’ is the caller-part of this caller_callee-construct.

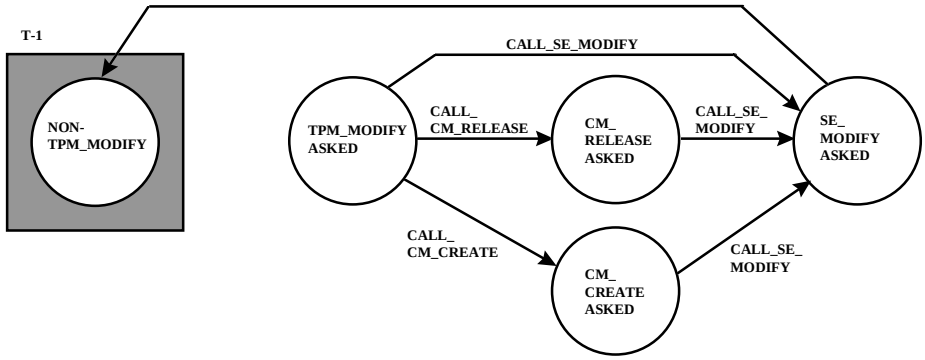


figure 4.13 employee int-tpm_modify : subprocess S1

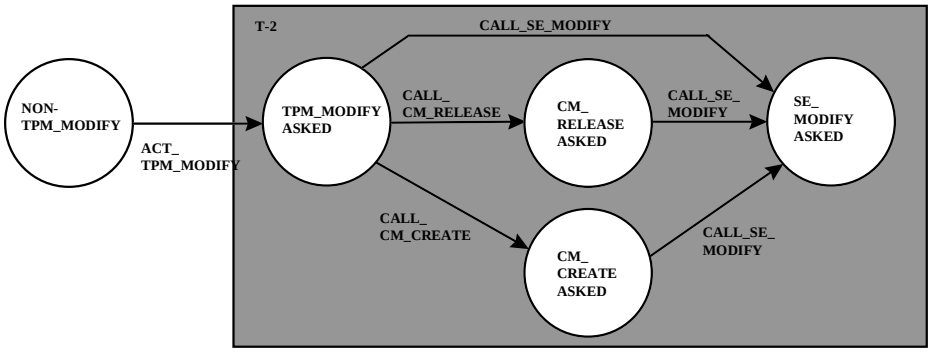


figure 4.14 employee int-tpm_modify : subprocess S2

When the act-construct is applied to the first employee, the autonomous operation ‘tpm_modify’, the result is the following. This employee has two subprocesses S1 and S2, and two traps T-1 and T-2. In its neutral state the manager prescribes the subprocess S1 for this employee. The manager can transit to the state ‘starting tpm_modify’ when the subprocess S1is in trap T-1 and the technical_project_manager decides autonomously to perform the operation ‘tpm_modify’. In the state ‘starting tpm_modify’ the manager prescribes the subprocess S2. This means that the employee can make its transition ‘act_tpm_modify’ (activate_tpm_modify) which means that the operation can start executing. As soon as the subprocess S2 enters in trap T-2, the manager can transit back to the neutral state. The manager can be back in the neutral state before the operation ‘tpm_modify’ has finished executing. In the neutral state another operation can be started to run concurrently with ‘tpm_modify’. Or another instance of ‘tpm_modify’ can be started because the internal STD of ‘tpm_modify’ has the default muliplicity of concurrency of zero or more.

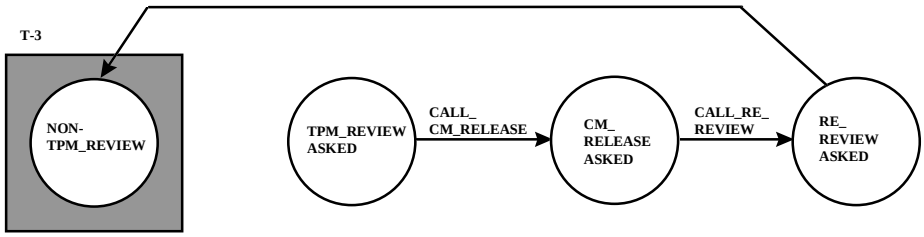


figure 4.15 employee int-tpm_review : subprocess S3

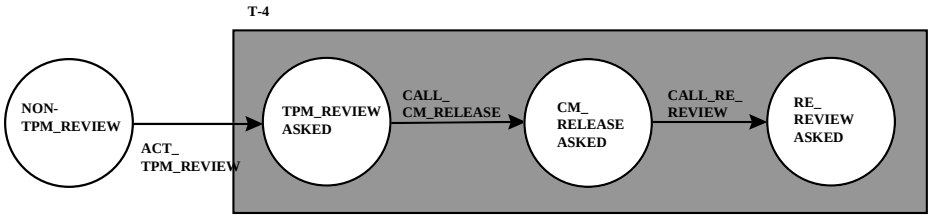


figure 4.16 employee int-tpm_review : subprocess S4

The application of the act-construct to the second employee, the autonomous operation ‘tpm_review’, results in the same structure as that of ‘tpm_modify’. The employee ‘tpm_review’ has two subprocesses S3 and S4, and two traps T-3 and T-4. In its neutral state the manager prescribes the subprocess S3 for this employee. The manager can transit to the state ‘starting tpm_review’ when the subprocess S3 is in trap T-3 and the technical_project_manager decides autonomously to perform the operation ‘tpm_review’. In the state ‘starting tpm_modify’ the manager prescribes the subprocess S4. This means that the employee can make its transition ‘act_tpm_review’ which means that the operation can start executing. As soon as the subprocess S4 enters in trap T-4, the manager can transit back to the neutral state. It is then ready to service another call.

The application of the act-construct to the third employee, the autonomous operation ‘tpm_dt_test’, results in the same structure as that of ‘tpm_modify’. The employee ‘tpm_dt_test’ has two subprocesses S5 and S6 and two traps T-5 and T-6.

The application of the act-construct to the fourth employee, the autonomous operation ‘tpm_ac_test’, results in the same structure as that of ‘tpm_modify’. The employee ‘tpm_ac_test’ has two subprocesses S7 and S8 and two traps T-7 and T-8.

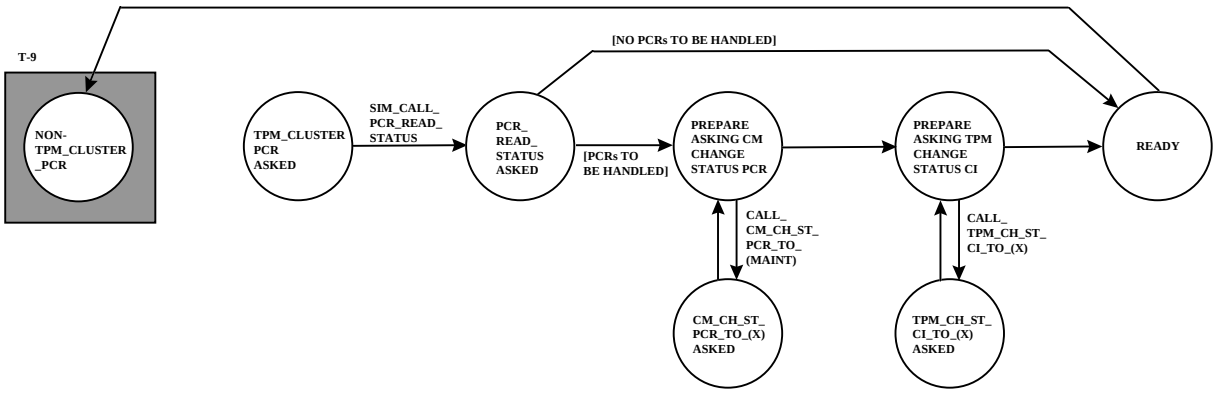


figure 4.17 employee int-tpm_cluster_pcr : subprocess S9

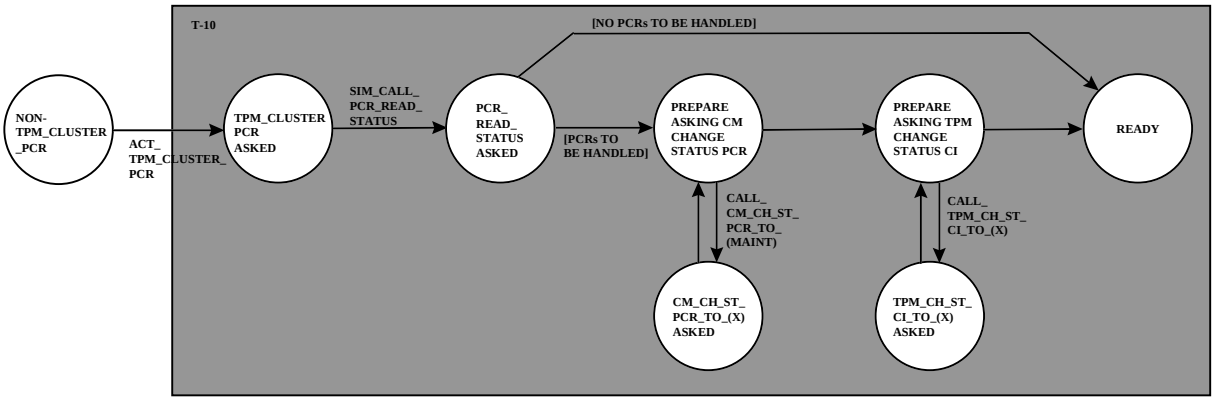


figure 4.18 employee int-tpm_cluster_pcr : subprocess S10

The application of the act-construct to the fifth employee, the autonomous operation ‘tpm_cluster_pcr’, results in the same structure as that of ‘tpm_modify’. The employee ‘tpm_cluster_pcr’ has two subprocesses S9 and S10 and two traps T-9 and T-10.

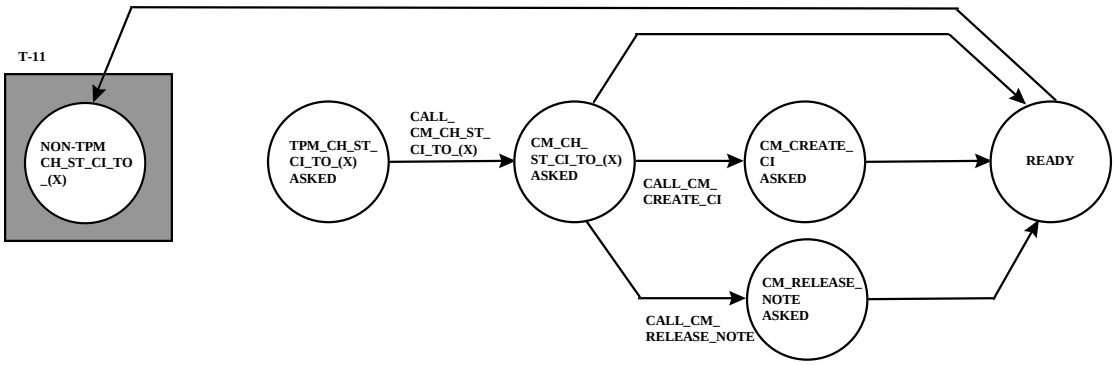


figure 4.19 employee int-tpm_ch_st_ci_to_(x) : subprocess S11

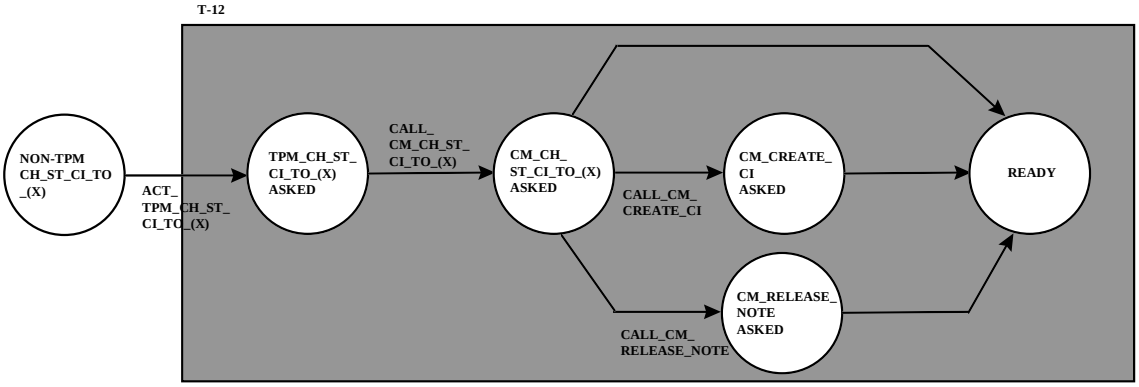


figure 4.20 employee int-tpm_ch_st_ci_to_(x) : subprocess S12

The sixth employee of the manager STD is the ‘tpm_ch_st_ci_to_(x)’ operation. This operation can be called autonomously by the technical_project_manager or by the ‘cu_acceptance_test’ operation of the customer or by the ‘tpm_cluster_pcr’ operation of the technical_project_manager itself. In all cases it is started with the act-construct. When the operation ‘tpm_ch_st_ci_to_(x)’ is used autonomously, the act-construct is ‘stand-alone’. When the operation is called by ‘cu_acceptance_test’ or ‘tpm_cluster_pcr’ the act-construct is part of the caller_callee-construct.

The application of the act-construct to the employee, the autonomous operation ‘tpm_ch_st_ci_to_(x)’, results in the same structure as that of ‘tpm_modify’. The employee ‘tpm_ch_st_ci_to_(x)’ has two subprocesses S11 and S12 and two traps T-11 and T-12. In its neutral state the manager prescribes the subprocess S11 for this employee. The manager can transit to the state ‘starting tpm_ch_st_ci_to_(x) (autonomous)’ when the subprocess S11 is in trap T-11 and the technical_project_manager decides autonomously to perform the operation ‘tpm_ch_st_ci_to_(x)’. In the state ‘starting

tpm_ch_st_to_ci_(x) (autonomous)' the manager prescribes the subprocess S12. This means that the employee can make its transition 'act_tpm_ch_st_ci_to_(x)' which means that the operation can start executing. As soon as the subprocess S12 enters in trap T-12, the manager can transit back to the neutral state.

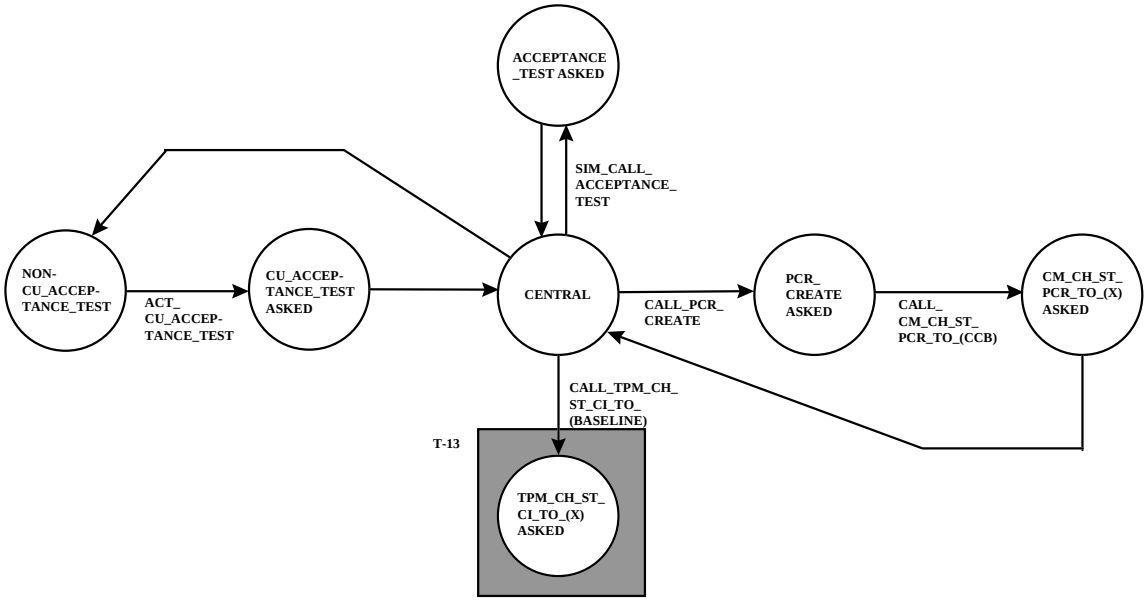


figure 4.21 employee int-cu_acceptance_test : subprocess S13

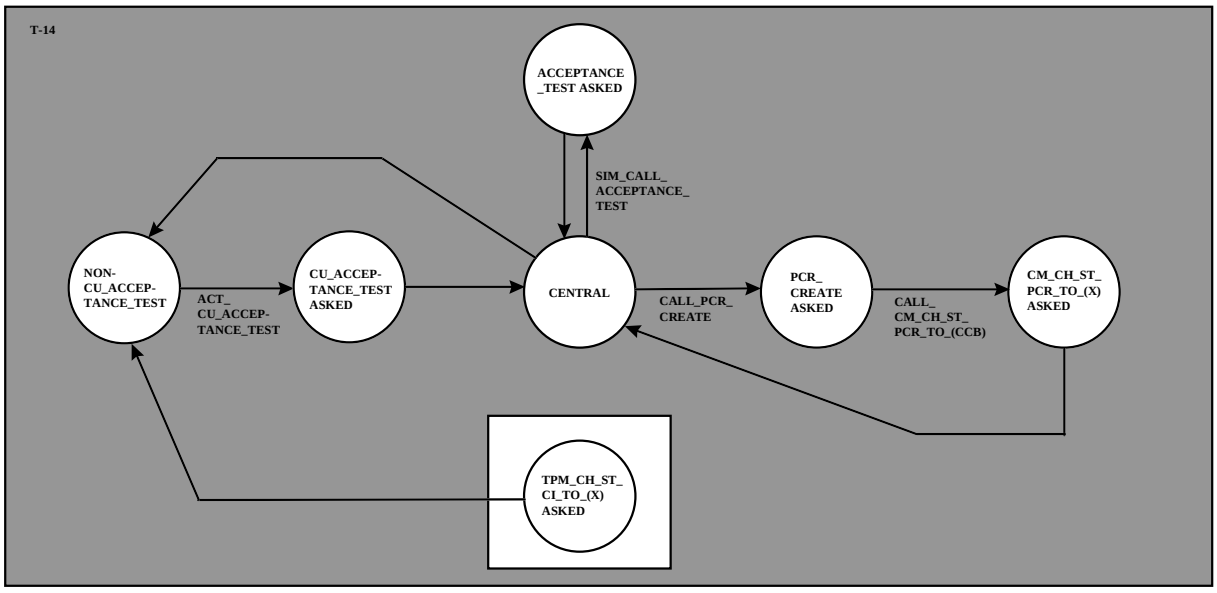


figure 4.22 employee int-cu_acceptance_test : subprocess S14

The seventh employee is the operation 'cu_acceptance_test' of the customer as part of the caller-callee pair 'cu_acceptance_test' and 'tpm_ch_st_ci_to_(x)'. The callee 'tpm_ch_st_to_(x)' is started with the act-construct which is part of the caller_callee construct. It is identical with the sixth employee and has the same has two subprocesses S11 and S12 and two traps T-11 and T-12. The application of the caller_callee-construct to the caller 'cu_acceptance_test' results in the subprocesses S13 and S14 and two traps T-13 and T-14 for this caller.

In its neutral state the manager prescribes the subprocess S13 for 'cu_acceptance_test' and the subprocess S11 for 'tpm_ch_st_ci_to(x)' (This is the same as for the autonomous use 'tpm_ch_st_ci_to_(x)'). The manager can transit to the state 'disc_starting tpm_ch_st_ci_to_(x) (non_autonomous)' when the subprocess S13 is in trap T-13 and the subprocess S11 is in trap T-11. In the state 'disc_starting tpm_ch_st_to_ci_(x) (non_autonomous)' the manager prescribes the subprocess S14 for 'cu_acceptance_test' and subprocess S12 for 'tpm_ch_st_ci_to_(x)'. This means that the operation 'cu_acceptance_test' can proceed executing its next partial behavior. And 'tpm_ch_st_ci_to_(x)' can make its transition

'act_tpm_ch_st_ci_to_(x)' which means that the operation can start executing. As soon as the subprocess S12 enters in trap T-12 and subprocess S14 enters in trap T-14, the manager can transit back to the neutral state.

By calling the operation 'tpm_ch_st_ci_to_(x)' the customer in fact informs the technical_project_manager that the acceptance test was ok and that the status of the configuration_item that was tested can be changed to 'baseline'. The customer does not expect a response from the technical_project_manager. After the customer has placed the call and the technical_project_manager has started the internal operation 'tpm_ch_st_ci_to_(x)', the customer can proceed with its own behavior without waiting for any result from 'tpm_ch_st_ci_to_(x)'. The 'non waiting'-variant of the caller_callee-construct is used to model this.

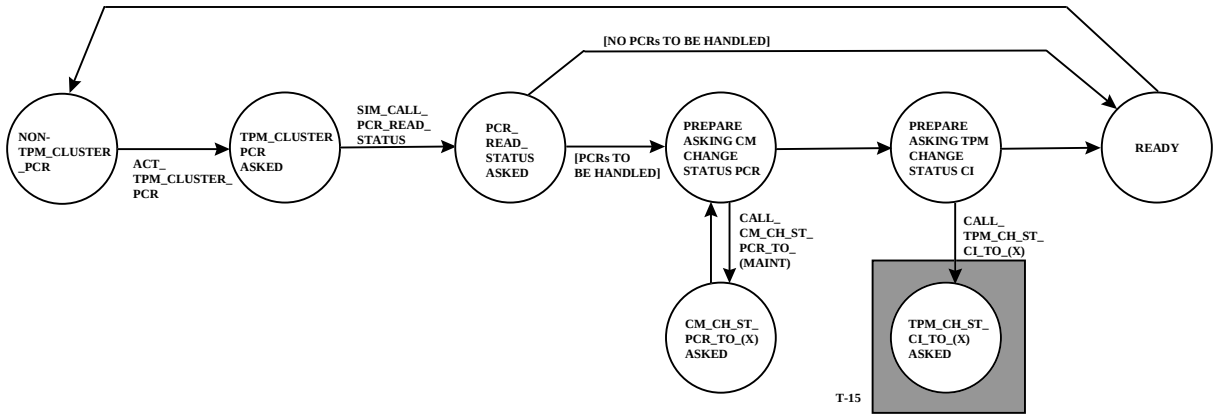


figure 4.23 employee int-tpm_cluster_pcr : subprocess S15

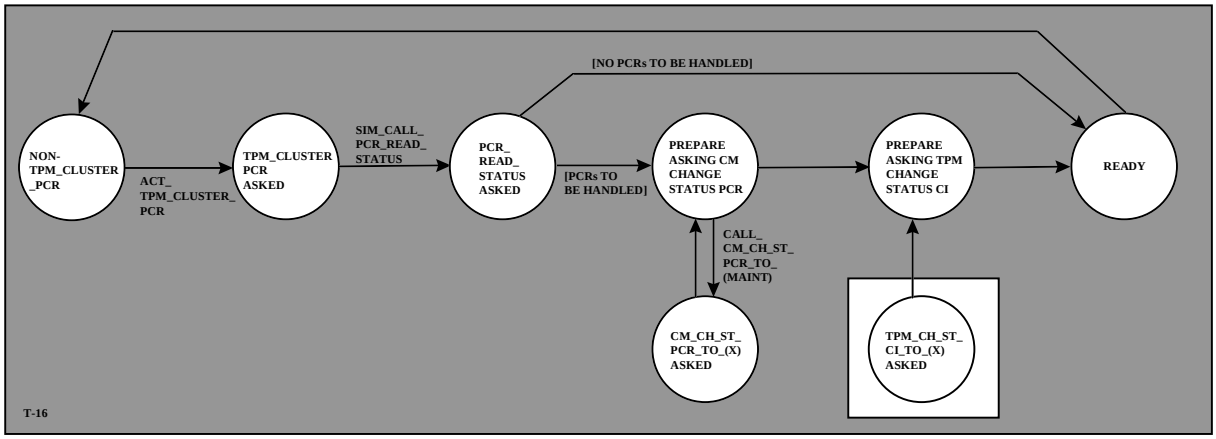


figure 4.24 employee int-pcr_cluster_pcr : subprocess S16

The 8th is the operation 'tpm_cluster_pcr' of the technical_project_manager as part of the caller-callee pair 'tpm_cluster _pcr' and 'tpm_ch_st_ci_to_(x)'. The callee 'tpm_ch_st_to_(x)' is started with the act-construct which is part of the caller_callee construct. It is identical with the sixth employee and has the same has two subprocesses S11 and S12 and two traps T-11 and T-12. The application of the caller_callee-construct to the caller 'tpm_cluster_pcr' results in the subprocesses S15 and S16 and two traps T-15 and T-16 for this caller.

After the operation 'tpm_cluster_pcr' has placed the call and the callee operation is started, the operation 'tpm_cluster_ pcr' can proceed with its own behavior without waiting for the result from 'tpm_ch_st_ci_to_(x)'. The 'non waiting'-variant of the caller_callee-construct is used to model this.

4.3.2.2 Configuration_Manager

4.3.2.2.1 Configuration_Manager : external behavior-STD

The STD of the external behavior consists of a neutral state in which the configuration_manager waits for a call to one of its operations, starting states in which he starts his operations after they are called, and discriminator states. If an operation that can be called by more than one operation, is called, the STD transits to a discriminator state. Here he determines from which operation the call is coming and transits to the corresponding starting state. The discriminator state concept is described in more detail in the chapter explaining the SOCCA concepts. The callers of the operations of this class can be found in the import-export diagram. They are given in the the 'import_list' attribute of the 'uses association'.

There are four discriminator states, indicated by the prefix 'disc_' in the state name. The first discriminator state 'disc_starting_cm_ch_st_ci_to_(x)' is given in more detail. This shows the handling of a call to the 'cm_ch_st_ci_to_(x)' operation. The operation 'cm_ch_st_ci_to_(x)' can be called by 4 different callers. They are the operation 'tpm_ch_st_ci_to_(x)' of the class 'technical_project_manager', the operation 'se_modify' of the class 'software_engineer', the operation 're_review' of the class 'reviewer' and the operation 'te_dt_test' of the class 'test_engineer'.

The second discriminator state is 'disc_starting_cm_release' handling calls from 4 different callers. These callers are the operations 'tpm_modify', 'tpm_review', 'tpm_dt_test' and 'tpm_ac_test' all from the class 'technical_project_manager'.

The third discriminator state is 'disc_starting_cm_ch_st_pcr_to_(x)' handling calls from 6 different callers. These callers are the operation 'tpm_cluster_pcr' of the class 'technical_project_manager', the operation 'te_dt_test' of the class 'test_engineer', the operations 'cu_acceptance_test' and 'cu_issue_pcr' of the class 'customer', the operation 'scb_consider_pcr' of the class 'software_configuration_board' and the operation 'ccb_consider_pcr' of the class 'configuration_control_board'.

The fourth discriminator state is 'disc_starting_cm_create_ci' handling calls from 2 different callers. These callers are the operation 'tpm_modify' and the operation 'tpm_ch_st_ci_to_(x)' both from the class 'technical_project_manager'.

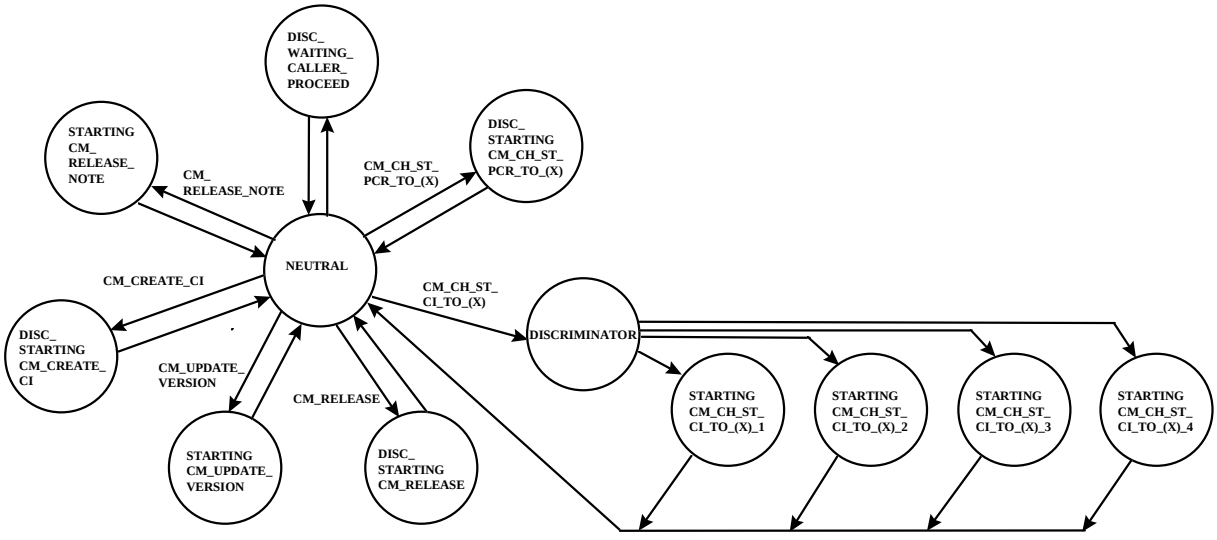


figure 4.25 configuration_manager : external behavior STD

The STD has also the state 'disc_waiting_caller_proceed'. Some of the callers of the operations of the configuration_manager have to wait for a result returned to them by these operations. Or they have to wait until these called operations have reached some specific point in their execution. The state 'disc_waiting_caller_proceed' is an aggregate state. When a callee has produced a result (or is far enough in its execution), the manager transits to the state 'disc_waiting_ caller_proceed'. Here the manager determines which waiting caller had called this callee. This caller is then allowed to proceed.

The callers of ‘cm_release’ are ‘tpm_modify’, ‘tpm_review’, ‘tpm_dt_test’ and ‘tpm_ac_test’. These callers must wait after their call until ‘cm_release’ has progressed far enough in its execution. I.e. until the configuration_item is released.

The callers of ‘cm_create_ci’ are ‘tpm_modify’ and ‘tpm_ch_st_ci_to_(x)’. These callers must wait after their call until ‘cm_create_ci’ has progressed far enough in its execution. I.e. until the configuration_item is created.

The caller of ‘cm_update_version’ is ‘se_modify’. This caller must wait after its call until ‘cm_update_version’ has progressed far enough in its execution. I.e. until the version of the configuration_item is updated.

The caller of ‘cm_release_note’ is ‘tpm_ch_st_ci_to_(x). It does not have to wait on some result of ‘cm_release_note’, but can continue right after its call to ‘cm_release_note’.

The callers of ‘cm_ch_st_ci_to_(x)’ are ‘tpm_ch_st_ci_to_(x)’, ‘se_modify’, ‘re_review’ and ‘te_dt_test’. These callers don’t have to wait for a result of ‘cm_ch_st_ci_to_(x)’. Firstly, they don’t have to work themself on the CI anymore, so they are not interested if the status change has been performed on the CI. Secondly, the next person to work on a particular CI does so only after this CI that has been ‘released’ from configuration management. A CI can only be ‘released’ when it has the correct status. This is enforced by the external STD of the configuration item (which specifies the sequence in which calls to the configuration item are serviced). So either the status change is already applied to the CI, and then it can be released and worked on, or the status change has not yet been applied to the CI and then it can not yet be released (the call to the operation ‘cm_release’ is not yet serviced).

The callers of ‘cm_ch_st_pcr_to_(x)’ are ‘tpm_cluster_pcr’, ‘te_dt_test’, ‘cu_acceptance_test’, ‘cu_issue_pcr’, ‘scb_consider_pcr’ and ‘ccb_consider_pcr’. These callers don’t have to wait for a result of ‘cm_ch_st_pcr_to_(x)’. Firstly, they don’t have any dealings with the PCR anymore, so they are not interested if the status change has been performed on the PCR. Secondly, for the persons going to perform some action on a PCR there are two possibilities. In one case they check the status of a PCR before they are using it. This is the case with the technical_project_manager when he clusters the PCRs. He will only cluster those PCRs that have the correct status. In the second case, the software_configuration_board and the configuration_control_board, the PCRs are given to these boards by the configuration_manager. He will do so only after the PCRs have gotten their correct status. The configuration manager indeed has to wait when he calls the status changing operation of the class ‘problem_and_change_report’.

4.3.2.2.2 Configuration_Manager : internal behavior-STDs

The 6 operations of the configuration_manager have the following internal behavior STDs.

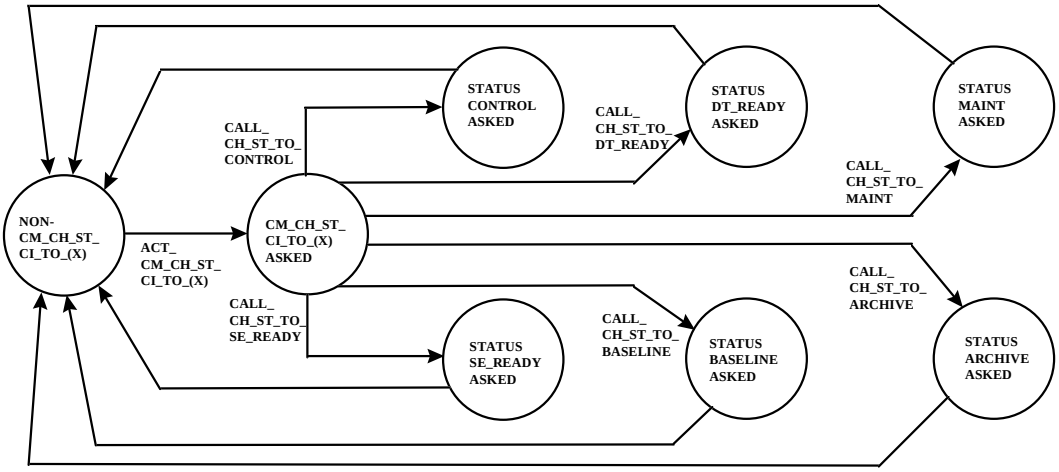


figure 4.26 int-cm_ch_st_ci_to_(x) : internal behavior STD

The formal parameters of the operation ‘cm_ch_st_ci_to_(x)’ are the id of the configuration_item whose status has to be changed and the new status. The STD calls the relevant configuration_item’s operation ‘ch_st_to_x’ according to the value of the parameter with which it is called.

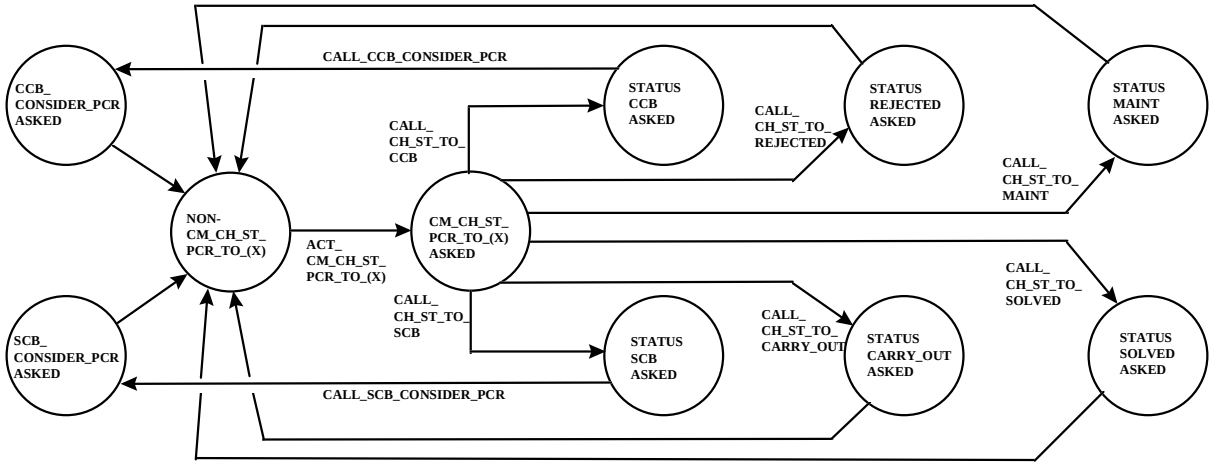


figure 4.27 int-cm_ch_st_pcr_to_(x) : internal behavior STD

The formal parameters of the operation ‘cm_ch_st_pcr_to_(x)’ are the id of the problem_and_change_report whose status has to be changed and the new status. The STD calls the relevant problem_and_change_report’s operation ‘ch_st_to_x’ according to the value of the parameter with which it is called. After a status change to ‘ccb’ the configuration_manager offers the pcr to the configuration_control_board and and after a status change to ‘scb’ he offers the pcr to the software_configuration_board.

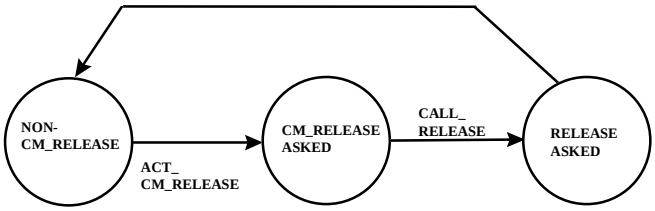


figure 4.28 int-cm_release : internal behavior STD

The operation ‘cm_release’ is calling the configuration_item’s operation ‘release’ on behalf of its callers.

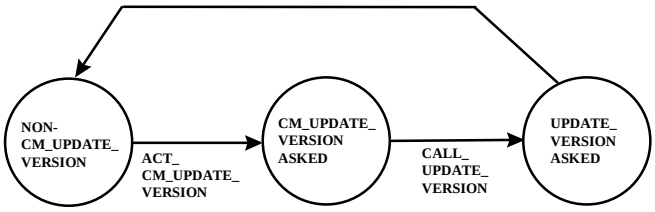


figure 4.29 int-cm_update_version : internal behavior STD

The operation ‘cm_update_version’ is calling the configuration_item’s operation ‘update_version’ on behalf of its callers.

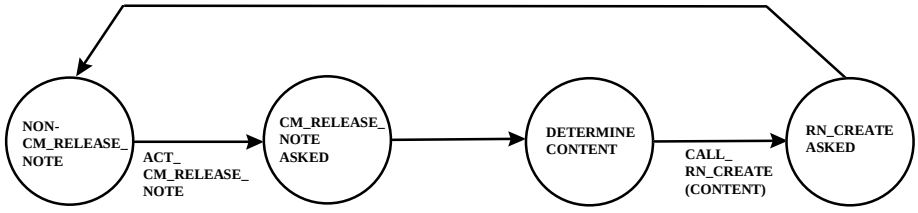


figure 4.30 int-cm_release_note : internal behavior STD

The operation ‘cm_release_note’ is calling the release_note’s operation ‘rn_create_(x)’ on behalf of its callers. When a release_note is created its attribute ‘content’ is initiated with the actual value of the patameter ‘x’. This models the writing of the release_note by the configuration_manager.

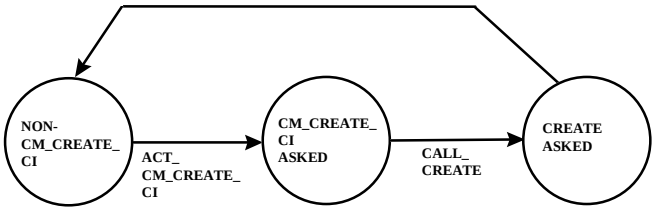


figure 4.31 int-cm_create_ci : internal behavior STD

With the operation 'cm_create_ci' the configuration_manager creates a new configuration_item. The configuration_ manager does this by calling the operation 'create' of the class 'configuration_item'. If needed the configuration-manager can let the operation 'create' initialize the content of the new configuration_item with the content of some already existing configuration_item. The configuration_manager has then to call the operation 'create' with the appropriate parameters.

4.3.2.2.3 Configuration_Manager : manager-STD

The communication between the configuration_manager's operations (callees) and their callers is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

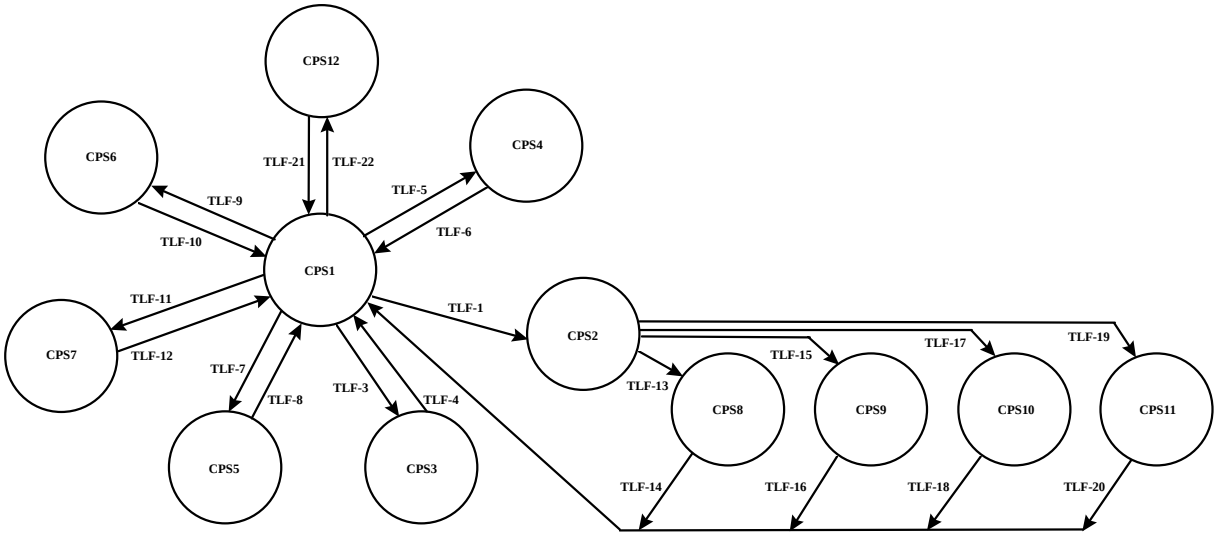


figure 4.32 configuration_manager : manager STD

The notation CPSx in the STD stands for Consolidated Prescribed Subprocesses and is a set of CCx's. The notation CCx in the STD stands for 'Caller(s)-Callee' and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for 'Traps Logical Formula' and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

The caller-waits construct is modeled by the 2^e variant of this construct (i.e. the 'waiting_caller_proceed'-construct). This means that the TLF-3, TLF-7, TLF-11 and TLF-22 have to have some additional information for the manager STD to decide which transition to take. This information comes from the internal bookkeeping of the manager STD. If an operation is started by the manager STD on behalf of a caller, a caller-callee relation is initiated. In this way the manager STD can check whether a non-active employee still has some caller waiting to be allowed to proceed. If both the callee and the caller have terminated, the particular caller-callee relation is cancelled in the internal administration of the manager STD.

In the state ‘neutral’ the CPS and the TLFs for the transitions leaving the state are :

CPS1 = {CC1-1, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1}
TLF-1 = T-1 and (T-3 or T-5 or T-7 or T-9)
TLF-3 = (T-11 and ((T-13 and not(caller-callee relation) or
(T-15 and not(caller-callee relation) or
(T-17 and not(caller-callee relation) or
(T-19 and not(caller-callee relation)))
TLF-5 = T-21 and (T-23 or T-25 or T-27 or T-29 or T-31 or T47)
TLF-7 = T-33 and (T-35 and not(caller-callee relation))
TLF-9 = T-37 and T-39
TLF-11= (T-24 and ((T-43 and not(caller-callee relation) or
(T-45) and not caller-callee relation)))
TLF-22 = (T-11 and ((T-13 and (caller-callee relation) or
(T-15 and (caller-callee relation) or
(T-17 and (caller-callee relation) or
(T-19 and (caller-callee relation)))
or (T-33 and (T-35 and (caller-callee-relation)))
or (T-24 and ((T-43 and not(caller-callee relation) or
(T-45) and not caller-callee relation)))

The discriminator state ‘disc_starting_cm_ch_st_ci_to_(x)’ is shown in more detail. It consists of the states ‘discriminator’, ‘starting_cm_ch_st_ci_to_(x)_1’, ‘starting_cm_ch_st_ci_to_(x)_2’, ‘starting_cm_ch_ch_st_ci_to_(x)_3’ and ‘starting_cm_ch_st_ci_to_(x)_4’.

The STD starts in its neutral state. When the TLF-1 becomes true the STD can transit to the state ‘discriminator’. From here it can transit to the state ‘starting_cm_ch_st_ci_to_(x)_1’ when trap T-3 has been entered, to ‘starting_cm_ch_st_ci_to_(x)_2’ when trap T-5 has been entered, to ‘starting_cm_ch_ch_st_ci_to_(x)_3’ when trap T-7 has been entered or to ‘starting_cm_ch_st_ci_to_(x)_4’ when trap T-9 has been entered. In these states the operation ‘cm_ch_st_ci_to_(x)’ is started and the caller is set to its ‘continue’-subprocess while the non-callers stay in their subprocesses as prescribed in ‘neutral’ state. If the internal operation has started, i.e. entered T-2 and the caller has entered its ‘continue’-trap, T-4 or T-6 or T-8 or T10, the manager STD can transit back to the state ‘neutral’.

In the state ‘discriminator’ the CPS and the TLFs for the transitions leaving the state are :

CPS2 = {CC1-1, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1}
TLF-13 = T-3
TLF-15 = T-5
TLF-17 = T-7
TLF-19 = T-9

In the state ‘starting_cm_ch_st_ci_to_(x)_1’ the CPS and the TLFs for the transitions leaving the state are :

CPS8 = {CC1-2, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1}
TLF-14 = T-2 and T-4

In the state ‘starting_cm_ch_st_ci_to_(x)_2’ the CPS and the TLFs for the transitions leaving the state are :

CPS9 = {CC1-3, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1}
TLF-16 = T-2 and T-6

In the state ‘starting_cm_ch_st_ci_to_(x)_3’ the CPS and the TLFs for the transitions leaving the state are :

CPS10 = {CC1-4, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1}
TLF-18 = T-2 and T-8

In the state ‘starting_cm_ch_st_ci_to_(x)_4’ the CPS and the TLFs for the transitions leaving the state are :

CPS11 = {CC1-5, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1}
TLF-20 = T-2 and T-10

In the discriminator state ‘disc_starting_cm_release’ the CPS and the TLFs for the transitions leaving the state are :

CPS3 = {CC1-1, CC2-2, CC3-1, CC4-1, CC5-1, CC6-1}
TLF-4 = T-12

In the discriminator state ‘disc_starting_cm_ch_st_pcr_to_(x)’ the CPS and the TLFs for the transitions leaving the state are :

CPS4 = {CC1-1, CC2-1, CC3-2, CC4-1, CC5-1, CC6-1}
TLF-6 = T-22 and (T-24 or T-26 or T-28 or T-30 or T-32 or T48)

In the state ‘starting_cm_update_version’ the CPS and the TLFs for the transitions leaving the state are :

CPS5 = {CC1-1, CC2-1, CC3-1, CC4-2, CC5-1, CC6-1}
TLF-8 = T-34

In the state ‘starting_cm_release_note’ the CPS and the TLFs for the transitions leaving the state are :

CPS6 = {CC1-1, CC2-1, CC3-1, CC4-1, CC5-2, CC6-1}
TLF-10 = T-38 and T-40

In the discriminator state ‘disc_starting_cm_create_ci’ the CPS and the TLFs for the transitions leaving the state are :

CPS7 = {CC1-1, CC2-1, CC3-1, CC4-1, CC5-1, CC6-2}
TLF-12 = T-42

The discriminator state ‘disc_waiting_caller_proceed’ is entered by the manager STD after it detects that some employee has returned some result (or has progressed far enough in its execution). The manager STD decides in this state which caller is allowed to proceed. The CPS and the TLFs for the transitions leaving the state are :

CPS12 = {CC1-1, CC2-3, CC3-1, CC4-1, CC5-1, CC6-2} or
{CC1-1, CC2-1, CC3-1, CC4-3, CC5-1, CC6-2} or
{CC1-1, CC2-1, CC3-1, CC4-1, CC5-1, CC6-3}
TLF-12 = (T-14 or T-16 or T-18 or T-20) or T-36 or (T-44 or T-46)

The caller-callee combinations for ‘cm_ch_st_ci_to_(x)’ and its 4 callers ‘tpm_ch_st_ci_to_(x)’, ‘se_modify’, ‘re_review’ and ‘te_dt_test’ are :

CC1-1 = {S1, S3, S5, S7, S9}
CC1-2 = {S2, S4, S5, S7, S9}
CC1-3 = {S2, S3, S6, S7, S9}
CC1-4 = {S2, S3, S5, S8, S9}
CC1-5 = {S2, S3, S5, S7, S10}

The caller-callee combinations for ‘cm_release’ and its 4 callers ‘tpm_modify’, ‘tpm_review’, ‘tpm_dt_test’ and ‘te_ac_test’ are :

CC2-1 = {S11, S13, S15, S17, S19}
CC2-2 = {S12, S13, S15, S17, S19} (either one of the callers has to wait)
CC2-3 = {S11, S14, S15, S17, S19}or (letting caller ‘tpm-modify’ proceed)
{S11, S13, S16, S17, S19}or (letting caller ‘tpm_review’ proceed)
{S11, S13, S15, S18, S19}or (letting caller ‘tpm_dt_test’ proceed)
{S11, S13, S15, S17, S20} (letting caller ‘tpm_ac_test’ proceed)

The caller-callee combinations for ‘cm_ch_st_pcr_to_(x)’ and its 6 callers ‘tpm_cluster_pcr’, ‘te_dt_test’, ‘cu_acceptance_test’, ‘scb_consider_pcr’, ‘ccb_consider_pcr’ and ‘cu_issue_pcr’ are :

CC3-1 = {S21, S23, S25, S27, S29, S31, S47}
CC3-2 = {S22, S24, S25, S27, S29, S31, S47}or
{S22, S23, S26, S27, S29, S31, S47}or

{S22, S23, S25, S28, S29, S31, S47}or
{S22, S23, S25, S27, S30, S31, S47}or
{S22, S23, S25, S27, S29, S32, S47}or
{S22, S23, S25, S27, S29, S31, S48}

The caller-callee combinations for ‘cm_update_version’ and its caller ‘se_modify’ are :

CC4-1 = {S33, S35}
CC4-2 = {S34, S35} (caller has to wait)
CC4-3 = {S33, S36} (letting caller ‘se_modify’ proceed)

The caller-callee combinations for ‘cm_release_note’ and its caller ‘tpm_ch_st_ci_to_(x)’ are :

CC5-1 = {S37, S39}
CC5-2 = {S38, S40}

The caller-callee combinations for ‘cm_create_ci’ and its 2 callers ‘tpm_modify’ and ‘tpm_ch_st_ci_to_(x)’ are :

CC6-1 = {S41, S43, S45}
CC6-1 = {S42, S43, S45} (either one of the callers has to wait)
CC6-3 = {S41, S44, S45}or} (letting caller ‘tpm_modify’ proceed)
{S41, S43, S46}} (letting caller ‘tpm_ch_st_ci_to_(x)’ proceed)

4.3.2.2.4 Configuration_Manager : employee-STDs

The manager STD has the following 24 employee STDs.

The first five employees are the callee ‘cm_ch_st_ci_to_(x)’ and its 4 callers ‘tpm_ch_st_ci_to_(x)’, ‘se_modify’, ‘re_review’ and ‘te_dt_test’. Which caller has executed the call is determined in the discriminator state ‘disc_starting_cm_ch_st_ci_to_(x)’ of the manager STD.

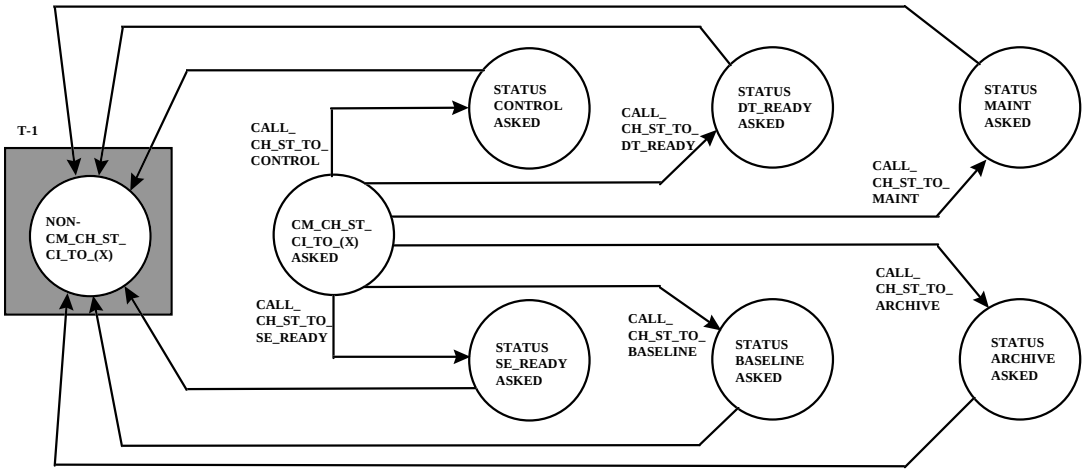


figure 4.33 employee int-cm_ch_st_ci_to_(x) : subprocess S1

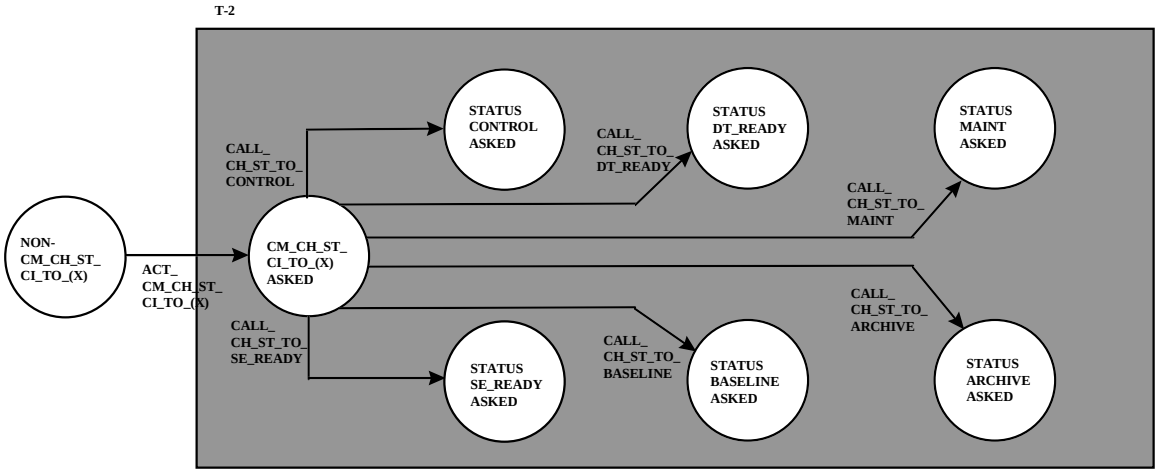


figure 4.34 employee int-cm_ch_st_ci_to_(x) : subprocess S2

The first employee is the own internal operation ‘cm_ch_st_ci_to_(x)’. This employee has two subprocesses S1 and S2 and two traps T-1 and T-2. The operation ‘cm_ch_st_ci_to_(x)’ is started after it has been determined which of its potential callers has executed the call.

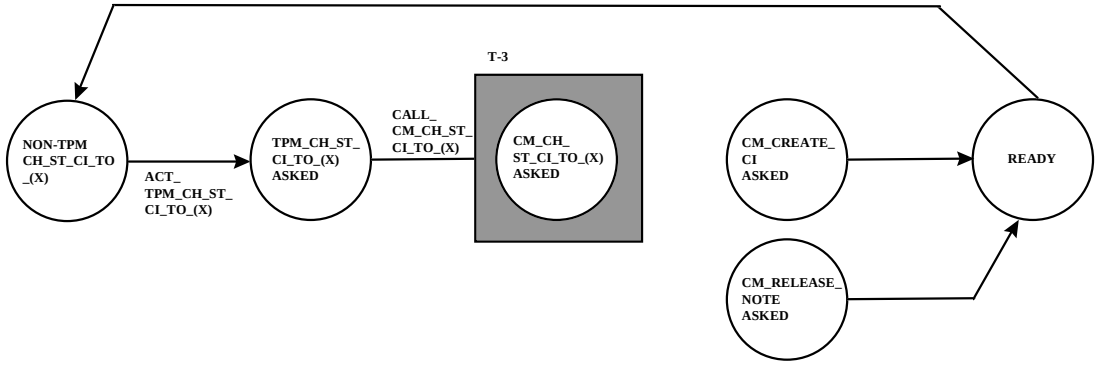


figure 4.35 employee int-tpm_ch_st_ci_to_(x) : subprocess S3

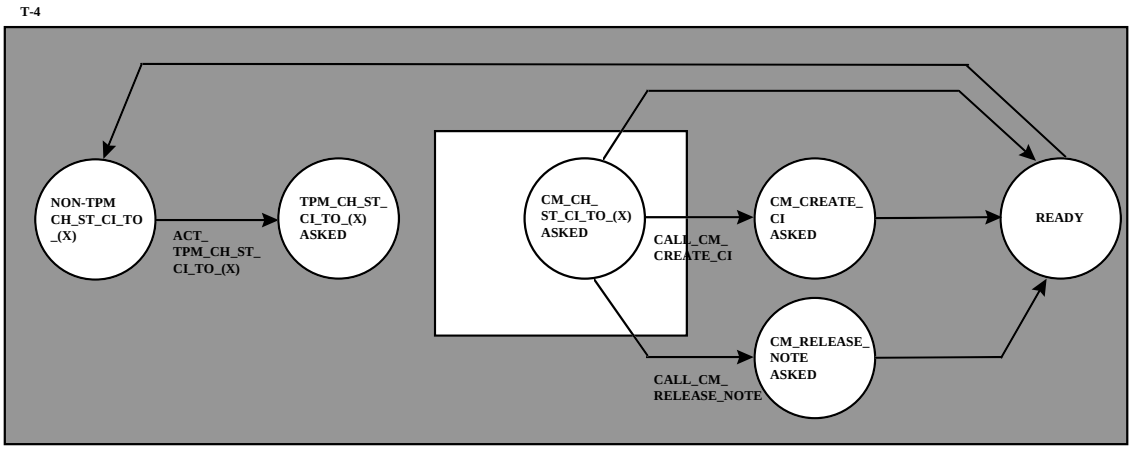


figure 4.36 employee int-tpm_ch_st_ci_to_(x) : subprocess S4

The second employee is the caller operation ‘tpm_ch_st_ci_to_(x)’. This employee has two subprocesses S3 and S4 and two traps T-3 and T-4. This caller does not have to wait after the call on some result from the callee (see the explanation in the external STD-subparagraph).

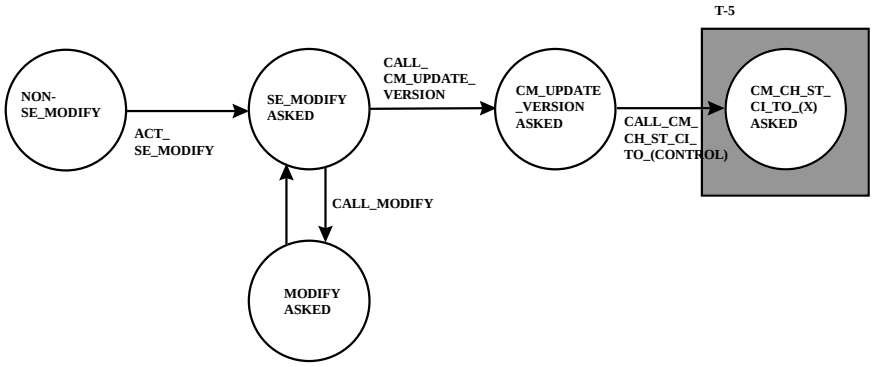


figure 4.37 employee int-se_modify : subprocess S5

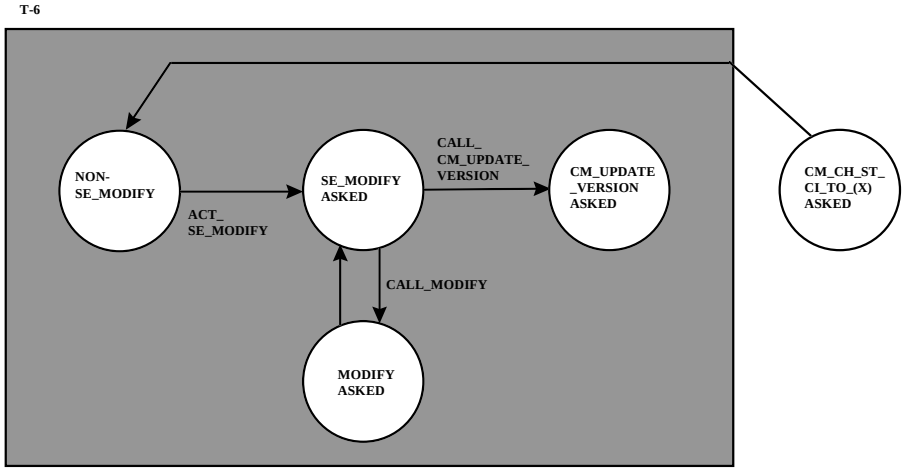


figure 4.38 employee int-se_modify : subprocess S6

The third employee is the caller operation ‘se_modify’. This employee has two subprocesses S5 and S6 and two traps T-5 and T-6. This caller does not have to wait after the call on some result from the callee (see the explanation in the external STD-subparagraph).

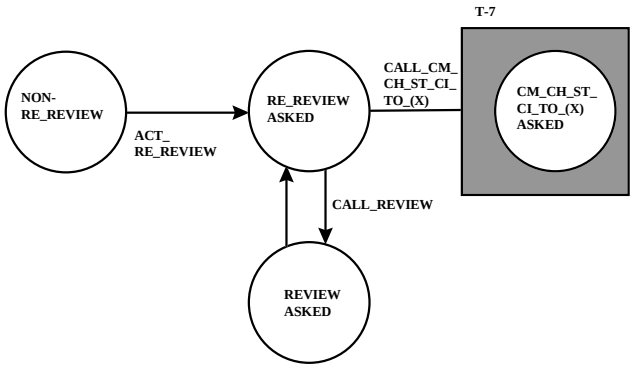


figure 4.39 employee int-re_review : subprocess S7

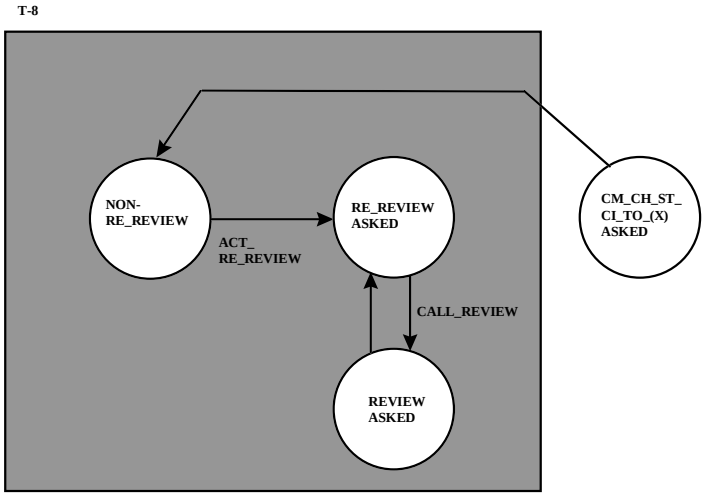


figure 4.40 employee int-re_review : subprocess S8

The 4th employee is the caller operation ‘re_review’. This employee has two subprocesses S7 and S8 and two traps T-7 and T-8. This caller does not have to wait after the call on some result from the callee (see the explanation in the external STD-subparagraph).

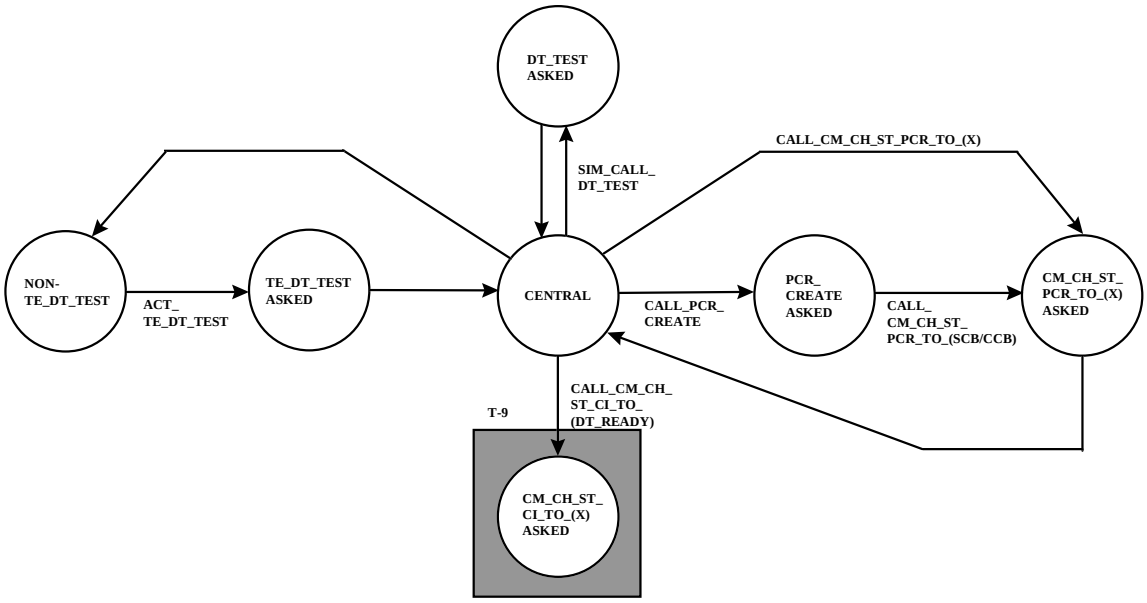


figure 4.41 employee int-te_dt_test : subprocess S9

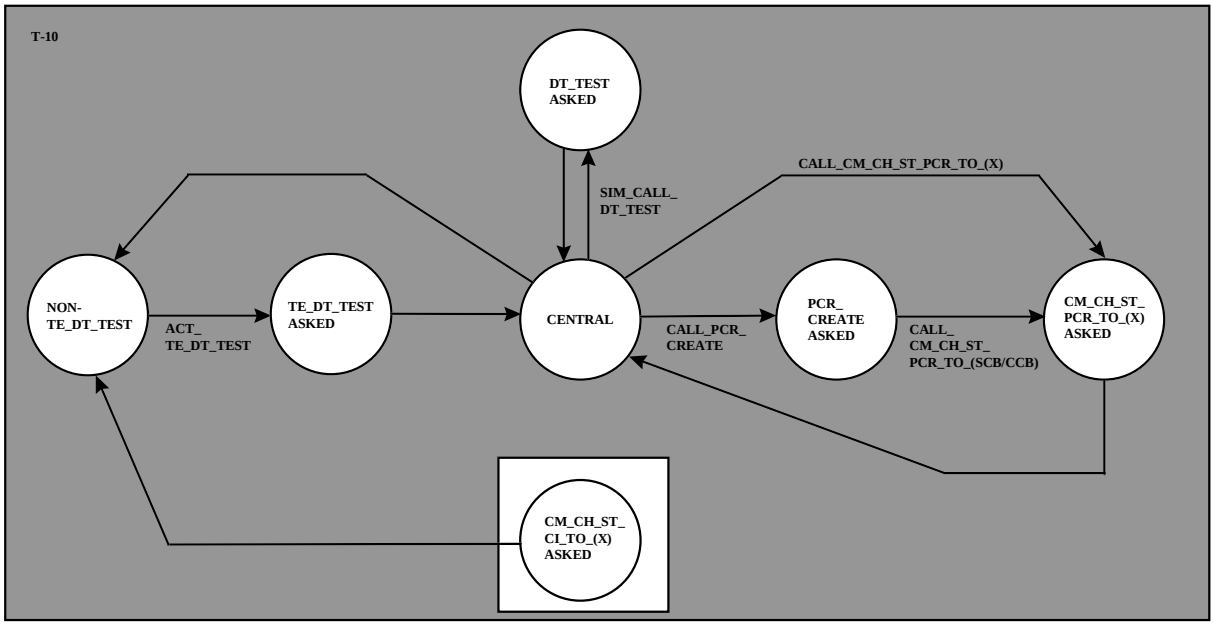


figure 4.42 employee int-te_dt_test : subprocess S10

The 5th employee is the caller operation ‘te_dt_test’. This employee has two subprocesses S9 and S10 and two traps T-9 and T-10. This caller does not have to wait after the call on some result from the callee (see the explanation in the external STD-subparagraph).

The sixth until tenth employees are the callee ‘cm_release’ and its 4 callers ‘tpm_modify’, ‘tpm_review’, ‘tpm_dt_test’ and ‘tpm_ac_test’. Which caller has executed the call is determined in the discriminator state ‘disc_starting_cm_release’ of the manager STD.

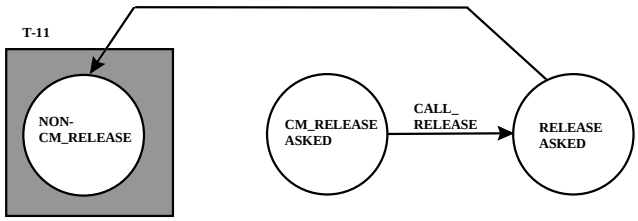


figure 4.43 employee int-cm_release : subprocess S11

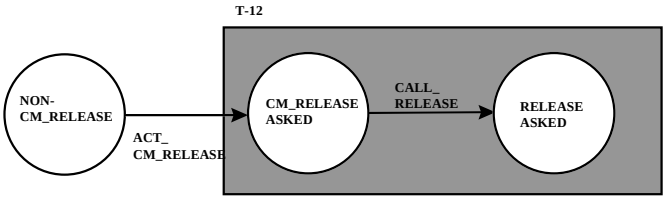


figure 4.44 employee int-cm_release : subprocess S12

The 6th employee is the own internal operation ‘cm_release’. This employee has two subprocesses S11 and S12 and two traps T-11 and T-12. The operation ‘cm_release’ is started after it has been determined which of its potential callers has performed the call.

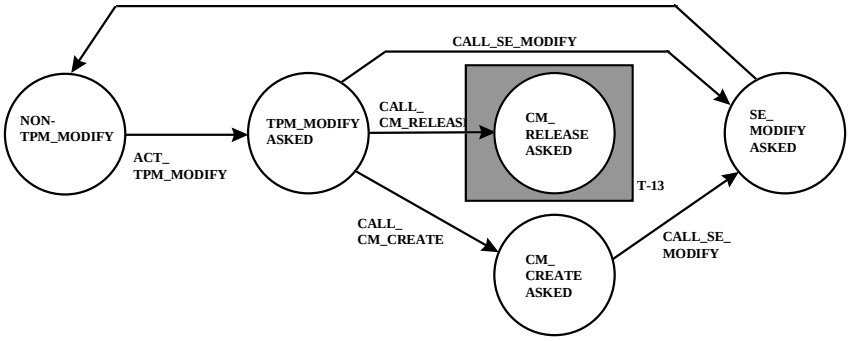


figure 4.45 employee int-tpm_modify : subprocess S13

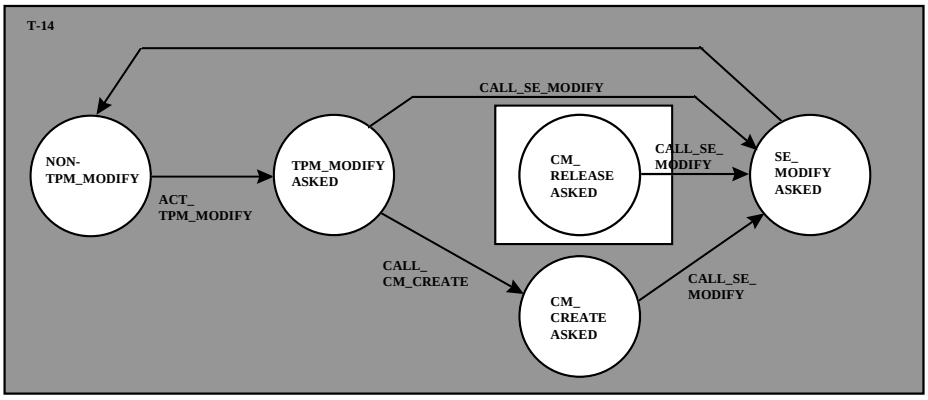


figure 4.46 employee int-tpm_modify : subprocess S14

The 7th employee is the caller operation ‘tpm_modify’. This employee has two subprocesses S13 and S14 and two traps T-13 and T-14. This caller does have to wait after the call on some result from the callee (see the explanation in the external STD-subparagraph). This is modeled by the ‘waiting_caller_proceed’-construct in the external/manager STD. This construct uses the same trap structure as the non-waiting caller_callee-construct.

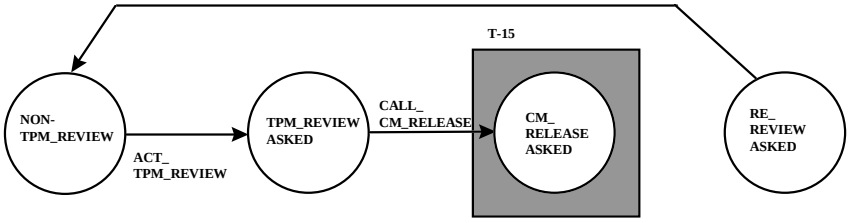


figure 4.47 employee int-tpm_review : subprocess S15

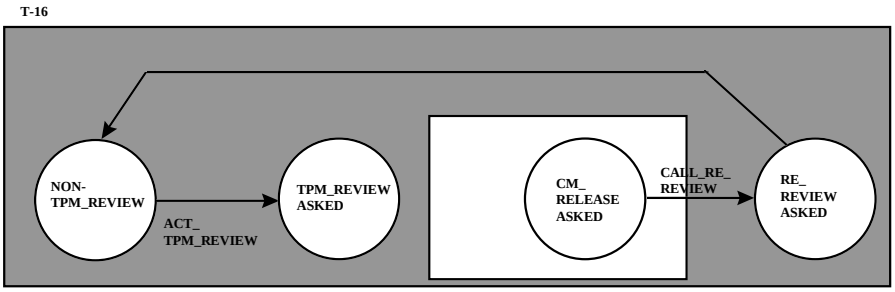


figure 4.48 employee int-tpm_review : subprocess S16

The 8th employee is the caller operation ‘tpm_modify’. This employee has two subprocesses S15 and S16 and two traps T-15 and T-16. This caller does have to wait after the call on some result from the callee (see the explanation in the external STD-subparagraph). This is modeled by the ‘waiting_caller_proceed’-construct in the external/manager STD. This construct uses the same trap structure as the non-waiting caller_callee-construct.

The 9th employee is the caller operation ‘tpm_dt_test’. This employee has two subprocesses S17 and S18 and two traps T-17 and T-18. This caller does have to wait after the call on some result from the callee (see the explanation in the external STD-subparagraph). This is modeled by the ‘waiting_caller_proceed’-construct in the external/manager STD. This construct uses the same trap structure as the non-waiting caller_callee-construct.

The 10th employee is the caller operation ‘tpm_ac_test’. This employee has two subprocesses S19 and S20 and two traps T-19 and T-20. This caller does have to wait after the call on some result from the callee (see the explanation in the external STD-subparagraph). This is modeled by the ‘waiting_caller_proceed’-construct in the external/manager STD. This construct uses the same trap structure as the non-waiting caller_callee-construct.

The eleventh until the 16th employees plus the 24th employee are the callee ‘cm_ch_st_pcr_to_(x)’ and its 6 callers ‘tpm_cluster_pcr’, ‘te_dt_test’, ‘cu_acceptance_test’, ‘scb_consider_pcr’, ‘ccb_consider_pcr’ and ‘cu_issue_pcr’. Which caller has executed the call is determined in the discriminator state ‘disc_starting_cm_ch_st_pcr_to_(x)’ of the manager STD.

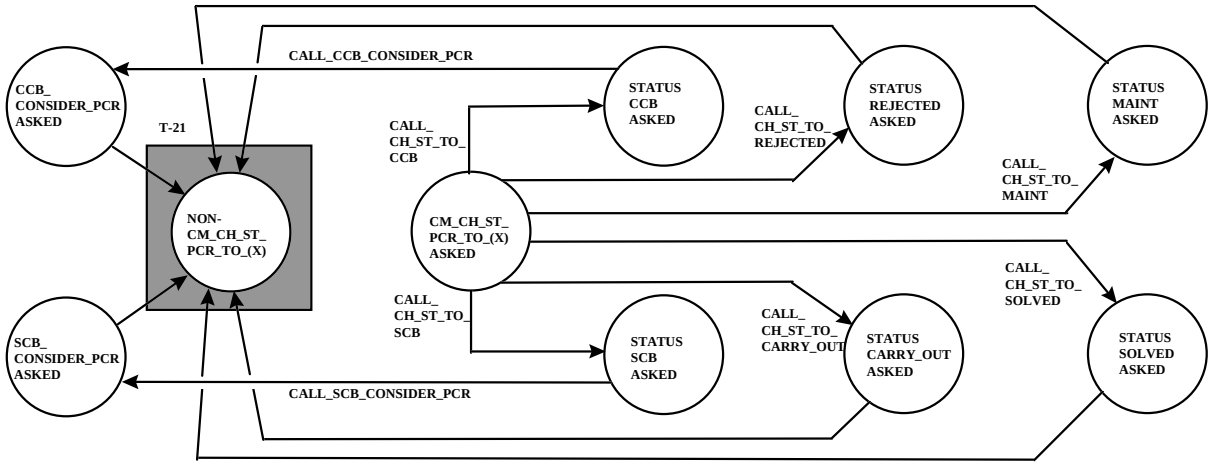


figure 4.49 employee int-cm_ch_st_pcr_to_(x) : subprocess S21

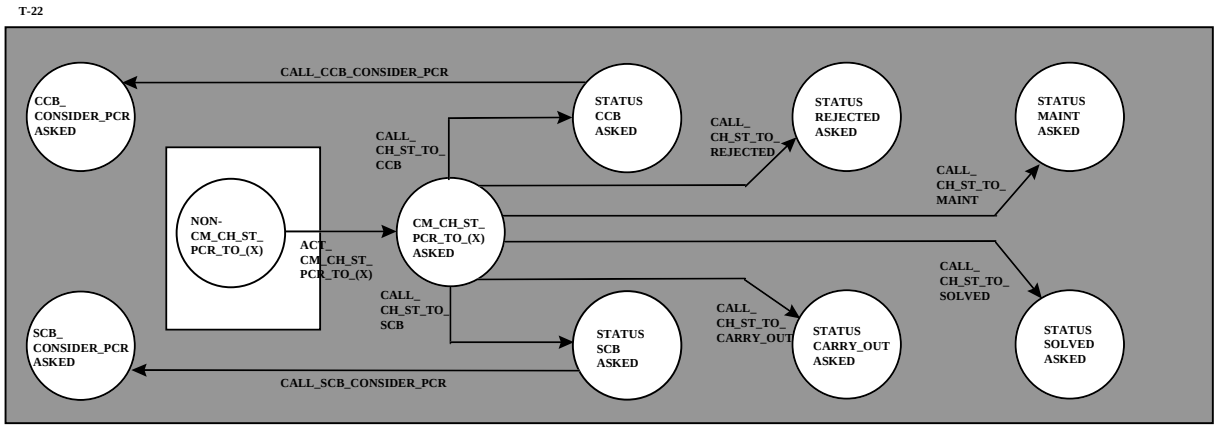


figure 4.50 employee int-cm_ch_st_pcr_to_(x) : subprocess S22

The eleventh employee is the own internal operation ‘cm_ch_st_pcr_to_(x)’. This employee has two subprocesses S21 and S22 and two traps T-21 and T-22. The operation ‘cm_ch_st_pcr_to_(x)’ is started after it has been determined which of its potential callers has executed the call.

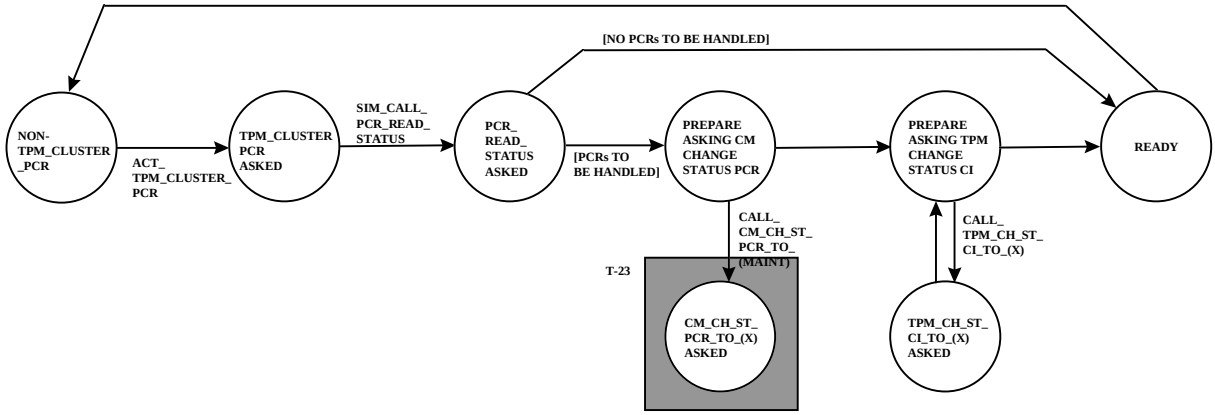


figure 4.51 employee int-tpm_cluster : subprocess S23

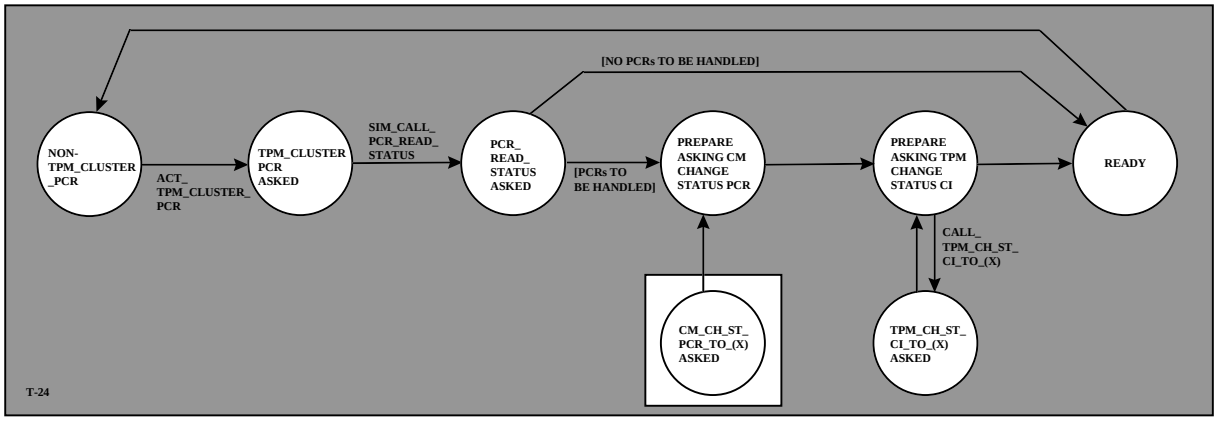


figure 4.52 employee int-tpm_cluster : subprocess S24

The 12th employee is the caller operation ‘tpm_cluster_pcr’. This employee has two subprocesses S23 and S24 and two traps T-23 and T-24. This caller does not have to wait after the call on some result from the callee (see the explanation in the external STD-subparagraph).

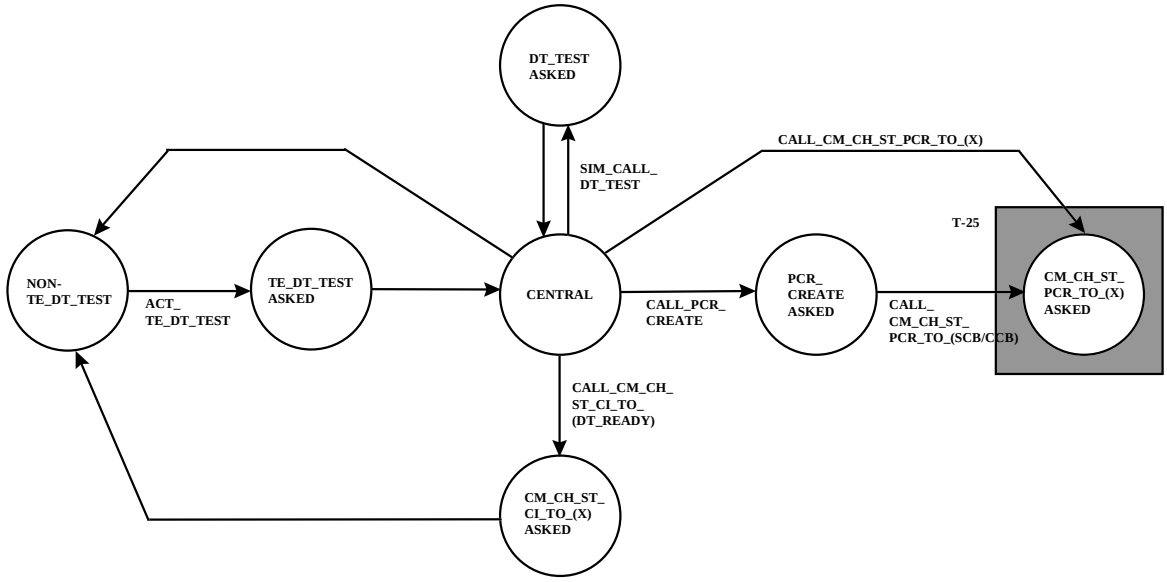


figure 4.53 employee int-te_dt_test : subprocess S25

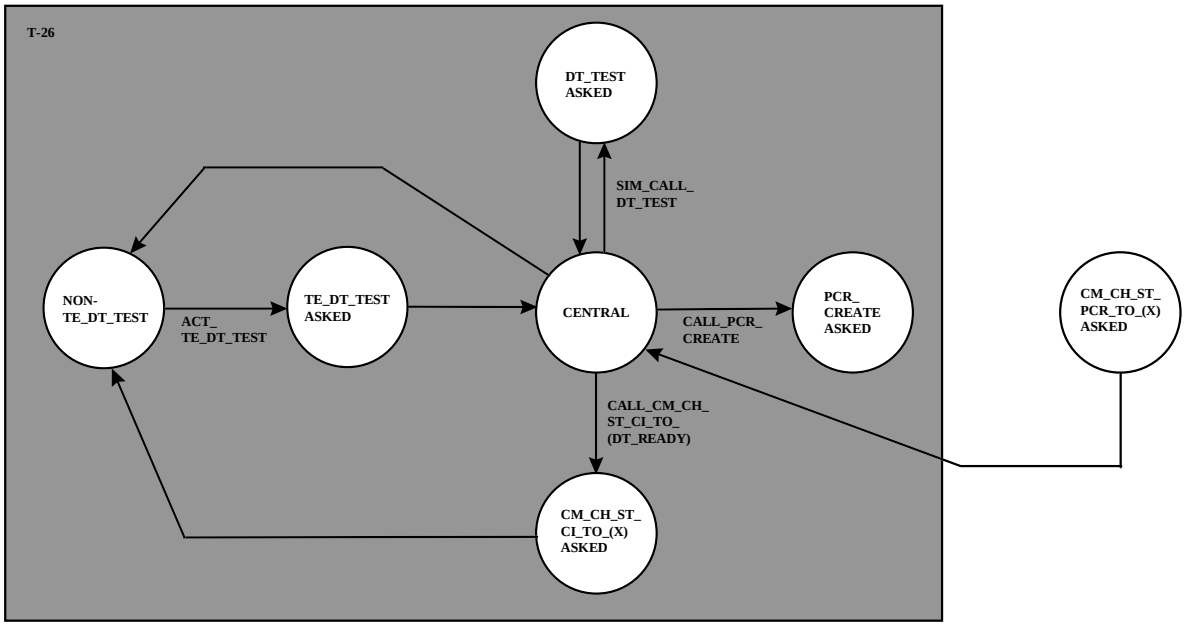


figure 4.54 employee int-te_dt_test : subprocess S26

The 13th employee is the caller operation ‘te_dt_test’. This employee has two subprocesses S25 and S26 and two traps T-25 and T-26. This caller does not have to wait after the call on some result from the callee (see the explanation in the external STD-subparagraph).

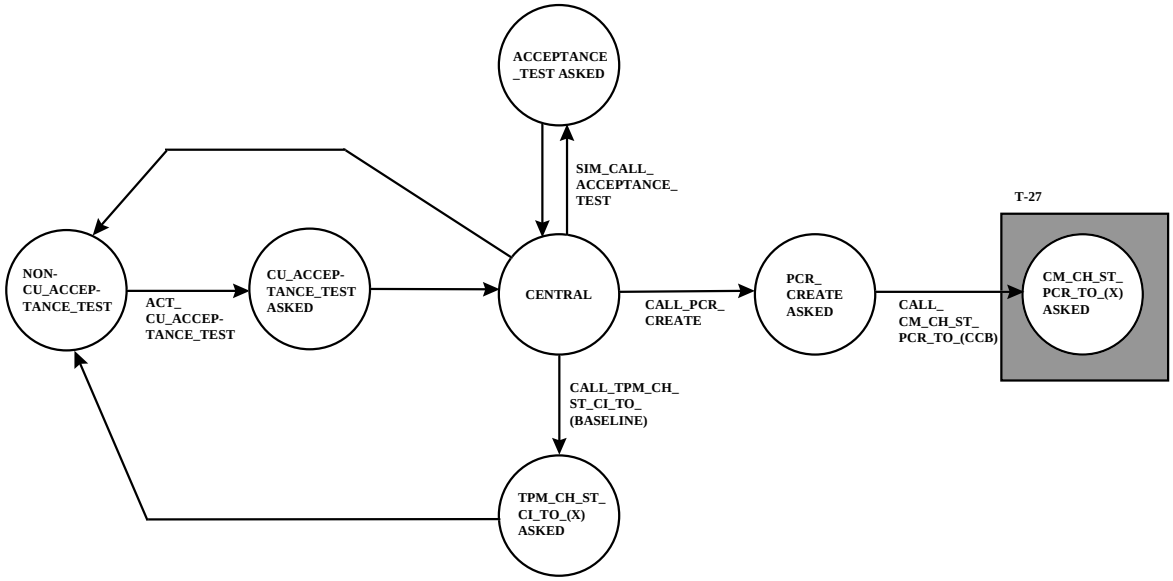


figure 4.55 employee int-cu_acceptance_test : subprocess S27

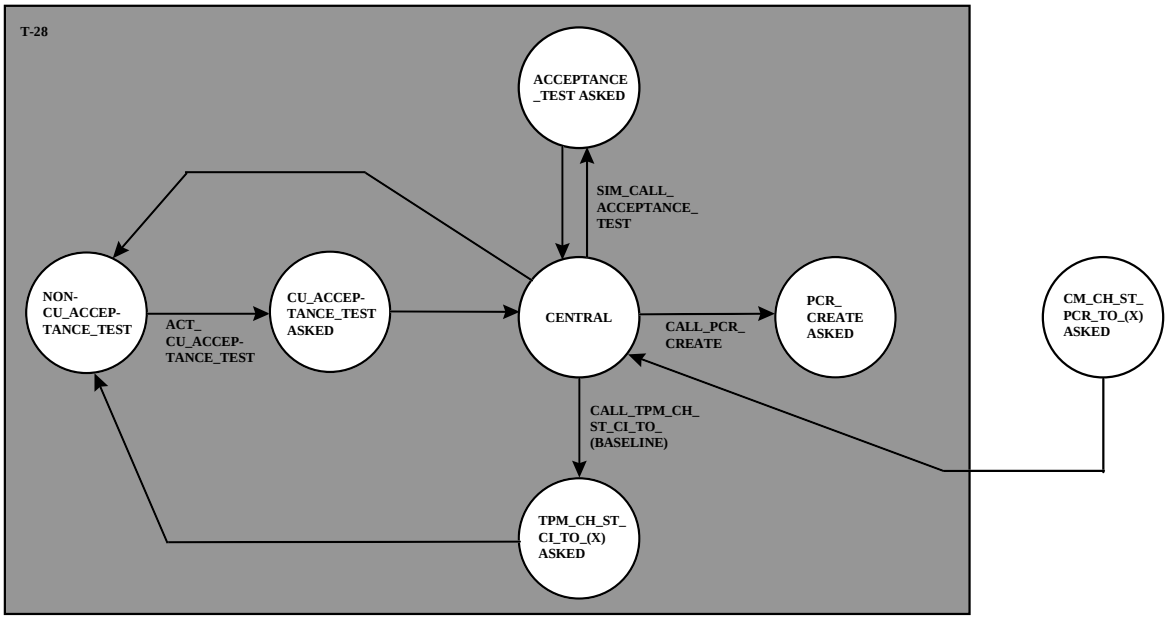


figure 4.56 employee int-cu_acceptance_test : subprocess S28

The 14th employee is the caller operation ‘cu_acceptance_test’. This employee has two subprocesses S27 and S28 and two traps T-27 and T-28. This caller does not have to wait after the call on some result from the callee (see the explanation in the external STD-subparagraph).

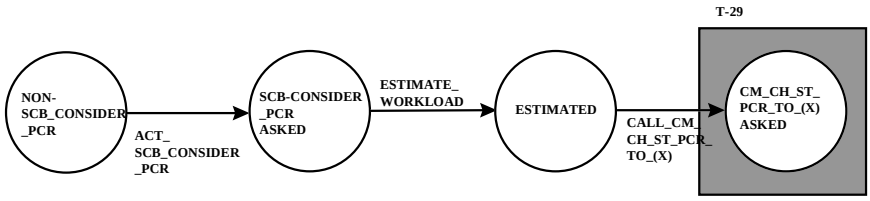


figure 4.57 employee int-scb_consider_pcr : subprocess S29

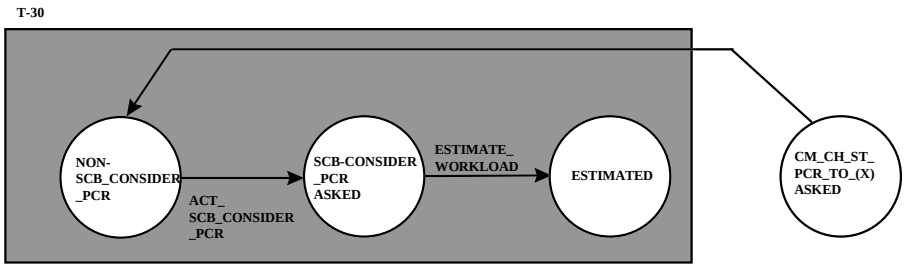


figure 4.58 employee int-scb_consider_pcr : subprocess S30

The 15th employee is the caller operation ‘scb_consider_pcr’. This employee has two subprocesses S29 and S30 and two traps T-29 and T-30. This caller does not have to wait after the call on some result from the callee (see the explanation in the external STD-subparagraph).

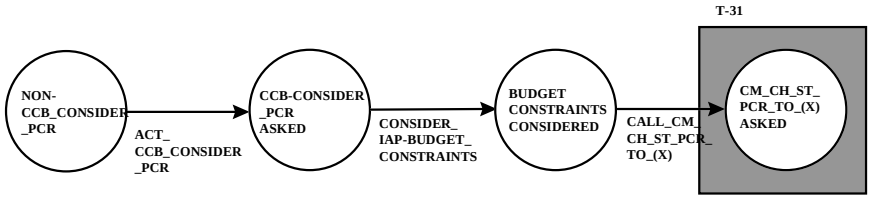


figure 4.59 employee int-ccb_consider_pcr : subprocess S31

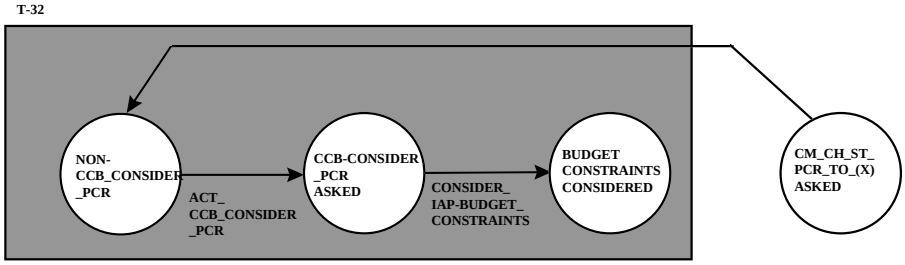


figure 4.60 employee int-ccb_consider_pcr : subprocess S32

The 16th employee is the caller operation ‘ccb_consider_pcr’. This employee has two subprocesses S31 and S32 and two traps T-31 and T-32. This caller does not have to wait after the call on some result from the callee (see the explanation in the external STD-subparagraph).

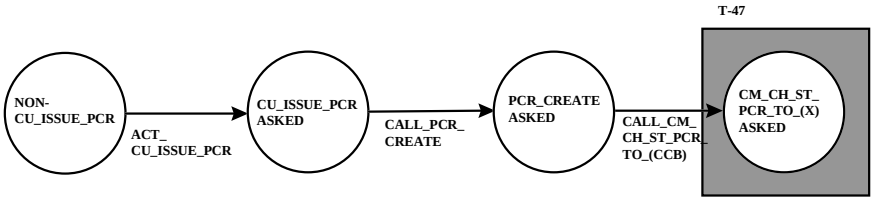


figure 4.61 employee int-cu_issue_pcr : subprocess S47

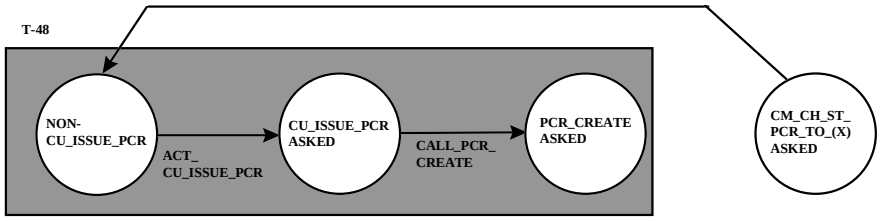


figure 4.62 employee int-cu_issue_pcr : subprocess S48

The 24th employee is the caller operation ‘cu_issue_pcr’. This employee has two subprocesses S47 and S48 and two traps T-47 and T-48. This caller does not have to wait after the call on some result from the callee (see the explanation in the external STD-subparagraph).

The 17th and 18th employees are the callee-caller pair ‘cm_update_version’ and ‘se_modify’.

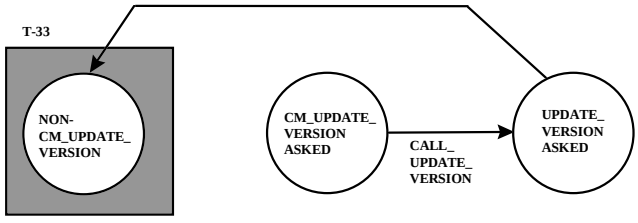


figure 4.63 employee int-cm_update_version : subprocess S33

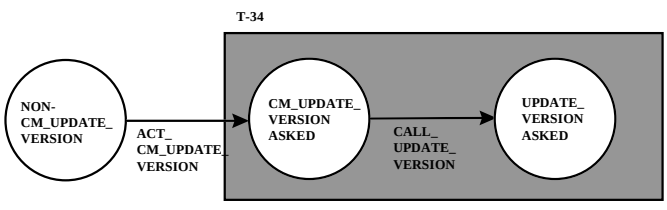


figure 4.64 employee int-cm_update_version : subprocess S34

The 17th employee is the own internal operation ‘cm_update_version’. This employee has two subprocesses S33 and S34 and two traps T-33 and T-34.

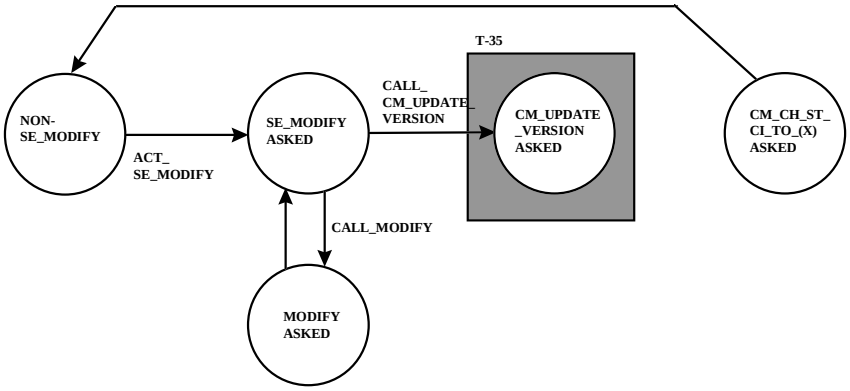


figure 4.65 employee int-ccb_consider_pcr : subprocess S35

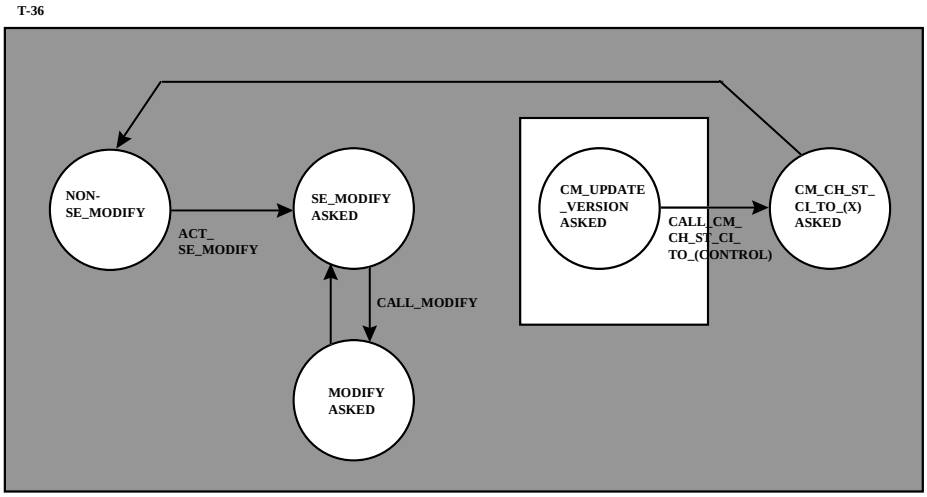


figure 4.66 employee int-ccb_consider_pcr : subprocess S36

The 18th employee is the caller operation ‘se_modify’. This employee has two subprocesses S35 and S36 and two traps T-35 and T-36. This caller does have to wait after the call on some result from the callee (see the explanation in the external STD-subparagraph). This is modeled by the ‘waiting_caller_proceed’-construct in the external/manager STD. This construct uses the same trap structure as the non-waiting caller_callee-construct.

The 19th and 20th employees are the callee-caller pair ‘cm_release_note’ and ‘tpm_ch_st_ci_to_(x)’

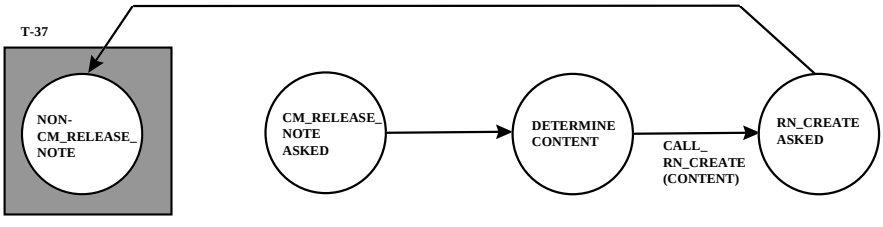


figure 4.67 employee int-cm_release_note : subprocess S37

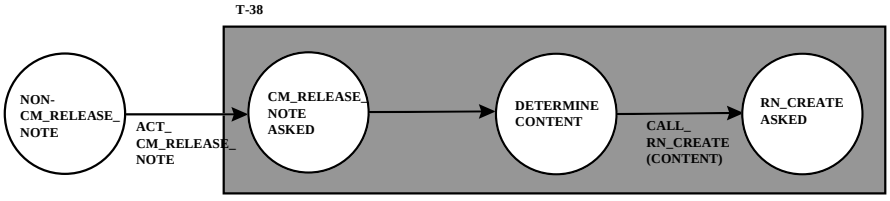


figure 4.68 employee int-cm_release_note : subprocess S38

The 19th employee is the own internal operation ‘cm_release_note’. This employee has two subprocesses S37 and S38 and two traps T-37 and T-38.

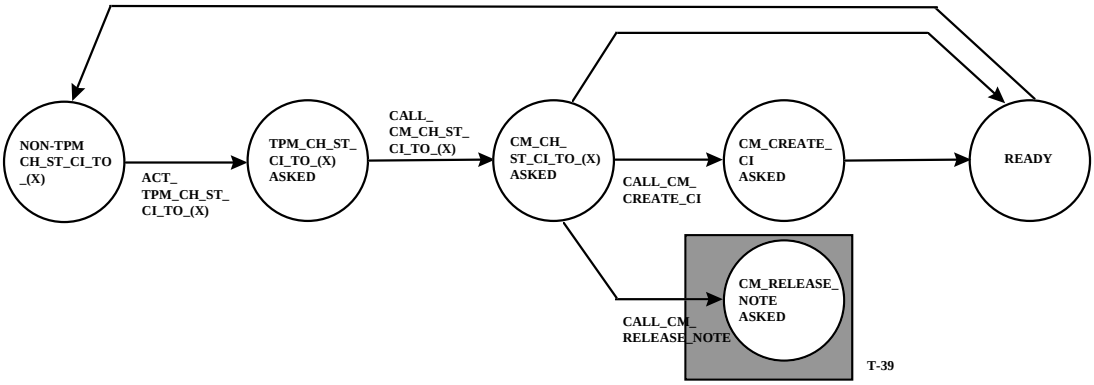


figure 4.69 employee int-tpm_ch_st_ci_to_(x) : subprocess S39

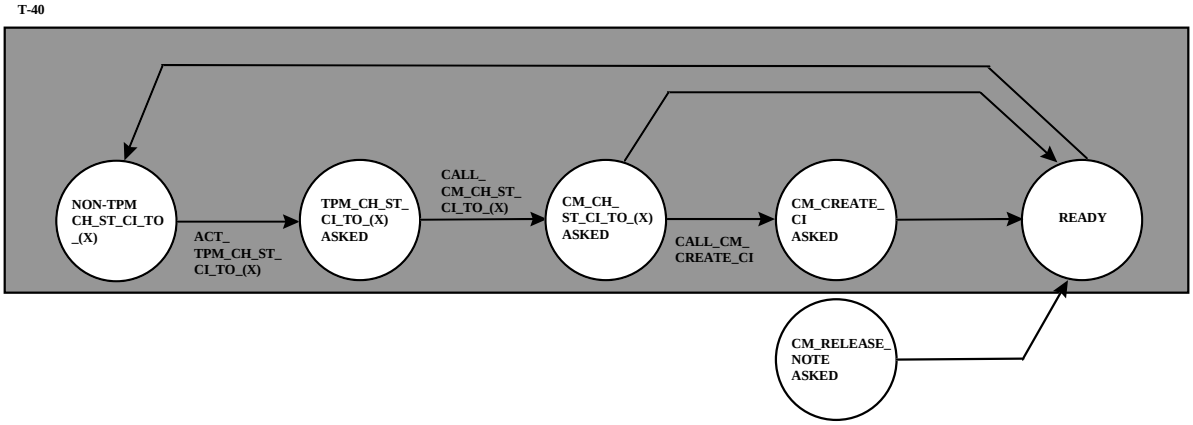


figure 4.70 employee int-tpm_ch_st_ci_to_(x) : subprocess S40

The 20th employee is the caller operation ‘tpm_ch_st_ci_to_(x)’. This employee has two subprocesses S39 and S40 and two traps T-39 and T-40. This caller does not have to wait after the call on some result from the callee (see the explanation in the external STD-subparagraph).

The 21st until the 23rd employees are the callee ‘cm_create_ci’ and its 2 callers ‘tpm_modify’ and ‘tpm_ch_st_ci _to_(x)’. Which caller has executed the call is determined in the discriminator state ‘disc_starting_cm_create_ci’ of the manager STD.

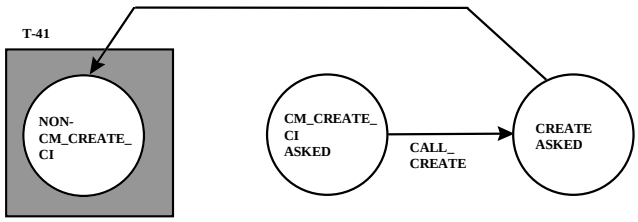


figure 4.71 employee int-cm_create_ci : subprocess S41

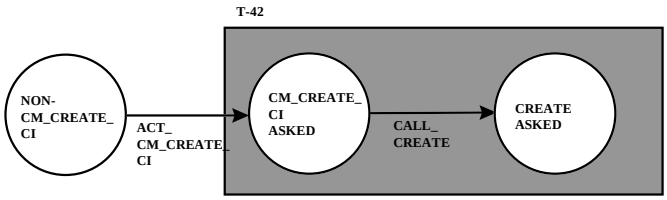


figure 4.72 employee int-cm_create_ci : subprocess S42

The 21st employee is the own internal operation ‘cm_create_ci’. This employee has two subprocesses S41 and S42 and two traps T-41 and T-42.

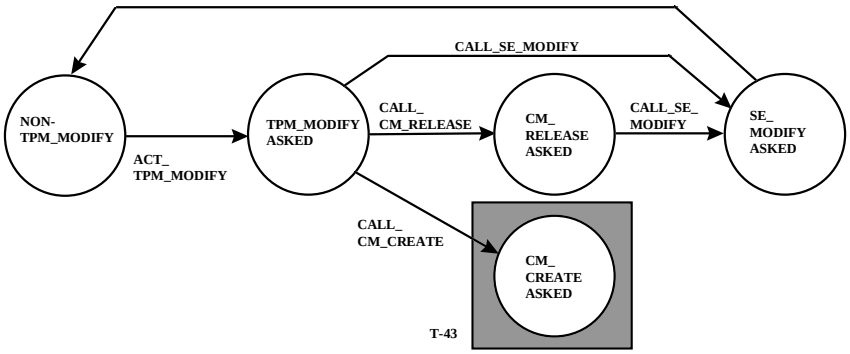


figure 4.73 employee int-tpm_modify : subprocess S43

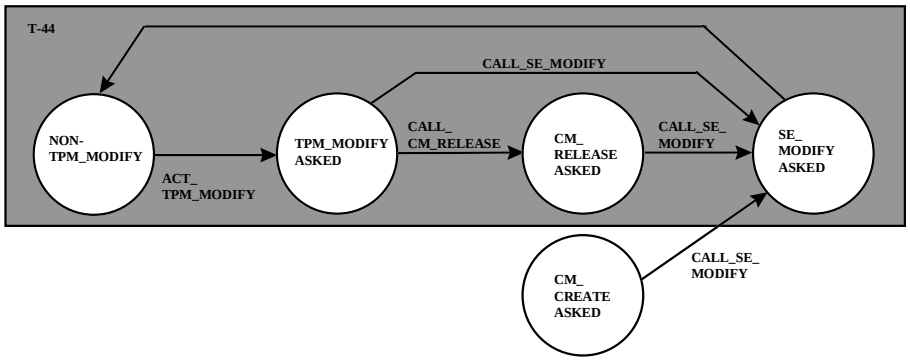


figure 4.74 employee int-tpm_modify : subprocess S44

The 22nd employee is the caller operation ‘tpm_modify’. This employee has two subprocesses S43 and S44 and two traps T-43 and T-44. This caller does have to wait after the call on some result from the callee (see the explanation in the external STD-subparagraph). This is modeled by the ‘waiting_caller_proceed’-construct in the external/manager STD. This construct uses the same trap structure as the non-waiting caller_callee-construct.

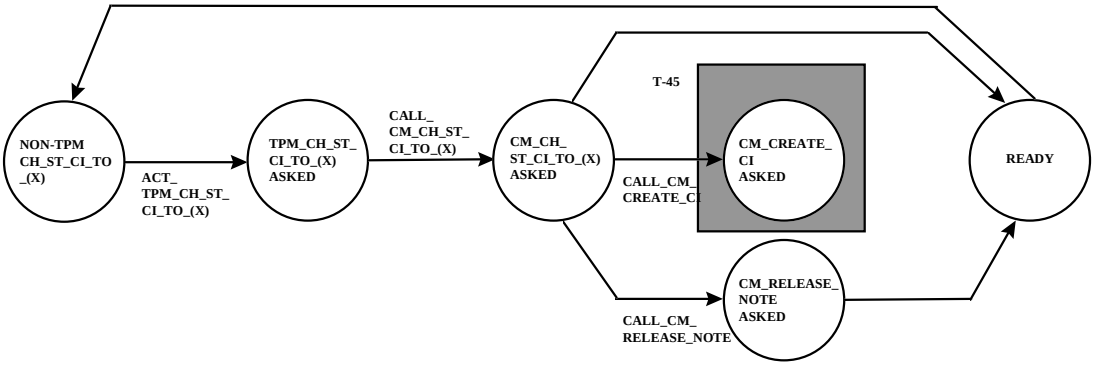


figure 4.75 employee int-tpm_ch_st_ci_to_(x) : subprocess S45

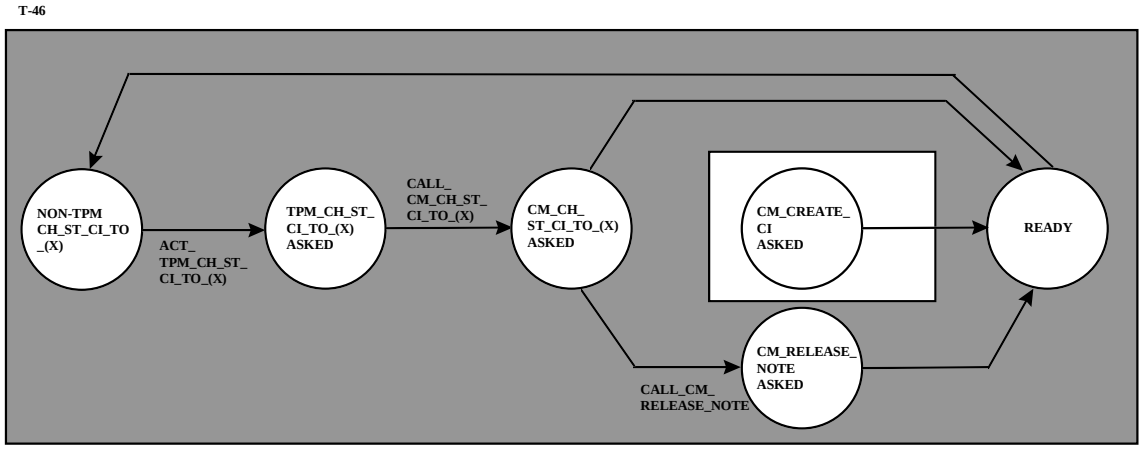


figure 4.76 employee int-tpm_ch_st_ci_to_(x) : subprocess S46

The 23rd employee is the caller operation ‘tpm_ch_st_ci_to_(x)’. This employee has two subprocesses S45 and S46 and two traps T-45 and T-46. This caller does have to wait after the call on some result from the callee (see the explanation in the external STD-subparagraph). This is modeled by the ‘waiting_caller_proceed’-construct in the external/manager STD. This construct uses the same trap structure as the non-waiting caller_callee-construct.

The 24th employee is the operation ‘cu_issue_pcr’. This operation calls the operation ‘cm_ch_st_pcr_to_(x)’. Its subprocesses and traps are described earlier in this subparagraph together with the descriptions of the other callers (11th until 16th employee) of ‘cm_ch_st_pcr_to_(x)’.

4.3.2.3 Configuration_Item

4.3.2.3.1 Configuration_Item : external behavior-STD, organizational view

The external behavior STD shows the possible sequences in which the exported operation(s) can be started. The organizational view of the external STD does not show any communication details.

An organizational view of an external STD is given when there is a need to depict the core activity of a class more clearly. The core activity of the class ‘configuration_item’ is the administration of the status of a configuration_item (CI). To get a clear picture of the possible status changes of a CI an organizational view of external STD of the configuration_item is given. Only the transitions and states relevant for the status(changes) of the CI are shown.

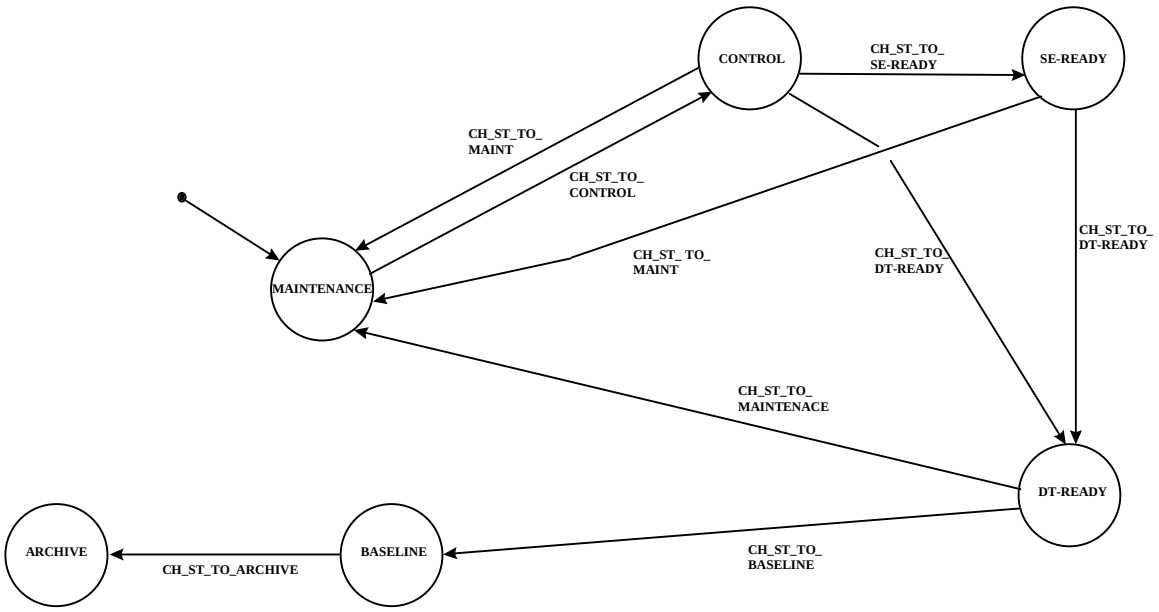


figure 4.77 configuration_item : external behavior STD, organizational view

This external STD, organizational view, can be constructed from the external STD, communicative view. Two construction methods are possible. These are the ‘homomorphic picture’-construction and the ‘aggregate state’-construction. Both construction methods are demonstrated below.

Homomorphic picture-construction of a (organizational) view

The external STD, communicative view is a homomorphic picture of the external STD, communicative view. (For an explanation of homomorphism, see [EBE].) To construct the organizational view from the communicative view, a homomorphism ‘h’ (a mapping function ‘h’) is needed. This mapping function ‘h’ maps the states from the communicative view STD onto the states of the organizational view STD in such a way that the resulting organizational view STD only contains those states and transitions that are relevant for the status(changes) of the CI.

The external STD, communicative view, as given in the next paragraph is used here in the construction of the external STD, organizational view.

The mapping function ‘h’ is given in the next figure. This figure shows in the upper righthand corner the external STD, communicative view. In the lower lefthand corner the figure shows the external STD, organizational view. The mapping function ‘h’ from the states of the communicative view STD to the states of the organizational view STD is shown by the ‘dashed’ lines.

The mapping function ‘h’ is then given explicitly by the table that follows the figure.

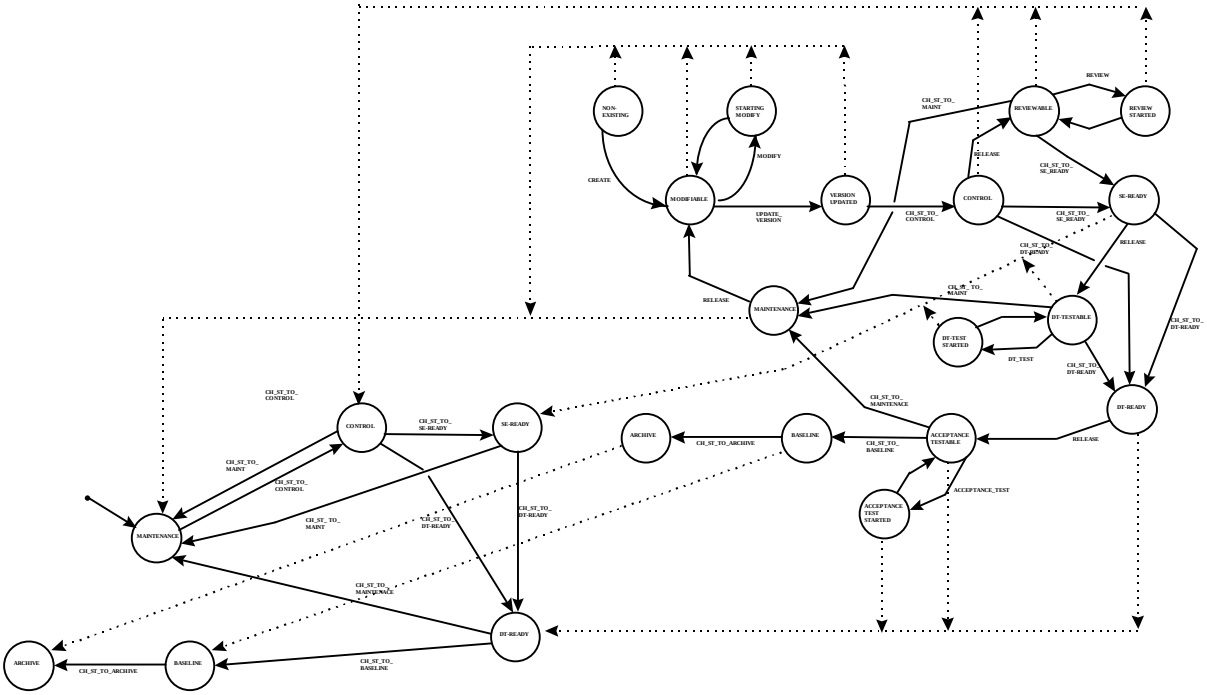


figure 4.78 mapping function 'h' (dashed lines)

from state (external STD, communicative view)	**to state** (external STD, organizational view)
maintenance	maintenance
non-existing	maintenance
modifiable	maintenance
starting modify	maintenance
version update	maintenance
control	control
reviewable	control
review started	control
se_ready	se_ready
dt_testable	se_ready
dt_test started	se_ready
dt_ready	dt_ready
acceptance testable	dt-ready
acceptance test started	dt_ready
baseline	baseline
archive	archive

table 4.1 mapping function 'h'

Aggregate state-construction of a (organizational) view

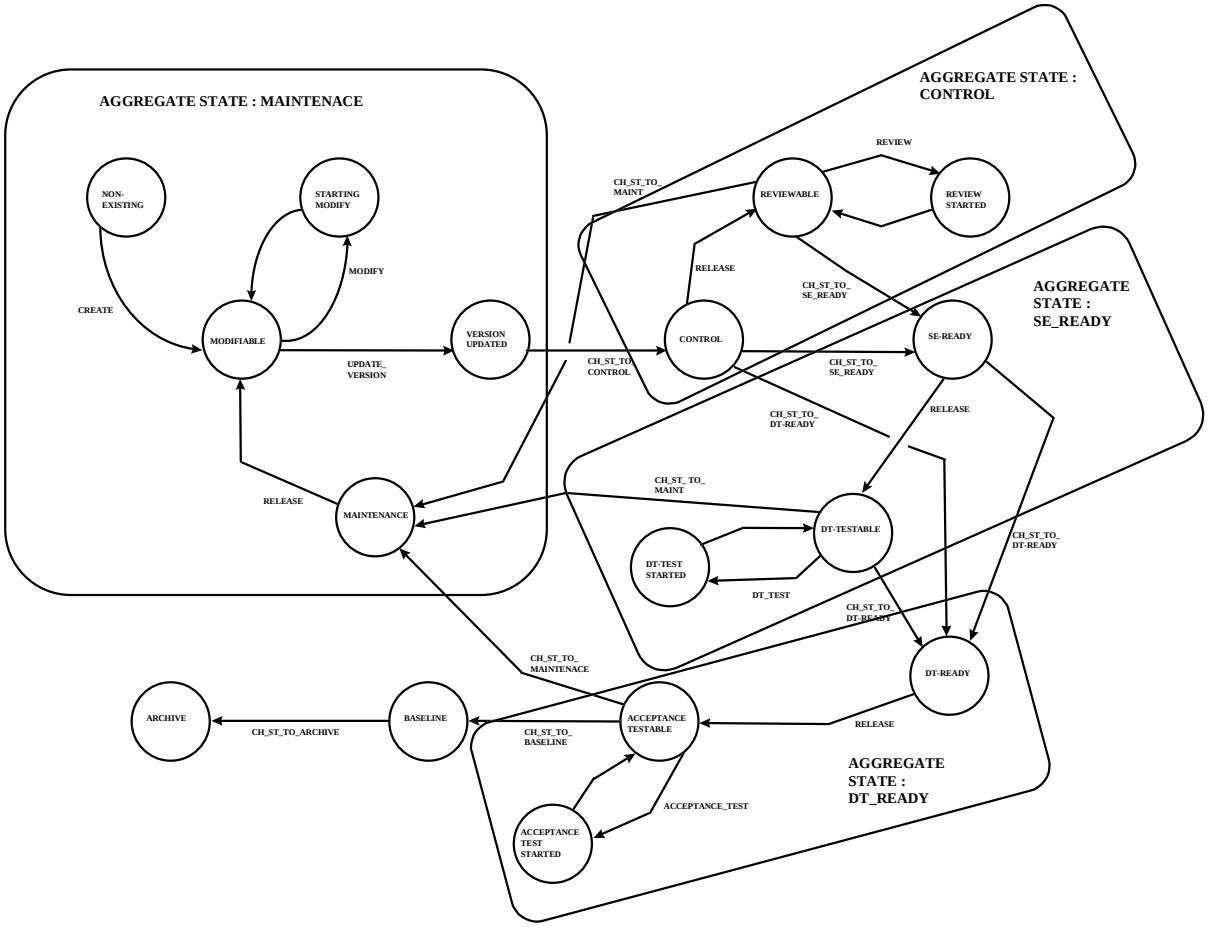


figure 4.79 external STD, communicative view, with aggregate states

Again the external STD, communicative view, as given in the next paragraph is used here in the construction of the external STD, organizational view. The aggregate state-construction consists of taking together certain states to form ‘aggregate’ states. These aggregate states constitute a certain (higher level) view of a STD. The construction is performed in the figure above. The aggregate states are shown as ‘rounded rectangulars’ that are imposed upon the original external STD, communicative view.

The states ‘maintenance’, ‘non-existing’, ‘modifiable’, ‘starting modify’ and ‘version update’ are grouped together in the aggregate state ‘maintenace’.

The states ‘control’, ‘reviewable’ and ‘review started’ are grouped together in the aggregate state ‘control’.

The states ‘se_ready’, ‘dt_testable’ and ‘dt_test started’ are grouped together in the aggregate state ‘se_ready’.

The states ‘dt_ready’, ‘acceptance testable’ and ‘acceptance test started’ are grouped together in the aggregate state ‘dt_ready’.

The states ‘baseline’ and ‘archive’ stay single as ‘baseline’ and ‘archive’.

The next step in the construction is to no longer show the states inside an aggregate state. When this step is performed, the external STD, organizational view, is found which only contains those states and transitions that are relevant for the status(changes) of the CI.

4.3.2.3.2 Configuration_Item : external behavior-STD, communicative view

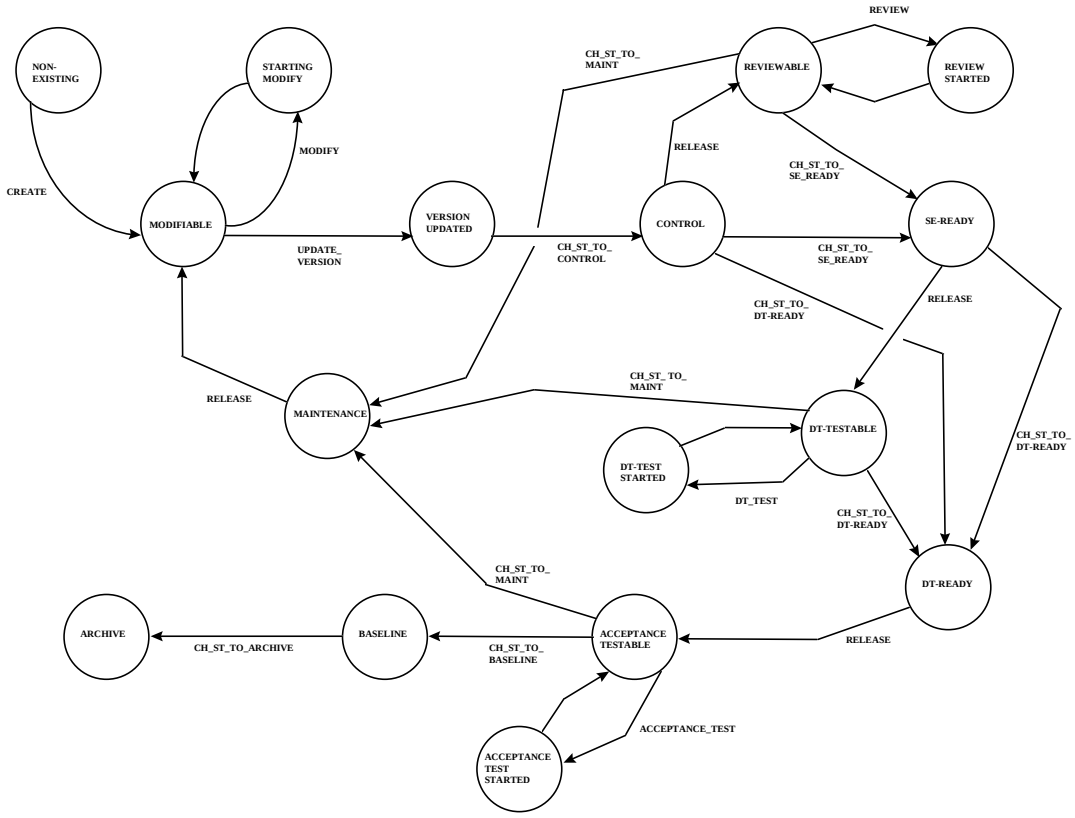


figure 4.80 configuration_item : external behavior STD, communicative view

This external behavior STD shows the states an object of the class ‘configuration_item’ (CI) can be in during its life cycle. It also shows when the transitions from one state to another are made as a response to calls being made to its operations.

The object starts by being created. It can then be modified. After being modified, its version is updated and it is brought under configuration management (CM) with the status ‘control’. Then it can be released from configuration management for review. Or the technical_project_manager can decide to bypass the review, in which case the status of the CI becomes ‘se_ready’ (software_engineer_ready) . Or the technical_project_manager can decide to bypass both the review and the dt-test. The status of the CI then becomes ‘dt_ready’ (development team_ready) .

If the result of the review is positive, the status of the CI becomes ‘se_ready’. If the result of the review is negative the status of the CI becomes ‘maintenance’.

From the status ‘se_ready’ the CI can be released from configuration management for the dt-test. Or the technical_project_manager can decide to bypass the dt-test. This results in the CI status ‘dt_ready’. Also if the result of the dt-test is positive, the CI status becomes ‘dt_ready’. If the result of the dt-test is negative, the CI gets the status ‘maintenance’.

From the status ‘dt_ready’ the CI can be released from configuration management for the acceptance-test by the customer. If it is accepted by the customer the CI gets the status ‘baseline’ else ‘maintenance’.

A CI of a system that is in operational use has the status ‘baseline’. If a problem_and_change_report (PCR) is issued on this CI the following will happen. The CI gets the status ‘archive’. A new CI is created. The new CI gets, when it is created, the same contents as the old CI. The software_engineer solves the problem in the new CI. The old CI is in fact ‘archived’.

4.3.2.3.3 Configuration_Item : internal behavior-STDs

The configuration_item has 13 operations : ‘create’, ‘update_version’, ‘release’, ‘modify’, ‘review’, ‘dt_test’, ‘acceptance_test’, ‘ch_st_to_control’, ‘ch_st_to_se_ready’, ‘ch_st_to_dt_ready’, ‘ch_st_to_baseline’, ‘ch_st_to_maint’ and ‘ch_st_to_archive’. These operations have the following internal behavior STDs.

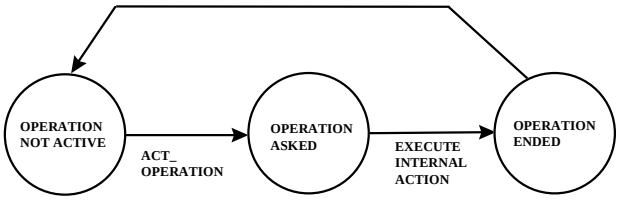


figure 4.81 only_internal_action_construct : internal behavior STD

All operations can be modeled by the ‘only_internal_action’-construct since they don’t execute any calls to other objects. They all follow the construct’s template and their actions are explained in the following table.

OPERATION	INTERNAL ACTION
CH_ST_TO_CONTROL	UPDATE STATUS ATTRIBUTE TO CONTROL AND RELEASE ATTRIBUTE TO NO
CH_ST_TO_SE_READY	UPDATE STATUS ATTRIBUTE TO SE_READY AND RELEASE ATTRIBUTE TO NO
CH_ST_TO_DT_READY	UPDATE STATUS ATTRIBUTE TO DT_READY AND RELEASE ATTRIBUTE TO NO
CH_ST_TO_BASELINE	UPDATE STATUS ATTRIBUTE TO BASELINE AND RELEASE ATTRIBUTE TO NO
CH_ST_TO_MAINT	UPDATE STATUS ATTRIBUTE TO MAINT AND RELEASE ATTRIBUTE TO NO
CH_ST_TO_ARCHIVE	UPDATE STATUS ATTRIBUTE TO ARCHIVE
UPDATE_VERSION	INCREMENT VERSION ATTRIBUTE
RELEASE	UPDATE RELEASE ATTRIBUTE TO YES
CREATE	CREATE NEW OBJECT AND INITIALIZE IT. THE INITIALIZATION CAN BE DONE WITH THE CONTENT OF ANOTHER EXISTING OBJECT
MODIFY	MODIFY (READ/WRITE/DELETE) THE CONTENT ATTRIBUTE
REVIEW	READ THE CONTENT ATTRIBUTE + UPDATE RESULT
DT_TEST	EXECUTE (PART OF) THE CONTENT ATTRIBUTE + UPDATE RESULT
ACCEPTANCE_TEST	EXECUTE (PART OF) THE CONTENT ATTRIBUTE + UPDATE RESULT

table 4.2 only_internal_action_construct : operations and actions

The ‘create’ operation has in fact not only an internal action but also interaction with another object when it has to read its contents to initialize the newly created object. But because this interaction is not further detailed here, the ‘create’ operation can be described by the ‘only_iternal_action’-construct.

The operations ‘dt_test’ and ‘acceptance_test’ execute (part of) the content attribute of the ci. These are ‘virtual’ operations in the sense that they model the property of the CI to be able to be executed after it has been compiled and linked into an executable. When a CI is tested individually, the CI runs in a test-harness. Every testcase executes part of this CI. In case of a system-level test the CI is linked with other CI’s into one executable which is tested by performing testcases on one or more CI’s.

The operations ‘ch_st_to_control’, ‘ch_st_to_se_ready’, ‘ch_st_to_dt_ready’, ‘ch_st_to_maint’, ‘ch_st_to_baseline’, ’ch_st_to_archive’, ‘release’ and ‘review’ have no formal parameter. The operation ‘create’ has two parameters. The first indicating if it has to initialize the newly created CI with the contents of an existing object and the second parameter gives the id of that object. The parameter of ‘update_version’ is the new version number. The operation ‘modify’ has as its parameter the updated (part of the) content of the CI.

4.3.2.3.4 Configuration_Item : manager-STD

The communication between the configuration_item’s operations (callees) and their callers is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

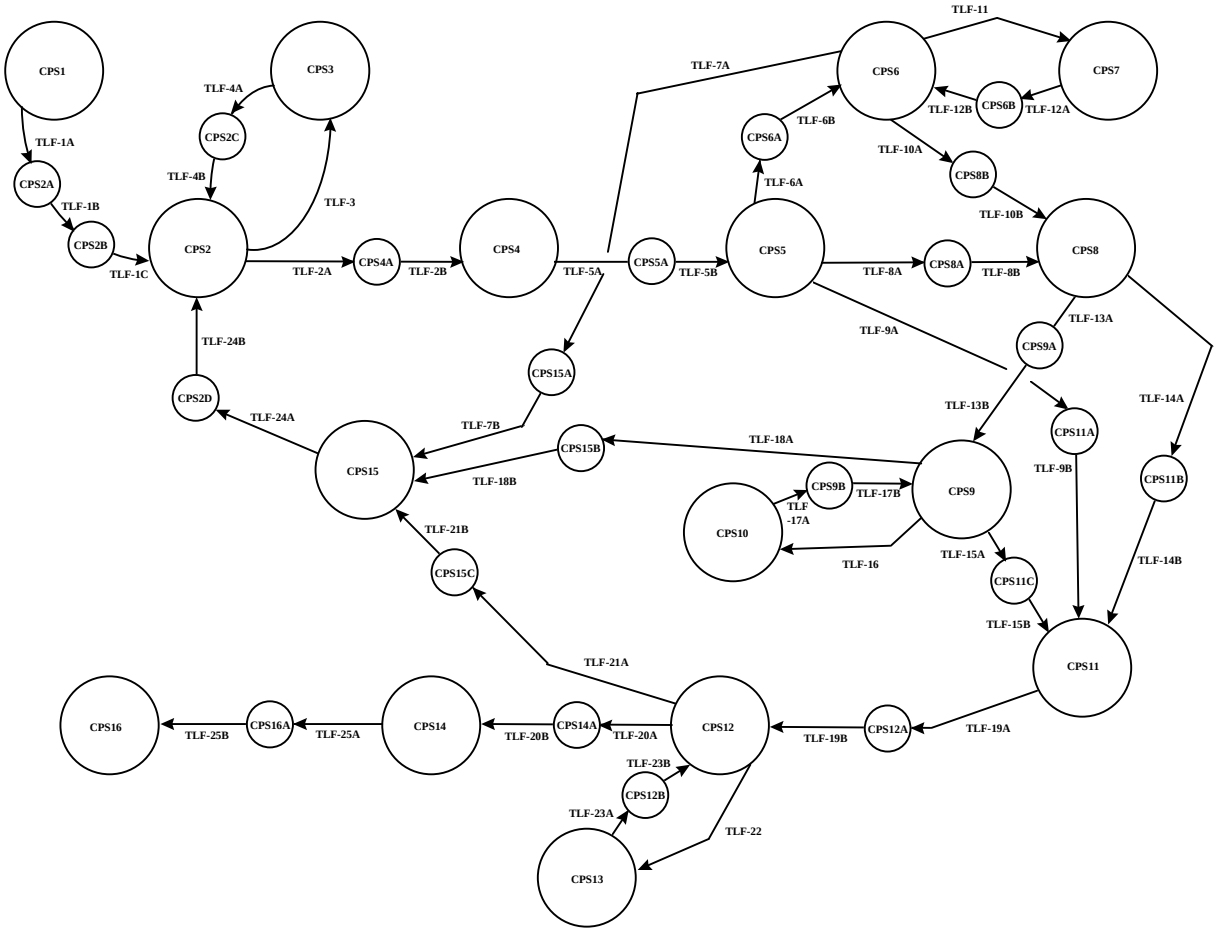


figure 4.82 configuration_item : manager STD

Every caller of every operation of the class configuration_item has to wait after the call, for the callee to reach some point in its execution. The status has to be updated before the caller may proceed. The configuration_item has to be created before the caller may proceed. The version of the CI has to be updated before the caller may proceed. The CI has to be released before the caller may proceed. The operations ‘modify’, ‘review’, ‘dt_test’ and ‘acceptance_test’ must perform their action and give some feedback to the caller. Only after the caller receives this feedback may he proceed.

To model this, every state of the external STD is given in the manager STD in an ‘exploded view’. Every state consists of several substates. In the above figure the exploded states are shown as a combination of a main substate and several, smaller ‘caller_waits’-substates. In these ‘caller_waits’-substates the callee is started and the caller must wait. Every callee is modeled with the ‘act-construct, small trap’. After the callee has reached some point in its execution (i.e enters its small trap) the transition out of the ‘caller_waits-substate is taken. In the next state the caller is then allowed to proceed. Also the second trap of the callee has to be small, because it has to finish performing its function before the manager STD may start the next operation (be it the same operation or another one). E.g. the status change to ‘control’ must be applied to the CI, before the operation ‘ch_st_to_se_ready’ may start its execution. If not, the operation ‘ch_st_to_se_ready’ could update the status attribute before the operation ‘ch_st_to_control’ does so, and an incorrect status would be the result (the ‘lost update’ problem [ELM]).

The notation CPSx in the STD stands for Consolidated Prescribed Subprocesses and is a set of CCx’s. The notation CCx in the STD stands for ‘Caller(s)-Callee’ and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for ‘Traps Logical Formula’ and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the numbering used in the next paragraph.

The CPS’s and the TLF’s for this manager are :

CPS1 = {CC1-1, CC2-1, CC3-1, CC4-3, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-1A = (T-19 and T-21)

CPS2A = {CC1-1, CC2-2, CC3-1, CC4-3, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-1B = T-20

CPS2B = {CC1-1, CC2-3, CC3-1, CC4-3, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-1B = T-22

CPS2C = {CC1-1, CC2-1, CC3-1, CC4-3, CC5-3, CC6-1, CC7-1, CC8-1}
TLF-4B = T-34

CPS2D = {CC1-1, CC2-1, CC3-1, CC4-2, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-24B= T-28

CPS2 = {CC1-1, CC2-1, CC3-1, CC4-3, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-2A = T-30 and (T-23 and T-25)
TLF-3 = T-30 and (T-31 and T-33)

CPS3 = {CC1-1, CC2-1, CC3-1, CC4-3, CC5-2, CC6-1, CC7-1, CC8-1}
TLF-4A = T-32

CPS4A = {CC1-1, CC2-1, CC3-2, CC4-1, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-2B = T-24

CPS4 = {CC1-1, CC2-1, CC3-3, CC4-1, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-5A = T-26 and (T-1 and T-13)

CPS5A = {CC1-2, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-5B = T-2

CPS5 = {CC1-3, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-6A = (T-27 and T-29)
TLF-8A = (T-3 and T-14a)
TLF-9A = (T-5 and T-14b)

CPS6A = {CC1-3, CC2-1, CC3-1, CC4-2, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-6B = T-28

CPS6B = {CC1-3, CC2-1, CC3-1, CC4-3, CC5-1, CC6-3, CC7-1, CC8-1}
TLF-12B= T-38

CPS6 = {CC1-3, CC2-1, CC3-1, CC4-3, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-7A = T-30 and (T-7 and T14c)
TLF-10A= T-30 and (T-3 and T14a)
TLF-11 = T-30 and (T-35 and T-37)

CPS7 = {CC1-3, CC2-1, CC3-1, CC4-3, CC5-1, CC6-2, CC7-1, CC8-1}
TLF-12A= T-36

CPS8A = {CC1-4, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-8B = T-4

CPS8B = {CC1-4, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-10B= T-4

CPS8 = {CC1-5, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-13A= (T-27 and T-29)
TLF-14A= (T-5 and T-15a)

CPS9A = {CC1-5, CC2-1, CC3-1, CC4-2, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-13A= T-28

CPS9B = {CC1-5, CC2-1, CC3-1, CC4-3, CC5-1, CC6-1, CC7-3, CC8-1}

TLF-17B= T-42

CPS9 = {CC1-5, CC2-1, CC3-1, CC4-3, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-15A= T-30 and (T-5 and T-15a)
TLF-16 = T-30 and (T-39 and T-41)
TLF-18A= T-30 and (T-7 and T-15b)

CPS10 = {CC1-5, CC2-1, CC3-1, CC4-3, CC5-1, CC6-1, CC7-2, CC8-1}
TLF-17A= T-40

CPS11A = {CC1-6, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-9B = T-6

CPS11B = {CC1-8, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-14B= T-6

CPS11C = {CC1-8, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-15B= T-6

CPS11 = {CC1-7, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-19A= (T-27 and T-29)

CPS12A = {CC1-7, CC2-1, CC3-1, CC4-2, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-19B= T-28

CPS12B = {CC1-7, CC2-1, CC3-1, CC4-3, CC5-1, CC6-1, CC7-1, CC8-3}
TLF-23B= T-46

CPS12 = {CC1-7, CC2-1, CC3-1, CC4-3, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-20A= T-30 and (T-9 and T-16a)
TLF-21A= T-30 and (T-7 and T-16b)
TLF-22 = T-30 and (T-43 and T-45)

CPS13 = {CC1-7, CC2-1, CC3-1, CC4-3, CC5-1, CC6-1, CC7-1, CC8-2}
TLF-23A= T-44

CPS14A = {CC1-9, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-20B= T-10

CPS14 = {CC1-10, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-25A= (T-11 and T-17)

CPS15A = {CC1-13, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-7B = T-8

CPS15B = {CC1-14, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-18A= T-8

CPS15C = {CC1-15, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-21A= T-8

CPS15 = {CC1-1, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-24A= (T-27 and T-29)

CPS16A = {CC1-11, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1, CC7-1, CC8-1}
TLF-25B= T-12

CPS16 = {CC1-12, CC2-1, CC3-1, CC4-1, CC5-1, CC6-1, CC7-1, CC8-1}

The caller-callee combinations for ‘ch_st_to_x’ and ‘cm_ch_st_ci_to_(x)’ are :

CC1-1 = {S1, S3, S5, S7, S9 , S11, S13}
CC1-2 = {S2, S3, S5, S7, S9 , S11, S13}
CC1-3 = {S1, S3, S5, S7, S9 , S11, S14}
CC1-4 = {S1, S4, S5, S7, S9 , S11, S14}
CC1-5 = {S1, S3, S5, S7, S9 , S11, S15}
CC1-6 = {S1, S3, S6, S7, S9 , S11, S14}
CC1-7 = {S1, S3, S5, S7, S9 , S11, S16}
CC1-8 = {S1, S3, S6, S7, S9 , S11, S15}
CC1-9 = {S1, S3, S5, S7, S10, S11, S16}
CC1-10= {S1, S3, S5, S7, S9 , S11, S17}
CC1-11= {S1, S3, S5, S7, S9 , S12, S17}
CC1-12= {S1, S3, S5, S7, S9 , S11, S18}
CC1-13= {S1, S3, S5, S8, S9 , S11, S14}
CC1-14= {S1, S3, S5, S8, S9 , S11, S15}
CC1-15= {S1, S3, S5, S8, S9 , S11, S16}

The caller-callee combinations for ‘create’ and ‘cm_create_ci’ are :

CC2-1 = {S19, S21}
CC2-2 = {S20, S21}
CC2-3 = {S19, S22}

The caller-callee combinations for ‘update_version’ and ‘cm_update_version’ are :

CC3-1 = {S23, S25}
CC3-2 = {S24, S25}
CC3-3 = {S23, S26}

The caller-callee combinations for ‘release’ and ‘cm_release’ are :

CC4-1 = {S27, S29}
CC4-2 = {S28, S29}
CC4-3 = {S27, S30}

The caller-callee combinations for ‘modify’ and ‘se_modify’ are :

CC5-1 = {S31, S33}
CC5-2 = {S32, S33}
CC5-3 = {S31, S34}

The caller-callee combinations for ‘review’ and ‘re_review’ are :

CC6-1 = {S35, S37}
CC6-2 = {S36, S37}
CC6-2 = {S35, S38}

The caller-callee combinations for ‘dt_test’ and ‘te_dt_test’ are :

CC7-1 = {S39, S41}
CC7-2 = {S40, S41}
CC7-3 = {S39, S42}

The caller-callee combinations for ‘acceptance_test’ and ‘cu_acceptance_test’ are :

CC8-1 = {S43, S45}
CC8-2 = {S44, S45}
CC8-3 = {S43, S46}

4.3.2.3.5 Configuration_Item : employee-STDs

The manager STD has the following 21 employee STDs. The employees are the own internal callees ‘ch_st_to_control’, ‘ch_st_to_se_ready’, ‘ch_st_to_dt_ready’, ‘ch_st_to_baseline’, ‘ch_st_to_maint’, ‘ch_st_to_archive’ and their caller ‘cm_ch_st_ci_to_(x)’ of the class configuration_manager. These 6 operations are called in turn by the configuration_manager. The callee ‘create’ and its caller ‘cm_create’ of the class configuration_manager. The callee ‘update_version’ and its caller ‘cm_update_version’ of the configuration_manager. The callee ‘release’ and its caller ‘cm_release’ of the configuration_manager. The callee ‘modify’ and its caller ‘se_modify’ of the class software_engineer. The callee ‘review’ and its caller ‘re_review’ of the class reviewer. The callee ‘dt_test’ and its caller ‘te_dt_test’ of the test_engineer. And the callee ‘acceptance_test’ and its caller ‘cu_acceptance_test’ of the class customer.

The first 7 employees are the callees ‘ch_st_to_control’, ‘ch_st_to_se_ready’, ‘ch_st_to_dt_ready’, ‘ch_st_to_baseline’, ‘ch_st_to_maint’, ‘ch_st_to_archive’ and their caller ‘cm_ch_st_ci_to_(x)’.

Normally when using the act-construct in the caller_callee-construct, the first trap T-X of the callee is small, the one state ‘operation not active’, and the second trap T-Y is large, the rest of the states. This allows the manager to proceed as soon as possible after the start of the callee (depending also on the trap structure of the caller). All the state-changing callees of the configuration_item require however that the internal operation to be finished before the next operation may start. This ensures that the state-attribute is correctly updated. The second trap of the act-construct has to be placed after the internal operation has ended, T-Z. So the manager can only proceed to its next ‘starting’ state after the state-attribute has been updated. The second reason for the small trap is the fact that the caller is waiting after its call, and has to be notified that it can proceed because the requested state change has been applied to the CI.

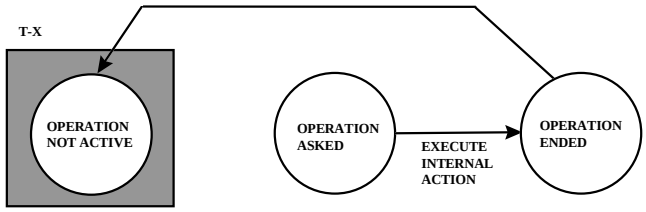


figure 4.83 act-construct : subprocess SX

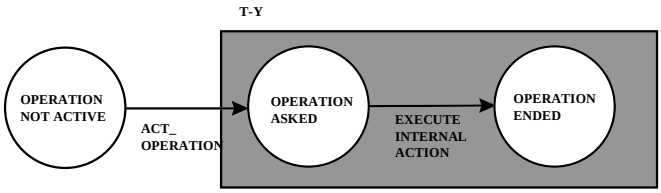


figure 4.84 act-construct : subprocess SY, large trap

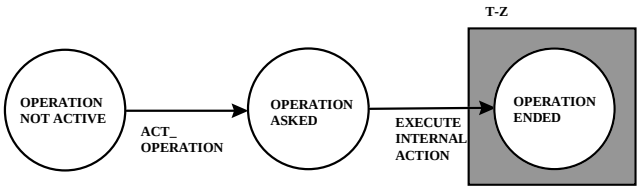


figure 4.85 act-construct : subprocess SZ, small trap

The first employee is the callee operation ‘ch_st_to_control’. This employee has two subprocesses S1 and S2 and two traps T1 and T2 according to the ‘act-construct, small second trap’.

The second employee is the callee operation ‘ch_st_to_se_ready’. This employee has two subprocesses S3 and S4 and two traps T3 and T4 according to the ‘act-construct, small second trap’.

The third employee is the callee operation ‘ch_st_to_dt_ready’. This employee has two subprocesses S5 and S6 and two traps T5 and T6 according to the ‘act-construct’, small second trap.

The fourth employee is the callee operation ‘ch_st_to_maint’. This employee has two subprocesses S7 and S8 and two traps T7 and T8 according to the ‘act-construct, small second trap’.

The 5th employee is the callee operation ‘ch_st_to_baseline’. This employee has two subprocesses S9 and S10 and two traps T9 and T10 according to the ‘act-construct, small second trap’.

The 6th employee is the callee operation ‘ch_st_to_archive’. This employee has two subprocesses S11 and S12 and two traps T11 and T12 according to the ‘act-construct, small second trap’.

The 7th employee is the caller operation ‘cm_ch_st_ci_to_(x)’. This employee has the subprocesses S13 until S18 and the traps T13, T14a, T14b, T14c, T15a, T15b, T16a, T16b, T17 and T18. The employee’s subprocesses are modeled according to the caller_callee-construct. The following figures show these subprocesses. The first figure is that of subprocess S18 showing the total internal STD. T18 is the so-called trivial trap which encompasses the whole internal STD of the operation.

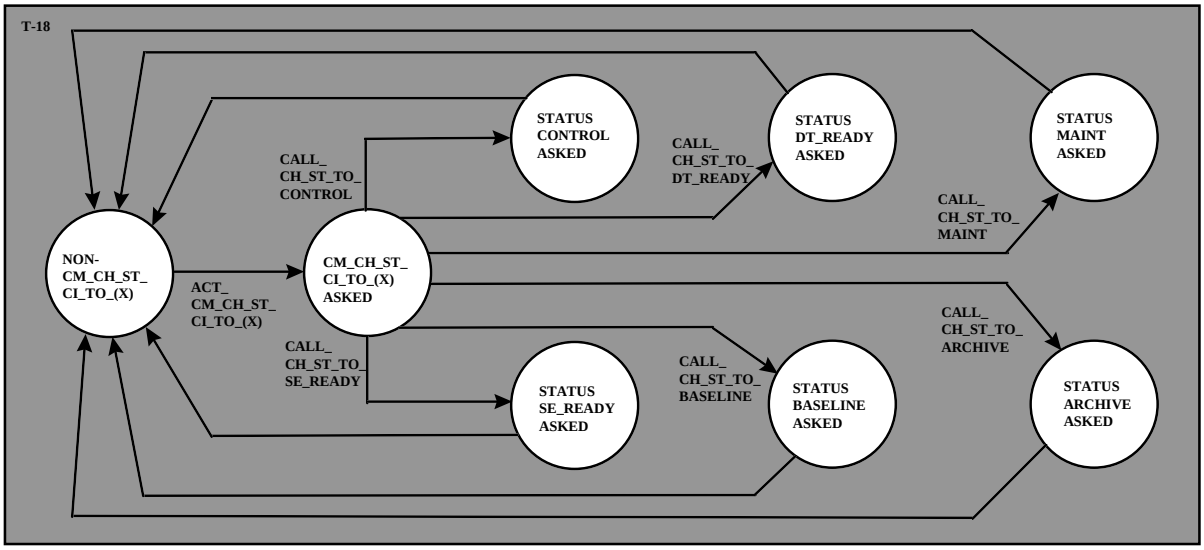


figure 4.86 employee int-cm_ch_st_ci_to_(x) : subprocess S18

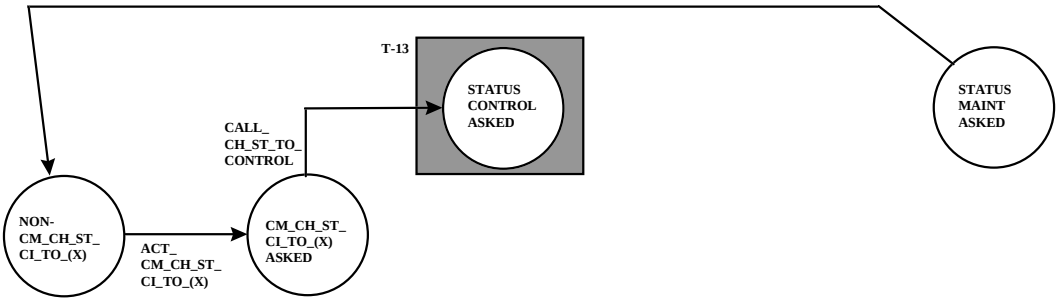


figure 4.87 employee int-cm_ch_st_ci_to_(x) : subprocess S13

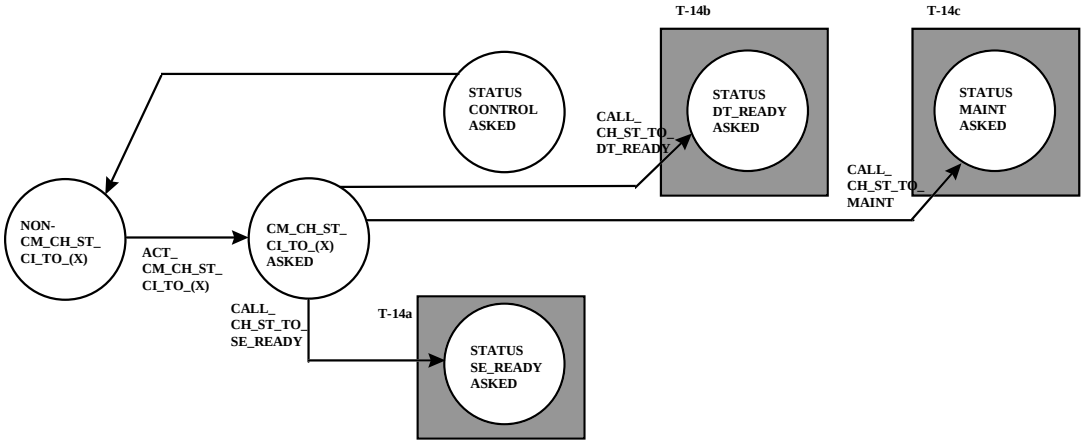


figure 4.88 employee int-cm_ch_st_ci_to_(x) : subprocess S14

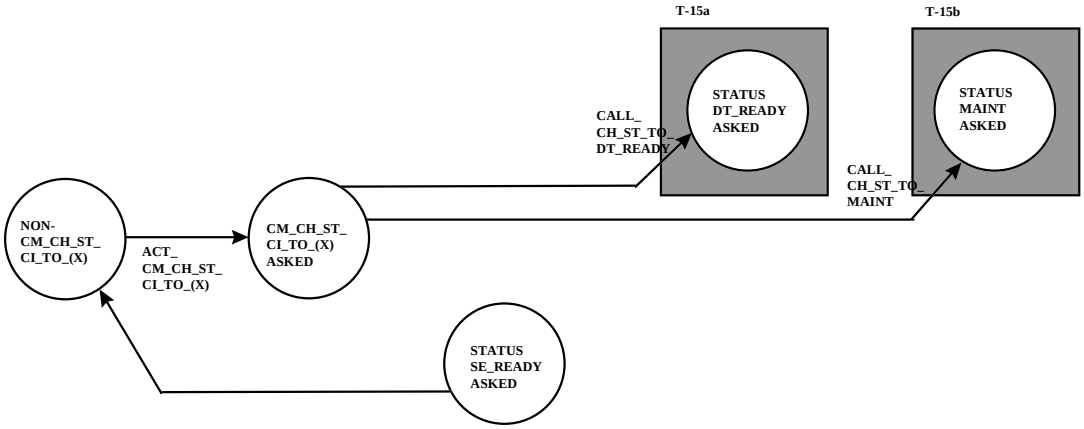


figure 4.89 employee int-cm_ch_st_ci_to_(x) : subprocess S15

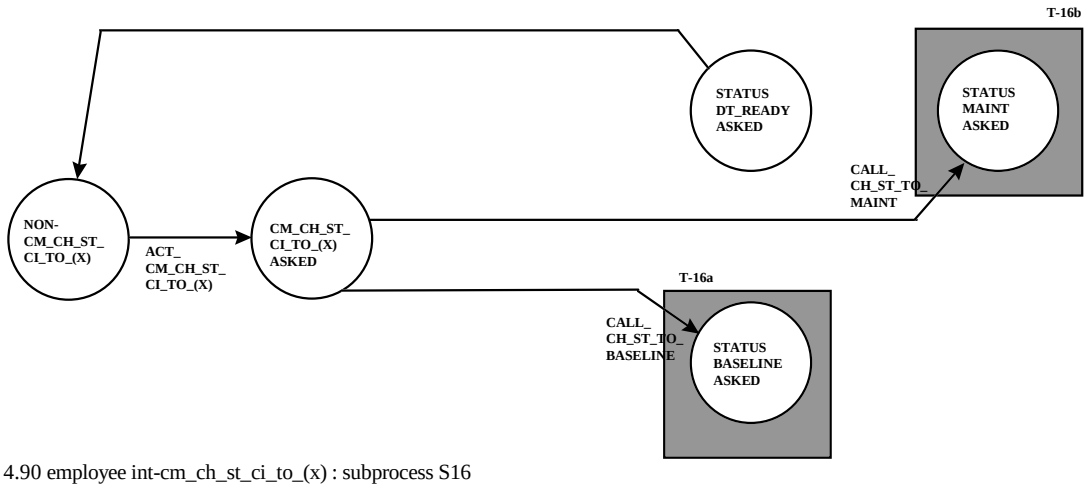


figure 4.90 employee int-cm_ch_st_ci_to_(x) : subprocess S16

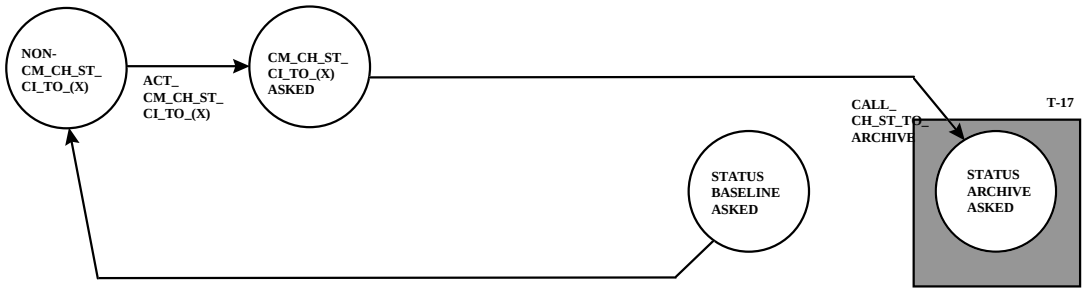


figure 4.91 employee int-cm_ch_st_ci_to_(x) : subprocess S17

The 8th and 9th employee are the callee operation ‘create’ and its caller ‘cm_create’. These are modeled according to the ‘caller-callee’-construct. The employee ‘create’ has two subprocesses S19 and S20 and two traps T19 and T20. Its internal operation has to be finished before the next operation may be started, so it is modeled with the ‘act-construct, small trap’. The second reason for the small trap is to notify the waiting caller ‘cm_create_ci’ that it can proceed. The employee ‘cm_create_ci’ has two subprocesses S21 and S22 and two traps T21 and T22.

The 10th and 11th employee are the callee operation ‘update_version’ and its caller ‘cm_update_version’. These are modeled according to the ‘caller-callee’-construct. The employee ‘update_version’ has two subprocesses S23 and S24 and two traps T23 and T24. Its internal operation has to be finished before the next operation may be started, so it is modeled with the ‘act-construct, small trap’. The second reason for the small trap is to notify the waiting caller ‘cm_update_version’ that it can proceed. The employee ‘cm_update_version’ has two subprocesses S25 and S26 and two traps T25 and T26.

The 12th and 13th employee are the callee operation ‘release’ and its caller ‘cm_release’. These are modeled according to the ‘caller-callee’-construct. The employee ‘release’ has two subprocesses S27 and S28 and two traps T27 and T28. Its internal operation has to be finished before the next operation may be started, so it is modeled with the ‘act-construct, small trap’. The second reason for the small trap is to notify the waiting caller ‘cm_release’ that it can proceed. The employee ‘cm_release’ has two subprocesses S29 and S30 and two traps T29 and T30.

The 14th and 15th employee are the callee operation ‘modify’ and its caller ‘se_modify’. These are modeled according to the ‘caller-callee’-construct. The employee ‘modify’ has two subprocesses S31 and S32 and two traps T31 and T32. Its internal operation has to be finished before the next operation may be started, so it is modeled with the ‘act-construct, small trap’. The second reason for the small trap is to notify the waiting caller ‘se_modify’ that it can proceed.The employee ‘se_modify’ has two subprocesses S33 and S34 and two traps T33 and T34.

The 16th and 17th employee are the callee operation ‘review’ and its caller ‘re_review’. These are modeled according to the ‘caller-callee’-construct. The employee ‘review’ has two subprocesses S35 and S36 and two traps T35 and T36. Its internal operation has to be finished before the next operation may be started, so it is modeled with the ‘act-construct, small trap’. The second reason for the small trap is to notify the waiting caller ‘re_review’ that it can proceed. The employee ‘re_review’ has two subprocesses S37 and S38 and two traps T37 and T38.

The 18th and 19th employee are the callee operation ‘dt_test’ and its caller ‘te_dt_test’. These are modeled according to the ‘caller-callee’-construct. The employee ‘dt_test’ has two subprocesses S39 and S40 and two traps T39 and T40. Its internal operation has to be finished before the next operation may be started so it is modeled with the ‘act-construct, small trap’. The second reason for the small trap is to notify the waiting caller ‘te_dt_test’ that it can proceed. The employee ‘te_dt_test’ has two subprocesses S41 and S42 and two traps T41 and T42.

The 20th and 21th employee are the callee operation ‘acceptance_test’ and its caller ‘cu_acceptance_test’. These are modeled according to the ‘caller-callee’-construct. The employee ‘acceptance_test’ has two subprocesses S43 and S44 and two traps T43 and T44. Its internal operation has to be finished before the next operation may be started, so it is modeled with the ‘act-construct, small trap’. The second reason for the small trap is to notify the waiting caller ‘cu_acceptance_test’ that it can proceed.

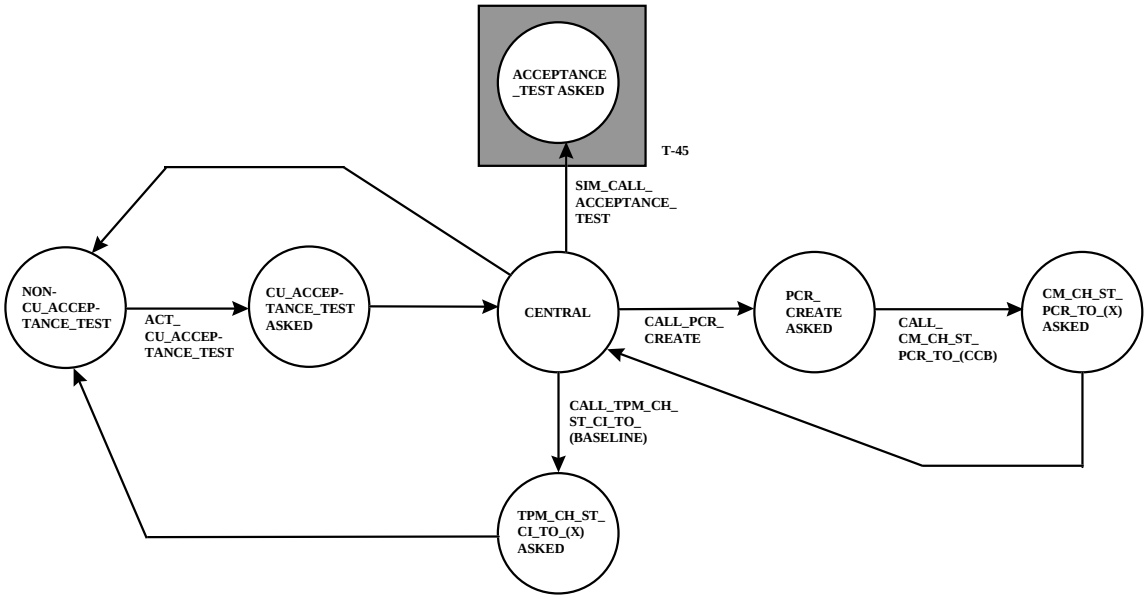


figure 4.92 employee int-cu_acceptance_test : subprocess S45

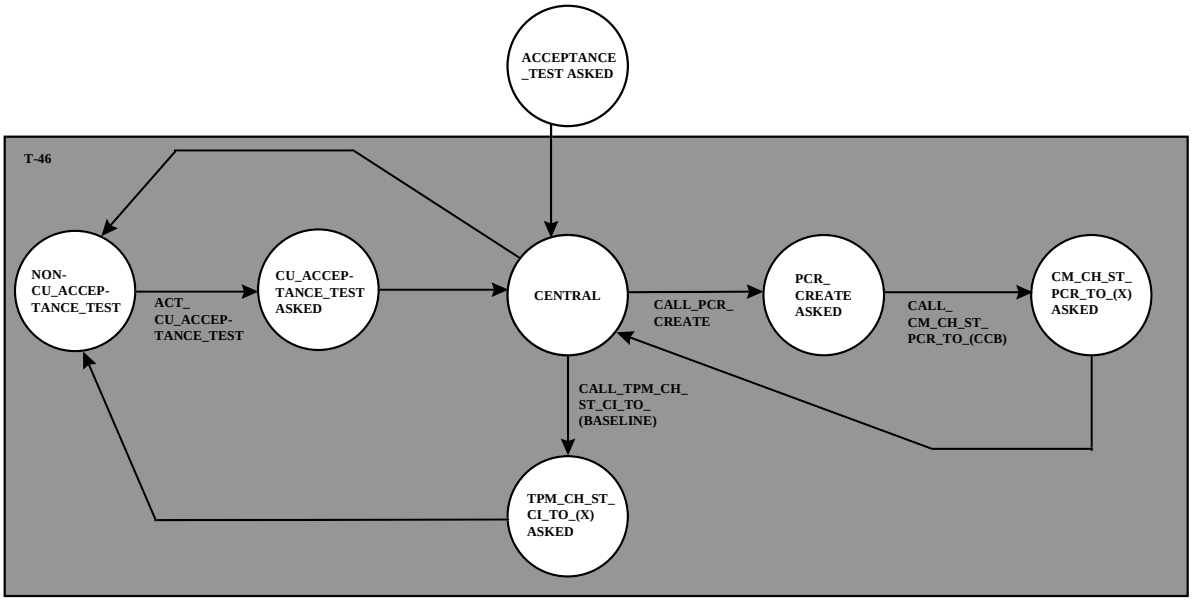


figure 4.93 employee int-cu_acceptance_test : subprocess S46

The employee ‘cu_acceptance_test’ has two subprocesses S45 and S46 and two traps T45 and T46.

4.3.2.4 Problem_and_Change_Report

4.3.2.4.1 Problem_and_Change_Report : external behavior-STD

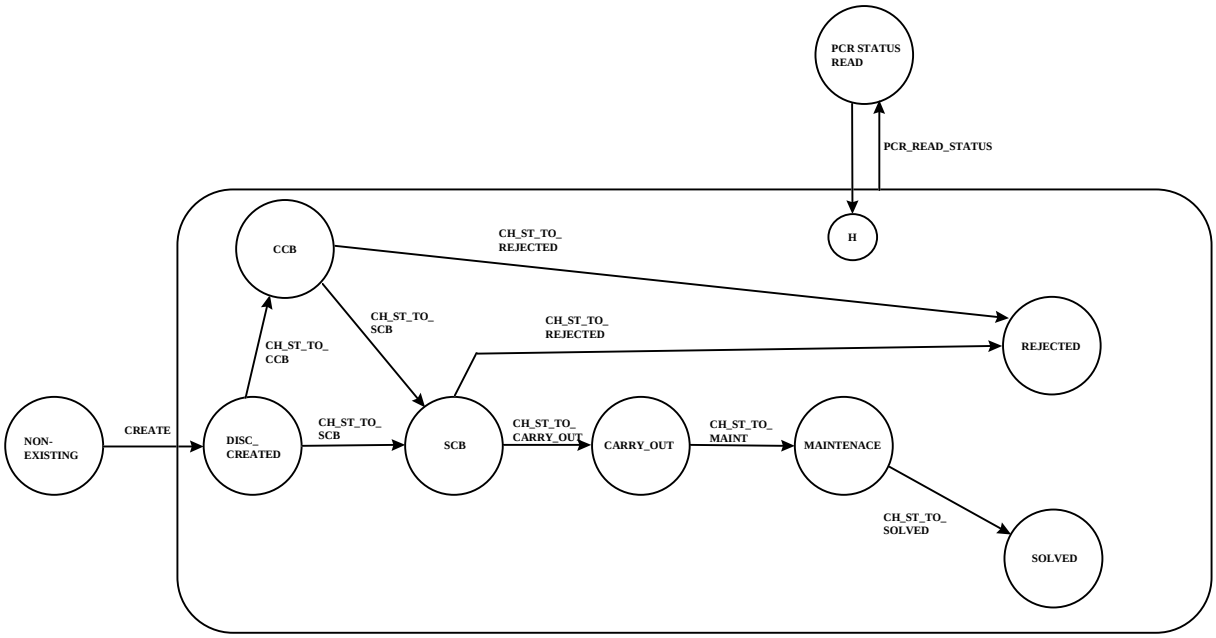


figure 4.94 problem_and_change_report: external behavior STD

This external behavior STD shows the states an object of the class ‘problem_and_change_report’ (PCR) can be in during its life cycle. Its also shows when the transitions from one state to another are made as a response to calls being made to its operations.

When the object is being created by the test-engineer or the customer it is also filled in by them. These filling in-interactions between test_engineer or customer and the problem_and_change_report object are not shown here. Because of the fact that the operation ‘pcr-create’ can be called by three different callers, ‘te_dt_test’, ‘cu_acceptance_test’ and ‘cu_issue_pcr’, the state ‘disc_created’ is a discriminator state which handles the three callers.

Aftter creation the PCR can then offered to the software_configuration_board, if it is of category A or C. The PCR status then becomes ‘SCB’. If it is of category B it will be offered to the configuration_control_board. The PCR status is then changed to ‘CCB’.

If the configuration_control_board decides that the problem_and_change_report has to be solved, it gives it to the software_configuration_board. The status of the PCR becomes ‘SCB’. If the configuration_control_board decides to reject the PCR, the new status will be ‘rejected’.

The software_configuration_board decides on the PCRs it receives, including the ones coming from the configuration_ control_board, and decides what to do with them. The ones to be carried out get the status ‘carry_out’. Or the software_configuration_board can reject them. In that case the PCR status becomes ‘rejected’.

When the project_manager clusters the problem_and_change_reports into logical groups to be worked upon, they get the status ‘maintenance’.

When the test_engineer has checked a PCR during the dt-test and found it to be ok, the problem_and_change_report’s status changes to ‘solved’.

In all states of the STD (excluding the state ‘non-existing’) the operation ‘pcr_read_status’ can be called. The PCR goes to the state ‘pcr status read’. Here the internal operation ‘pcr_read_status’ is started and the PCR transits back to the state it was in before it was called. Before it does so it returns the PCR status to the caller. The fact that the operation

'pcr_read_status' can be called in (almost) every state of the external STD, is modeled by the transition from the XOR-superstate (the big rectangle with the rounded corners) to the state 'pcr status read'. The fact that it remembers the state it was in when it was called and transits back to that state is modeled by the transition to the 'history state indicator', the small circle containing the letter 'H'. Superstate and history indicator are part of the UML notation for state diagrams, which is essentially the Harel statechart notation [HAR].

4.3.2.4.2 Problem_and_Change_Report : internal behavior-STDs

The problem_and_change_report has 8 operations : 'pcr_create', 'pcr_read_status', 'ch_st_to_scb', 'ch_st_to_ccb', 'ch_st_to_rejected', 'ch_st_to_carry_out', 'ch_st_to_maint' and 'ch_st_to_closed'. These operations have the following internal behavior STDs.

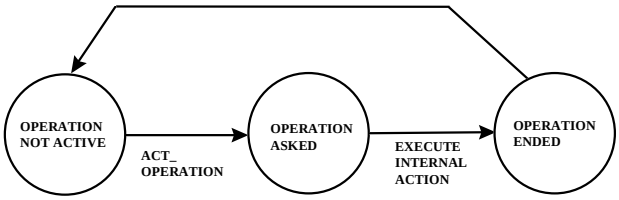


figure 4.95 only_internal_action-construct : internal behavior STD

All operations can be modeled by the 'only_internal_action-construct' since they don't execute any calls to other objects. They all follow the construct's template and their actions are explained in the following table.

OPERATION	INTERNAL ACTION
CH_ST_TO_SCB	UPDATE STATUS ATTRIBUTE TO SCB
CH_ST_TO_CCB	UPDATE STATUS ATTRIBUTE TO CCB
CH_ST_TO_REJECTED	UPDATE STATUS ATTRIBUTE TO REJECTED
CH_ST_TO_CARRY_OUT	UPDATE STATUS ATTRIBUTE TO CARRY_OUT
CH_ST_TO_MAINT	UPDATE STATUS ATTRIBUTE TO MAINT
CH_ST_TO_CLOSED	UPDATE STATUS ATTRIBUTE TO CLOSED
PCR_CREATE	CREATE NEW OBJECT AND FILL IT IN (=MODIFY THE DESCRIPTION ATTRIBUTE)
PCR_READ_STATUS	RETURN VALUE OF STATUS ATTRIBUTE

table 4.3 only_internal_action-construct : operations and actions

Although the 'pcr_create' has an interaction with other objects, the filling in of the report, it can be modeled by the 'only_internal_action-construct' because the filling in of the pcr is not further detailed. The formal parameter of 'pcr_create' is the updated (part of the) description of the pcr.

The operation 'pcr_read_status' has no formal parameters, but it returns a status value to the caller. The operations 'ch_st_to_scb', 'ch_st_to_ccb', 'ch_st_to_rejected', 'ch_st_to_carry_out', 'ch_st_to_maint' and 'ch_st_to_closed' have no formal parameters.

4.3.2.4.3 Problem_and_Change_Report : manager-STD

The communication between the problem_and_change_report's operations (callees) and their callers is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

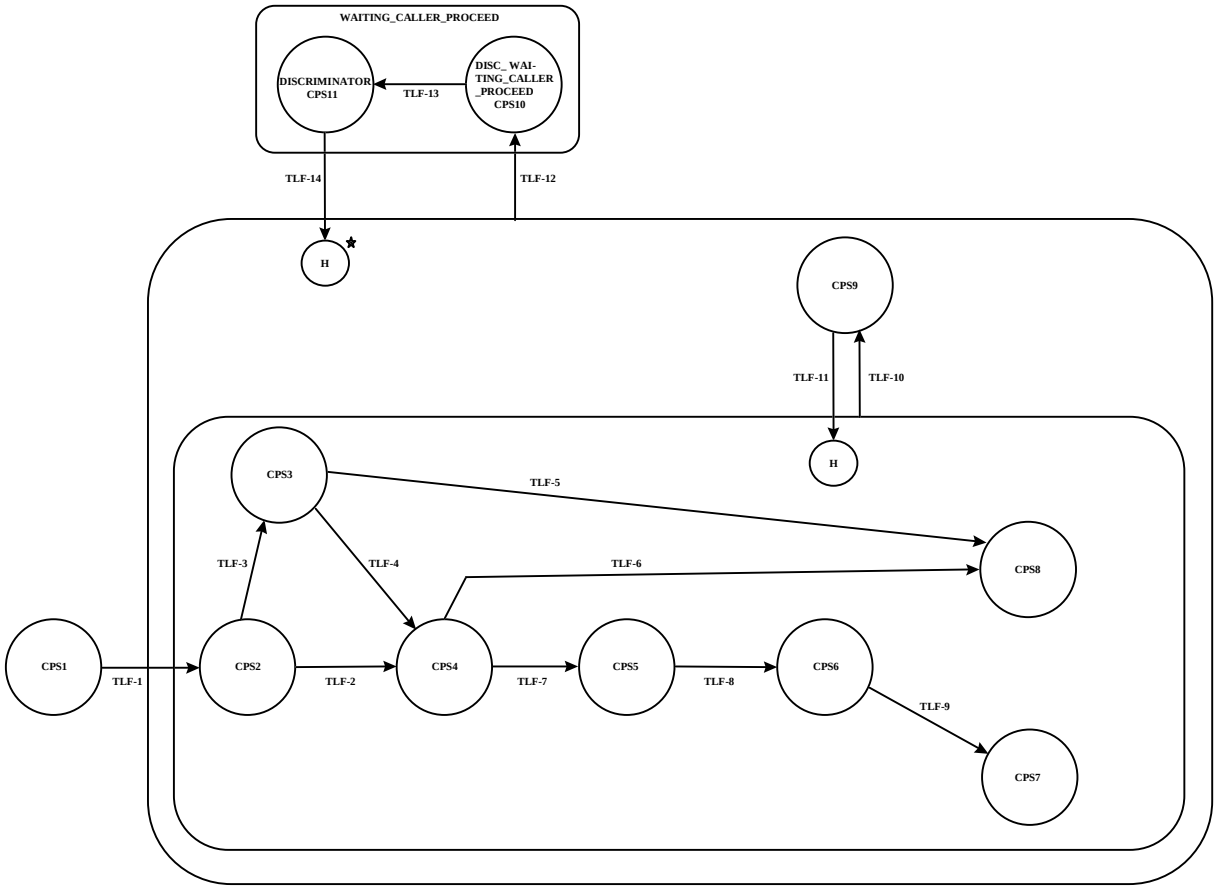


figure 4.96 problem_and_change_report : manager STD

Every caller of every operation of the class problem_and_change_report has to wait after the call, for the callee to reach some point in its execution. The status has to be updated before the caller may proceed. The problem_and_change_ report has to be created before the caller may proceed. The operation ‘pcr_read_status’ must return the status to the caller, before the caller may proceed.

To model this, an AND-superstate is created in the external STD and an extra state ‘waiting_caller_proceed’ is added to it. The state ‘waiting_caller_proceed’ consists of the substates ‘disc_waiting_caller_proceed’ and ‘discriminator’. From every state inside the AND-superstate the external STD can go to the state ‘waiting_caller_proceed’. From the state ‘waiting_caller_proceed’ the external STD transits back to to state it was in before it entered the state ‘waiting _caller_ proceed’. This is modeled by the ‘recursive history state’ ‘H’. ‘Recursive’ means that the STD returns to the correct level of depth (i.e the level shown in the STD) inside some aggregate state inside the AND-superstate. The recursion is indicated by the asterix (*) next to the ‘history’ state [HAR].

Every callee is modeled by the ‘act-construct, small second trap’. When some callee has progressed far enough in its execution (i.e. has entered its small trap), the external STD transits to ‘disc_waiting_caller_proceed’. Here it decides which callee has ended, and it allows its caller to proceed (i.e prescribes the next subprocess for the caller). When the caller has indeed proceeded (i.e. has entered the trap of its next subprocess), the external STD transits to the ‘discriminator’ state. Here it decides which caller it was that was allowed to proceed. For this caller it then prescribes its next subprocess. Then the external STD transits to the state it was in before it entered ‘waiting_caller_proceed’.

Also the second trap of the callee has to be small, because it has to finish performing its function before the manager STD may start the next operation (be it the same operation or another one).

An precaution is needed to ensure that when an callee is executing, the next operation can only be started after the first callee has ended. This takes the form of an extra ‘guard’ on the ‘operation-starting’-transition from one state to the next. This transition is guarded with the trap encompassing the ‘non-active’ state of the previous (possibly still executing) operation. So this transition can only be taken when the the previous operation has been prescribed it next (=first) subprocess in which it can reach its trap ‘non-active’ (it finishes its execution). The the previous operation is prescibed its next (=first) subprocess in the state ‘waiting_caller_ proceed’. So the external STD must first go to its state

'waiting_caller_proceed' before the next operation can be started. For example, in the state 'carry_out', the external STD must first take the transition from 'carry_out' to 'waiting_caller_ proceed' before it can take the transition from 'carry_out' to 'maintenance' (on which transition the next operation is started).

The notation CPSx in the STD stands for Consolidated Prescribed Subprocesses and is a set of CCx's. The notation CCx in the STD stands for 'Caller(s)-Callee' and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for 'Traps Logical Formula' and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

The CPS's and the TLF's for this manager are :

CPS1 = {CC1-1, CC2-1, CC3-1}
TLF-1 = T-19 and (T-21 or T-23 or T29)

CPS2 = {CC1-1, CC2-2 or CC2-1, CC3-1}
TLF-2 = T-19 and (T-1 and T-13b)
TLF-3 = T-19 and (T-3 and T-13a)

CPS3 = {CC1-4 or CC1-5, CC2-1, CC3-1}
TLF-4 = T-3 and (T-1 and T-14b)
TLF-5 = T-3 and (T-5 and T-14a)

CPS4 = {(CC1-2 or CC1-6) or CC1-3, CC2-1, CC3-1}
TLF-6 = T-1 and (T-5 and T-15a)
TLF-7 = T-1 and (T-7 and T-15b)

CPS5 = {CC1-10 or CC1-11, CC2-1, CC3-1}
TLF-8 = T-7 and (T-9 and T-16a)

CPS6 = {CC1-12 or CC1-13, CC2-1, CC3-1}
TLF-9 = T-9 and (T-11 and T-17a)

CPS7 = {CC1-14 or CC1-8, CC2-1, CC3-1}

CPS8 = {(CC1-7 or CC1-9) or CC1-8, CC2-1, CC3-1}

The state 'pcr status read' consists of 6 substates which are entered depending on the state the STD was in when the call to 'pcr_ready' occurred. This is reflected in the CPS9 and the TLF-10. Also reflected in TLF-10 is the fact that writing of the status attribute has to be finished before it may be read.

CPS9 = {CC1-1, CC2-1, CC3-2} or
{CC1-3, CC2-1, CC3-2} or
{CC1-5, CC2-1, CC3-2} or
{CC1-8, CC2-1, CC3-2} or
{CC1-11, CC2-1, CC3-1} or
{CC1-13, CC2-1, CC3-1}
TLF-10 = (T-25 and T-27) and (T-19 or T-1 or T-3 or T-5 or T-7 or T-9 or T-11)
TLF-11 = T-25

The state 'disc_waiting_caller_proceed' consists of 7 substates which are entered depending on the state the STD was in when one of the callees entered its small trap. This is reflected in CPS10 and TLF-12 and TLF-13.

CPS10 = {CC1-1, CC2-3, CC3-1} or
{CC1-3, CC2-1, CC3-1} or
{CC1-5, CC2-1, CC3-1} or
{CC1-8, CC2-1, CC3-1} or
{CC1-11, CC2-1, CC3-1} or
{CC1-13, CC2-1, CC3-1} or
{CC1-x as prescribed in CPS10, CC2-x as prescribed in CPS10, CC3-3}
TLF-12 = T-20 or T-2 or T-4 or T-6 or T-8 or T-10 or T-12 or T-26
TLF-13 = (T-22 or T-24 or T-30) or T-15c or T-14c or T-18 or T-16b or T-17b or T-28

The state ‘discriminator’ consists of 7 substates which are entered depending on the state the STD was in when one of the callees entered its small trap. This is reflected in CPS11. The transition TLF-14 is automatic, so no traps are guarding it.

CPS11 = {CC1-1, CC2-1, CC3-1} or
{CC1-3, CC2-1, CC3-1} or
{CC1-5, CC2-1, CC3-1} or
{CC1-8, CC2-1, CC3-1} or
{CC1-11, CC2-1, CC3-1} or
{CC1-13, CC2-1, CC3-1} or
{CC1-x as prescribed in CPS10, CC2-x as prescribed in CPS10, CC3-1}
TLF-14 = (-) (automatic transition)

The caller-callee combinations for ‘ch_st_to_x’ and ‘cm_ch_st_pcr_to_(x)’ are :

CC1-1 = {S1, S3, S5, S7, S9 , S11, S13}
CC1-2 = {S2, S3, S5, S7, S9 , S11, S13}
CC1-3 = {S1, S3, S5, S7, S9 , S11, S15}
CC1-4 = {S1, S4, S5, S7, S9 , S11, S13}
CC1-5 = {S1, S3, S5, S7, S9 , S11, S14}
CC1-6 = {S2, S3, S5, S7, S9 , S11, S14}
CC1-7 = {S1, S3, S6, S7, S9 , S11, S14}
CC1-8 = {S1, S3, S5, S7, S9 , S11, S18}
CC1-9 = {S1, S3, S6, S7, S9 , S11, S15}
CC1-10= {S1, S3, S5, S8, S9 , S11, S15}
CC1-11= {S1, S3, S5, S7, S9 , S11, S16}
CC1-12= {S1, S3, S5, S7, S10, S11, S16}
CC1-13= {S1, S3, S5, S7, S9 , S11, S17}
CC1-14= {S1, S3, S5, S7, S9 , S12, S17}

The caller-callee combinations for ‘pcr_create’ and ‘te_dt_test’, ‘cu_acceptance_test’ and ‘cu_issue_pcr’ are the following. The state ‘disc_create’ is a discriminator state and this is reflected in CC2-2.

CC2-1 = {S19, S21, S23, S29}
CC2-2 = {S20, S21, S23, S29}
CC2-3 = {S19, S22, S23, S29} or
{S19, S21, S24, S29} or
{S19, S21, S23, S30}

The caller-callee combinations for ‘pcr_read_status’ and ‘tpm_cluster_pcr’ are :

CC3-1 = {S25, S27}
CC3-2 = {S26, S27}
CC3-3 = {S25, S28}

4.3.2.4.4 Problem_and_Change_Report : employee-STDs

The manager STD has the following 13 employee STDs. The employees are the own internal callees ‘ch_st_to_scb’, ‘ch_st_to_ccb’, ‘ch_st_to_rejected’, ‘ch_st_carry_out’, ‘ch_st_to_maint’, ‘ch_st_to_solved’ and their caller ‘cm_ch_st_pcr_to_(x)’ of the class configuration_manager. These 6 operations are called in turn by the configuration_manager. The callee ‘pcr_create’ and its 3 callers ‘te_dt_test’ of the class test_engineer, ‘cu_acceptance_test’ and ‘cu_issue_pcr’ of the class customer. The callee ‘pcr_read_status’ and its caller ‘tpm_cluster_pcr’ of the technical_project_manager.

The first 7 employees are the callees ‘ch_st_to_scb’, ‘ch_st_to_ccb’, ‘ch_st_to_rejected’, ‘ch_st_to_carry_out’, ‘ch_st_to_maint’, ‘ch_st_to_solved’ and their caller ‘cm_ch_st_pcr_to_(x)’. The status changing operations must be completed before the next operation may be called, therefore they are modeled by the ‘small trap’-variant of the act-construct. The second reason for the small trap is the fact that the caller is waiting after its call, and has to be notified that it can proceed because the requested state change has been applied to the PCR.

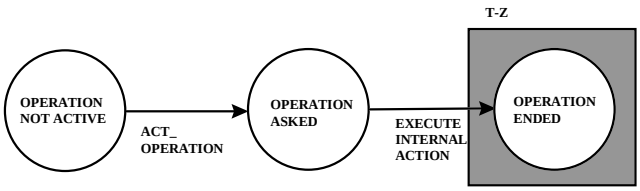


figure 4.97 act-construct : subprocess SZ, small trap

The first employee is the callee operation ‘ch_st_to_scb’. This employee has two subprocesses S1 and S2 and two traps T1 and T2 according to the ‘act-construct, small second trap’.

The second employee is the callee operation ‘ch_st_to_ccb’. This employee has two subprocesses S3 and S4 and two traps T3 and T4 according to the ‘act-construct, small second trap’.

The third employee is the callee operation ‘ch_st_to_rejected’. This employee has two subprocesses S5 and S6 and two traps T5 and T6 according to the ‘act-construct’, small second trap.

The fourth employee is the callee operation ‘ch_st_to_carry_out’. This employee has two subprocesses S7 and S8 and two traps T7 and T8 according to the ‘act-construct, small second trap’.

The 5th employee is the callee operation ‘ch_st_to_maint’. This employee has two subprocesses S9 and S10 and two traps T9 and T10 according to the ‘act-construct, small second trap’.

The 6th employee is the callee operation ‘ch_st_to_solved’. This employee has two subprocesses S11 and S12 and two traps T11 and T12 according to the ‘act-construct, small second trap’.

The 7th employee is the caller operation ‘cm_ch_st_pcr_to_(x)’. This employee has the subprocesses S13 until S18 and the traps T13a, T13b, T14a, T14b, T14c, T15a, T15b, T15c, T16a, T16b, T17a, T17b and T18. The employee’s subprocesses are modeled according to the caller_callee-construct. The following figures show these subprocesses. The first figure is that of subprocess S18 showing the total internal STD. T18 is the so-called trivial trap which encompasses the whole internal STD of the operation.

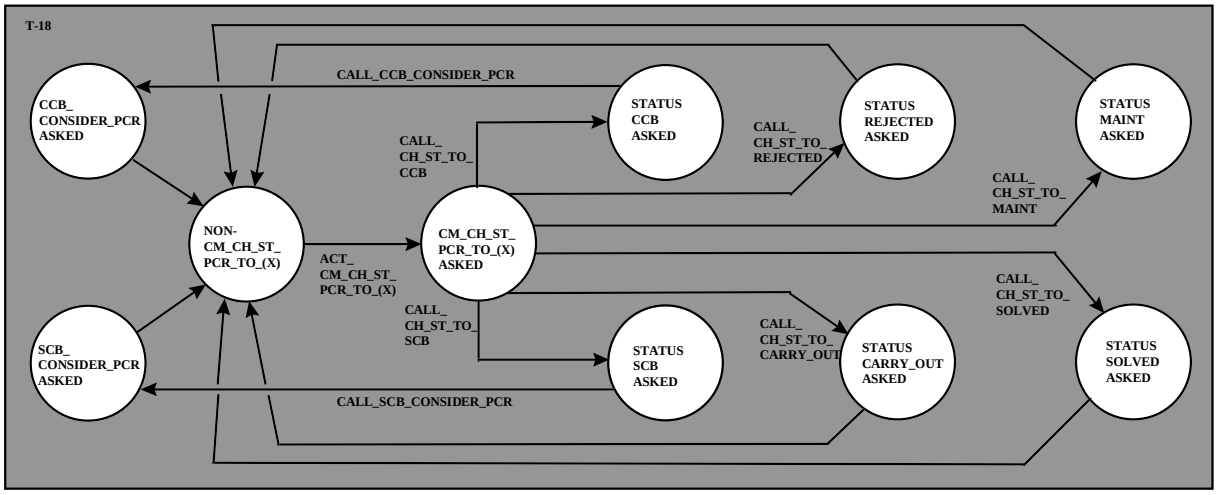


figure 4.98 employee int-cm_ch_st_pcr_to_(x) : subprocess S18

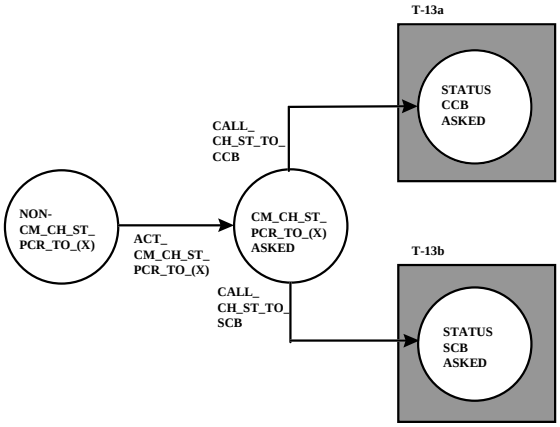


figure 4.99 employee int-cm_ch_st_pcr_to_(x) : subprocess S13

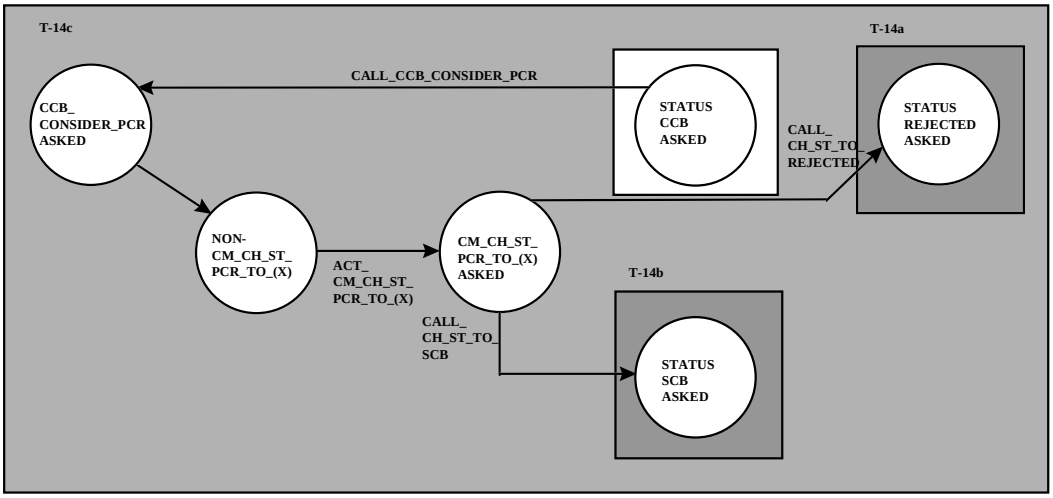


figure 4.100 employee int-cm_ch_st_pcr_to_(x) : subprocess S14

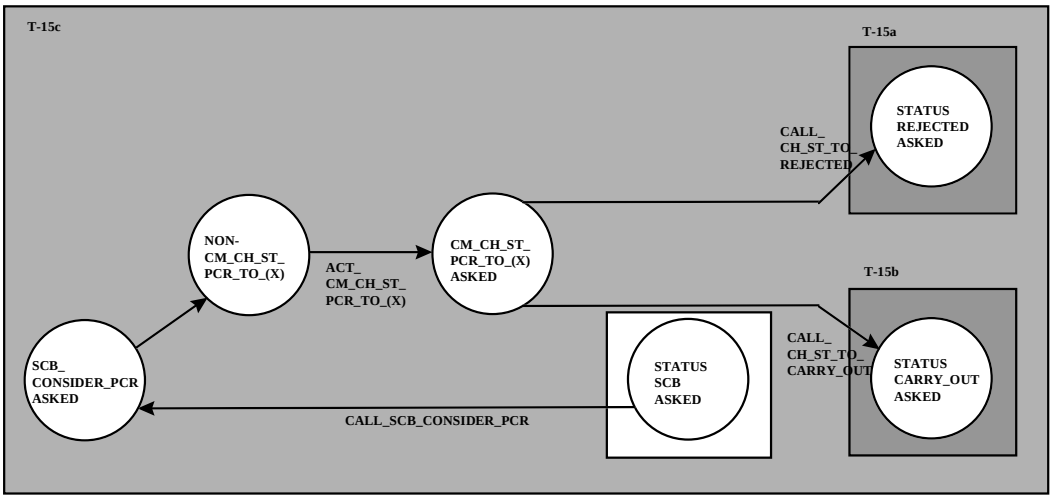


figure 4.101 employee int-cm_ch_st_pcr_to_(x) : subprocess S15

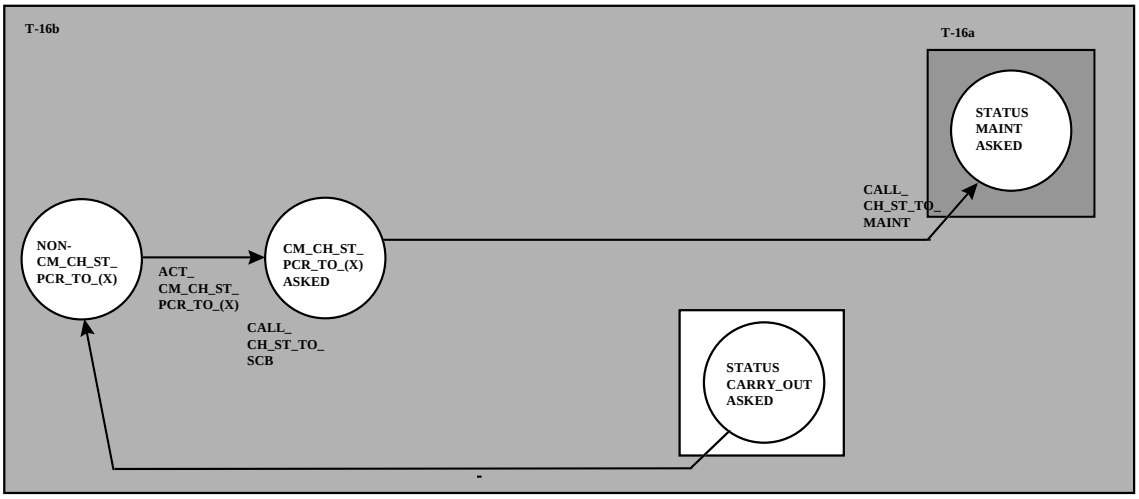


figure 4.102 employee int-cm_ch_st_pcr_to_(x) : subprocess S16

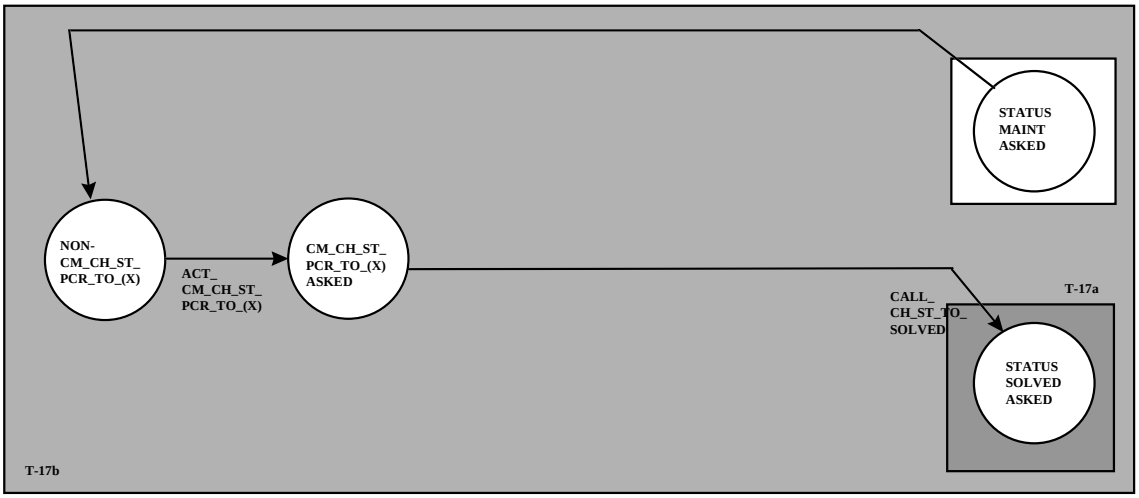


figure 4.103 employee int-cm_ch_st_pcr_to_(x) : subprocess S17

The eighth, nineth, tenth and 13th employee are the callee operation ‘pcr_create’ and its 3 callers ‘te_dt_test’ , ‘cu_acceptance_test’ and ‘cu_issue_pcr’ . These are modeled according to the ‘caller-callee’-construct. The employee ‘create’ has two subprocesses S19 and S20 and two traps T19 and T20. Its internal operation has to be finished before another operation may be started, so it is modeled with the ‘act-construct, small trap’. The second reason for the small trap is to notify the waiting that it can proceed. The employee ‘te_dt_test’ has two subprocesses S21 and S22 and two traps T21 and T22. The employee ‘cu_acceptance_test’ has two subprocesses S23 and S24 and two traps T23 and T24. The employee ‘cu_issue_pcr’ has two subprocesses S29 and S30 and two traps T29 and T30. The state ‘disc-created’ is a discriminator state which establishes which caller has executed the call and it prescribes the subprocesses accordingly.

The 11th and 12th employee are the callee operation ‘pcr_read_status’ and its caller ‘tpm_cluster_pcr’. These are modeled according to the ‘caller-callee’-construct. The employee ‘pcr_read_status’ has two subprocesses S25 and S26 and two traps T25 and T26. The status has to be read and returned to the caller before a new operation may be started, so it is modeled with the ‘act-construct, small trap’. The second reason for the small trap is to notify the waiting that it can proceed. The employee ‘tpm_cluster_pcr’ has two subprocesses S27and S28 and two traps T27 and T28. The operation ‘tpm_cluster_pcr’ uses the ‘simultaneous_call’-construct to read the status of a number of pcr’s simultaneously.

4.3.2.5 Software_Engineer

4.3.2.5.1 Software_Engineer : external behavior STD

The STD of the external behavior consists of a neutral state in which the software_engineer waits for a call to the only operation it makes available to other objects and of an activation state in which the internal ‘se_modify’ operation is started.

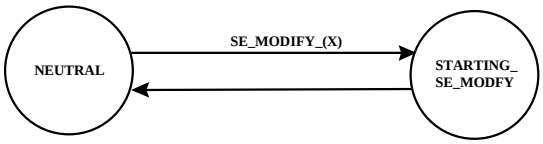


figure 4.104 software_engineer : external behavior STD

The operation takes the id of the configuration_item that has to be modified as its only formal parameter. The software_engineer does not wait until the ‘se_modify’ operation is finished, but returns as soon as possible to its neutral state. This increases the level of concurrency. Because the ‘se_modify’ operation can be called as well as activated again while the current execution of ‘se_modify’ is not yet finished. (So this is multi-threaded : simultaneous execution of different method invocations.)

4.3.2.5.2 Software_Engineer : internal behavior-STDs

The 1 operation ‘se_modify’ of the software_engineer has the following internal behavior STD.

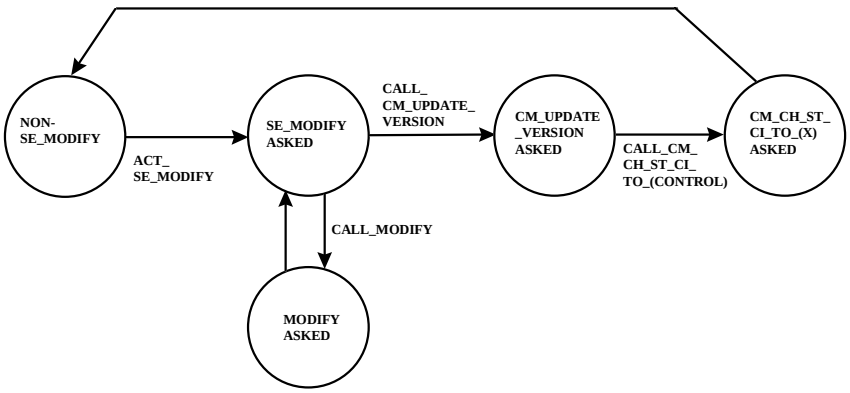


figure 4.105 int-se_modify : internal behavior STD

After the operation ‘se_modify’ has started, the software_engineer modifies the configuration_item (CI) one piece at the time (seperate calls to modify). If he is finished he asks the configuration_manager to update the version of the CI.

Next the software_engineer calls the operation ‘cm_ch_st_ci_to_(x)’ to ask the configuration_manager to change the status of the configuration_item to ‘control’ (i.e. put the CI under configuration management).

4.3.2.5.3 Software_Engineer : manager-STD

The communication between the software_engineer’s operation (callee) and its caller is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

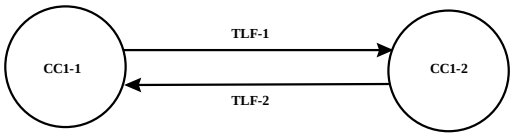


figure 4.106 software_engineer : manager STD

The notation CCx in the STD stands for 'Caller(s)-Callee' and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for 'Traps Logical Formula' and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

The CC's and the TLF's for this manager are the combinations for 'se_modify' and 'tpm_modify' :

CC1-1 = {S1, S3}
TLF-1 = T-1 and T-3

CC1-2 = {S2, S4}
TLF-2 = T-2 and T-4

4.3.2.5.4 Software_Engineer : employee-STDs

The manager STD has the following 2 employee STDs : the own internal callee 'se_modify' and its caller 'tpm_modify' of the class 'technical_project_manager'.

The first employee is the callee operation 'se_modify'. This employee has two subprocesses S1 and S2 and two traps T1 and T2 according to the 'act-construct, big second trap'.

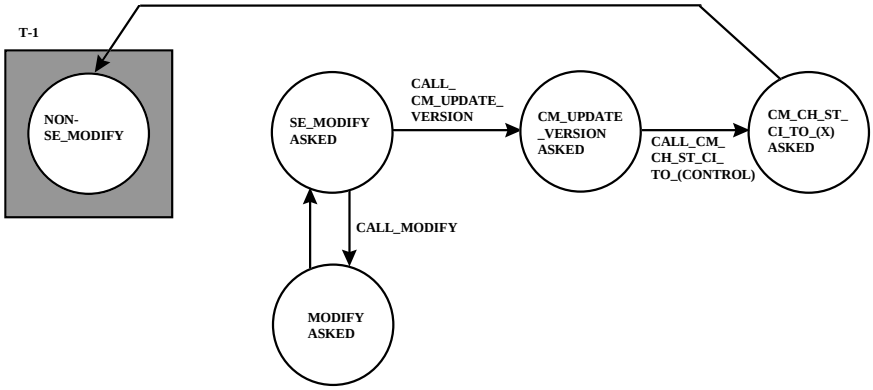


figure 4.107 employee int-se_modify : subprocess S1

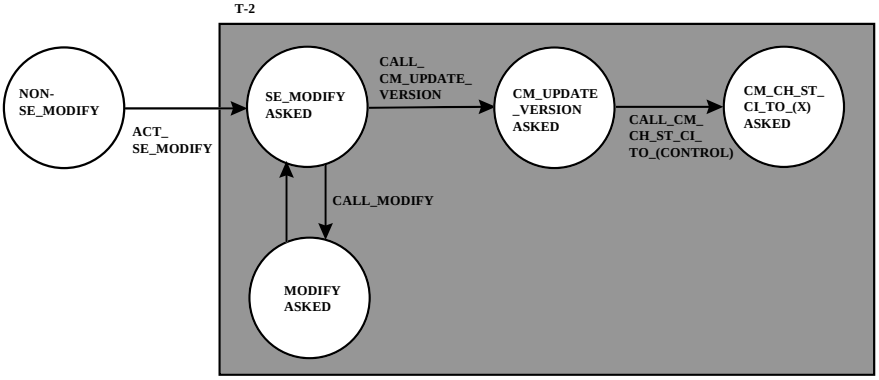


figure 4.108 employee int-se_modify : subprocess S2, big trap

The second employee is the caller operation 'tpm_modify'. This employee has two subprocesses S3 and S4 and two traps T3 and T4 according to the caller_callee-construct.

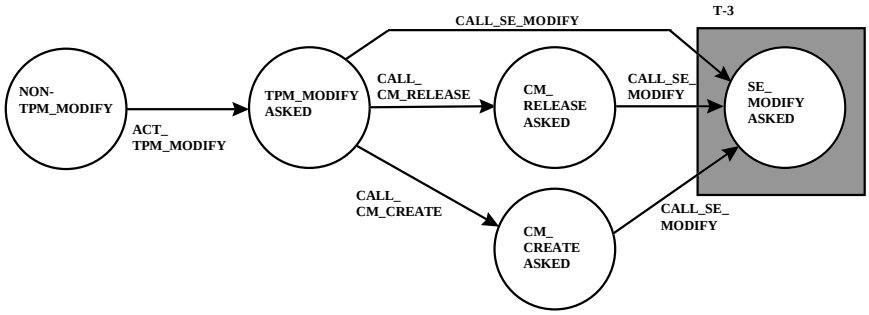


figure 4.109 employee int-tpm_modify : subprocess S3

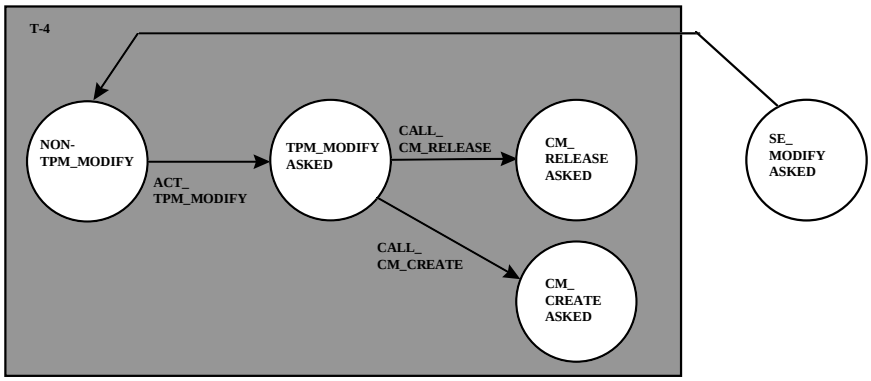


figure 4.110 employee int-tpm_modify : subprocess S4

4.3.2.6 Reviewer

4.3.2.6.1 Reviewer : external behavior-STD

The STD of the external behavior consists of a neutral state in which the reviewer waits for a call to the only operation it makes available to other objects and of an activation in which the internal ‘re_review’ operation is started.

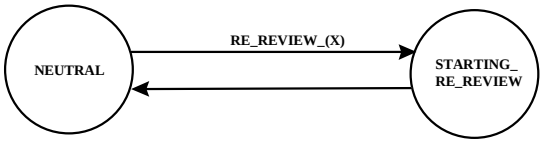


figure 4.111 reviewer : external behavior STD

The formal parameter of the operation is the id of the configuration_item that has to be reviewed. The reviewer does not wait until the ‘re_review’ operation is finished, but returns as soon as possible to its neutral state. This increases the level of concurrency. Because the ‘re_review’ operation can be called again as well as activated while the current execution has not yet terminated.

4.3.2.6.2 Reviewer : internal behavior-STDs

The 1 operation of the reviewer has the following internal behavior STD.

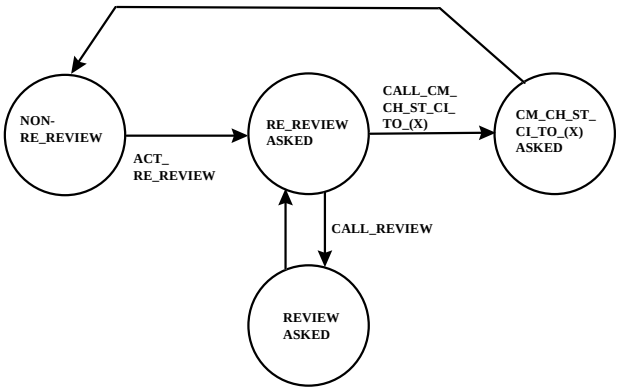


figure 4.112 int-re_review : internal behavior STD

After the operation ‘re_review’ has started, the reviewer reviews the configuration_item (CI) one piece at the time (seperate calls to the ‘review’ operation of the configuration_item). When the review is finished , the reviewer asks the configuration_manager to change the status of the CI. If the review is ok, then the status change is to ‘se_ready’ (call_cm_ch_st_ci_to_(se_ready)). If the review is not ok, then the the status change is to ‘maintenance’ (call_cm_ch_st_ci_to_(maint)).

4.3.2.6.3 Reviewer : manager-STD

The communication between the reviewer’s operation (callee) and its caller is managed by a manager STD.
The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

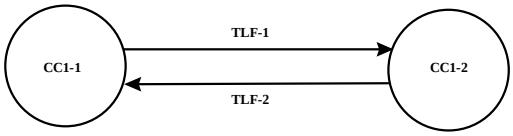


figure 4.113 reviewer : manager STD

The notation CCx in the STD stands for ‘Caller(s)-Callee’ and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for ‘Traps Logical Formula’ and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

The CC’s and the TLF’s for this manager are the combinations for ‘re_review’ and ‘tpm_review’ :

CC1-1 = {S1, S3}
TLF-1 = T-1 and T-3

CC1-2 = {S2, S4}
TLF-2 = T-2 and T-4

4.3.2.6.4 Reviewer : employee-STDs

The manager STD has the following 2 employee STDs : the own internal callee ‘re_review’ and its caller ‘tpm_review’ of the class ‘technical_project_manager’.

The first employee is the callee operation ‘re_review’. This employee has two subprocesses S1 and S2 and two traps T1 and T2 according to the ‘act-construct, big second trap’.

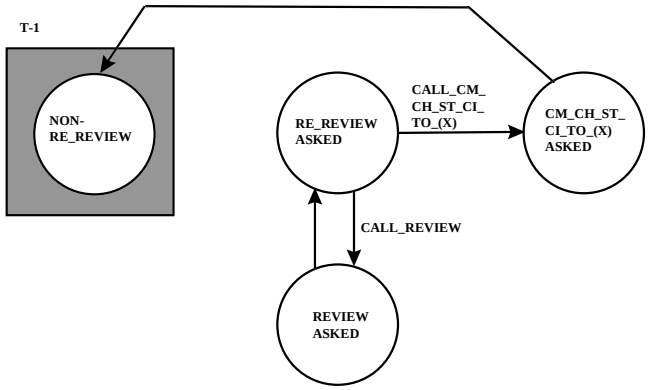


figure 4.114 employee int-re_review : subprocess S1

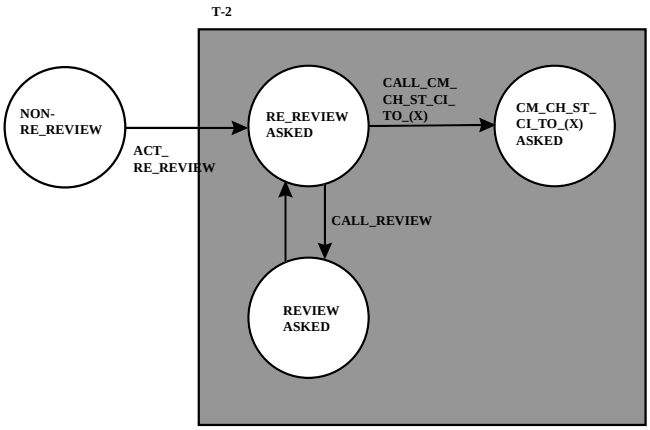


figure 4.115 employee int-re_review : subprocess S2, big trap

The second employee is the caller operation ‘tpm_review’. This employee has two subprocesses S3 and S4 and two traps T3 and T4 according to the caller_callee-construct.

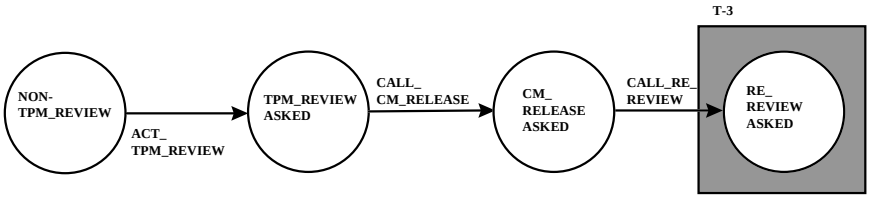


figure 4.116 employee int-tpm_review : subprocess S3

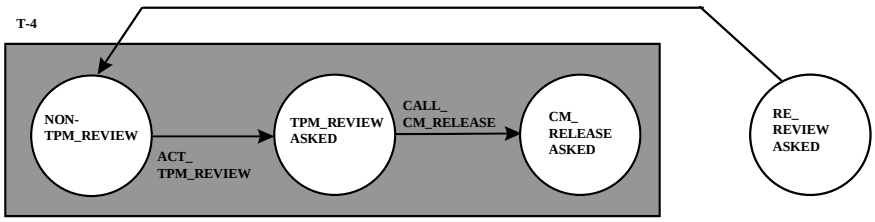


figure 4.117 employee int-tpm_review : subprocess S4

4.3.2.7 Software_Configuration_Board

4.3.2.7.1 Software_Configuration_Board : external behavior-STD

The STD of the external behavior consists of a neutral state in which the software_configuration_board waits for a call to the internal operation and of an activation state in which the internal 'scb_consider_(x)' operation is started.

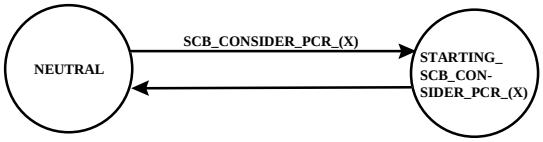


figure 4.118 software_configuration_board : external behavior STD

The formal parameter of the operation is the id of the problem_and_change_report that has to be considered. The software_configuration_board does not wait for the 'scb_consider' operation to finish, but returns as soon as possible to its neutral state. This increases the level of concurrency. The operation can be called again as well as activated while the current execution is stil taking place.

4.3.2.7.2 Software_Configuration_Board : internal behavior-STDs

The 1 operation 'csb_consider_pcr' of the software_configuration_board has the following internal behavior STD.

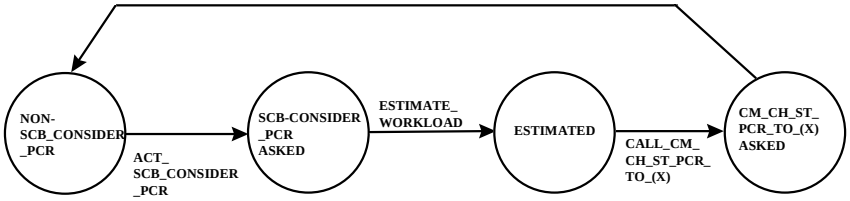


figure 4.119 int-scb_consider_pcr : internal behavior STD

After the operation 'scb_consider_pcr' has started, the scb makes an estimate of the work needed to solve the problem_and_change_report. It then decides to either reject the pcr (call_cm_ch_st_pcr_to_(rejected)) or not (call_cm_ch_st_pcr_to_(carry_out)). All status changes of the pcr are done via the configuration_manager.

4.3.2.7.3 Software_Configuration_Board : manager-STD

The communication between the software_configuration_board's operation (callee) and its caller is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

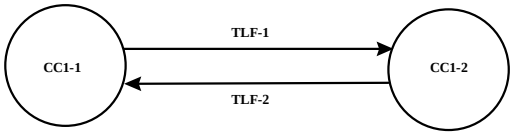


figure 4.120 software_configuration_board : manager STD

The notation CCx in the STD stands for 'Caller(s)-Callee' and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for 'Traps Logical Formula' and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

The CC's and the TLF's for this manager are the combinations for 'scb_consider_pcr' and 'cm_ch_st_pcr_to_(x)' :

CC1-1 = {S1, S3}
TLF-1 = T-1 and T-3

CC1-2 = {S2, S4}
TLF-2 = T-2 and T-4

4.3.2.7.4 Software_Configuration_Board : employee-STDs

The manager STD has the following 2 employee STDs : the own internal callee 'scb_consider_pcr' and its caller 'cm_ch_st_pcr_to_(x)' of the class configuration_manager.

The first employee is the callee operation 'scb_consider_pcr'. This employee has two subprocesses S1 and S2 and two traps T1 and T2 according to the 'act-construct, big second trap'.

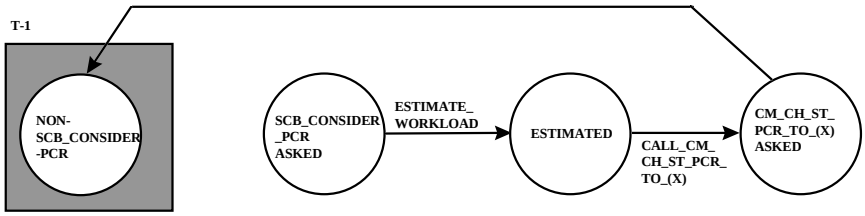


figure 4.121 employee int-scb_consider_pcr : subprocess S1

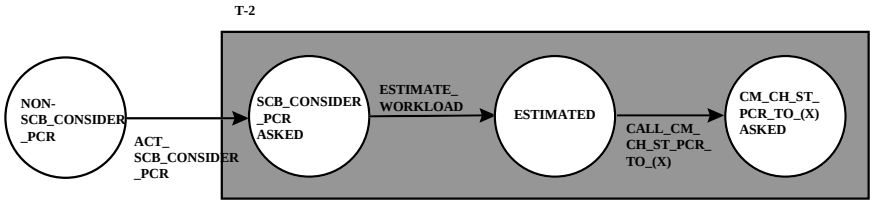


figure 4.122 employee int-scb_consider_pcr : subprocess S2, big trap

The second employee is the caller operation 'cm_ch_st_pcr_to_(x)'. This employee has two subprocesses S3 and S4 and two traps T3 and T4 according to the caller_callee-construct.

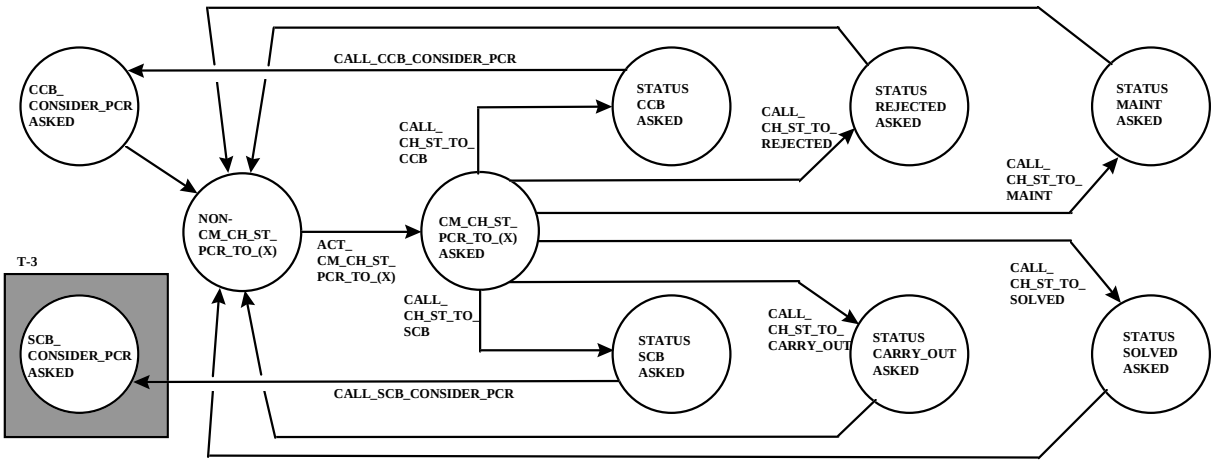


figure 4.123 employee int-cm_ch_st_pcr_to_(x) : subprocess S3

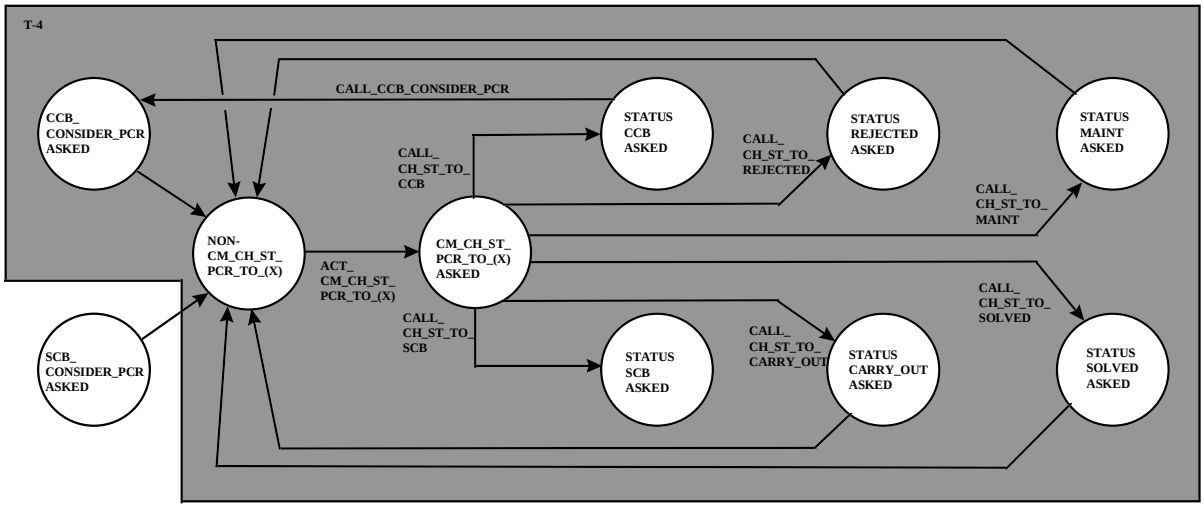


figure 4.124 employee int-cm_ch_st_pcr_to_(x) : subprocess S4

4.3.2.8 Configuration_Control_Board

4.3.2.8.1 Configuration_Control_Board : external behavior-STD

The STD of the external behavior consists of a neutral state in which the configuration_control_board waits for a call to the internal operation and of an activation state in which the internal 'ccb_consider_(x)' operation is started.

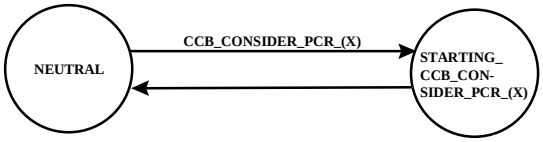


figure 4.125 configuration_control_board : external behavior STD

The formal parameter of the operation is the id of the problem_and_change_report that has to be considered. The configuration_control_board does not wait for the 'ccb_consider' operation to finish, but returns as soon as possible to its neutral state. This increases the level of concurrency. The operation can be called again as well as activated while the current execution is stil taking place.

4.3.2.8.2 Configuration_Control_Board : internal behavior-STDs

The 1 operation of the configuration_control_board has the following internal behavior STD.

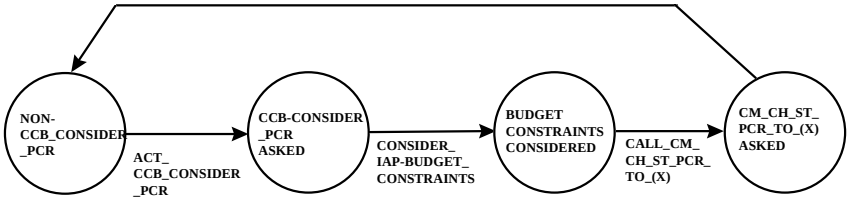


figure 4.126 int-ccb_consider_pcr : internal behavior STD

With the operation 'ccb_consider_pcr', the configuration-control_board checks if the IAP-budget (Internal Automation Projects-budget of the Dutch MoD) is sufficient to handle the problem_and_change_report. If it is, it directs the configuration_manager to present the pcr to the software_configuration_board for further handling (call_cm_ch_st_pcr_to_(scb)). If not, it rejects the pcr (call_cm_ch_st_pcr_to_(rejected)).

4.3.2.8.3 Configuration_Control_Board : manager-STD

The communication between the configuration_control_board's operation (callee) and its caller is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

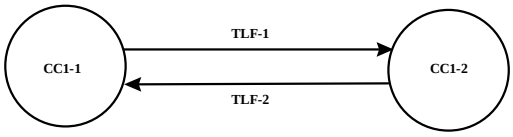


figure 4.127 configuration_control__board : manager STD

The notation CCx in the STD stands for 'Caller(s)-Callee' and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for 'Traps Logical Formula' and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

The CC's and the TLF's for this manager are the combinations for 'ccb_consider_pcr' and 'cm_ch_st_pcr_to_(x)' :

CC1-1 = {S1, S3}
TLF-1 = T-1 and T-3

CC1-2 = {S2, S4}
TLF-2 = T-2 and T-4

4.3.2.8.4 Configuration_Control_Board : employee-STDs

The manager STD has the following 2 employee STDs : the own internal callee 'ccb_consider_pcr' and its caller 'cm_ch_st_pcr_to_(x)' of the class 'configuration_manager'.

The first employee is the callee operation 'ccb_consider_pcr'. This employee has two subprocesses S1 and S2 and two traps T1 and T2 according to the 'act-construct, big second trap'.

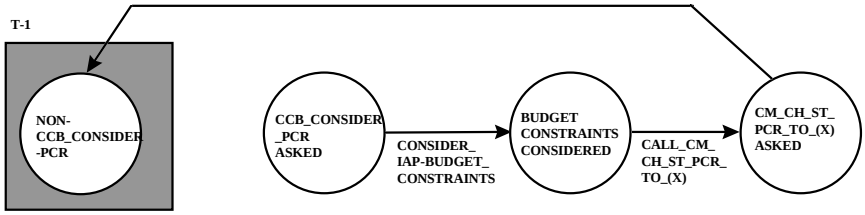


figure 4.128 employee int-ccb_consider_pcr : subprocess S1

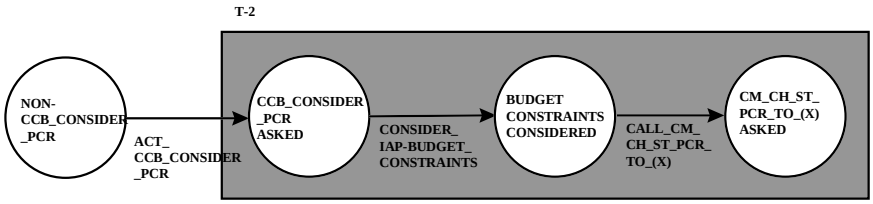


figure 4.129 employee int-ccb_consider_pcr : subprocess S2, big trap

The second employee is the caller operation 'cm_ch_st_pcr_to_(x)'. This employee has two subprocesses S3 and S4 and two traps T3 and T4 according to the caller_callee-construct.

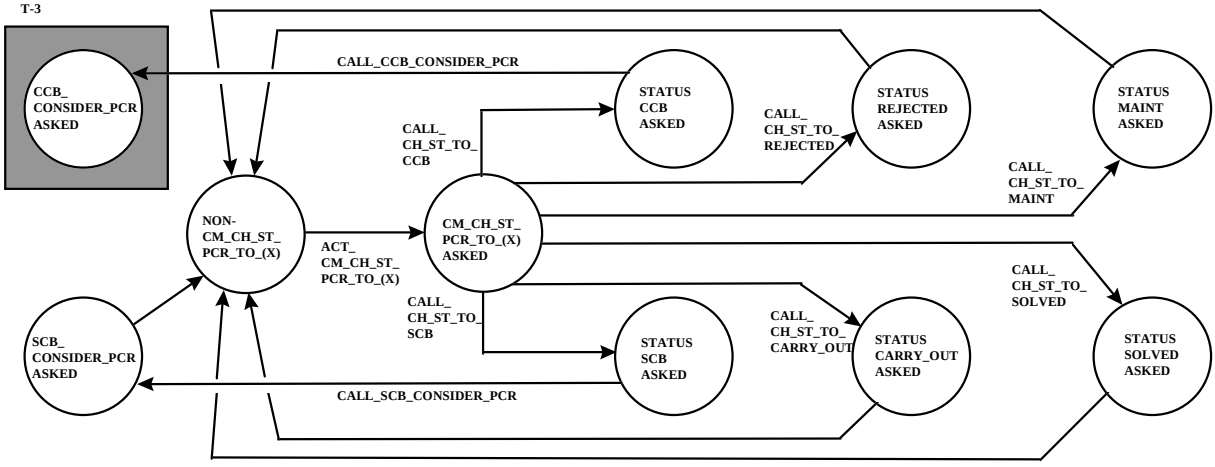


figure 4.130 employee int-cm_ch_st_pcr_to_(x) : subprocess S3

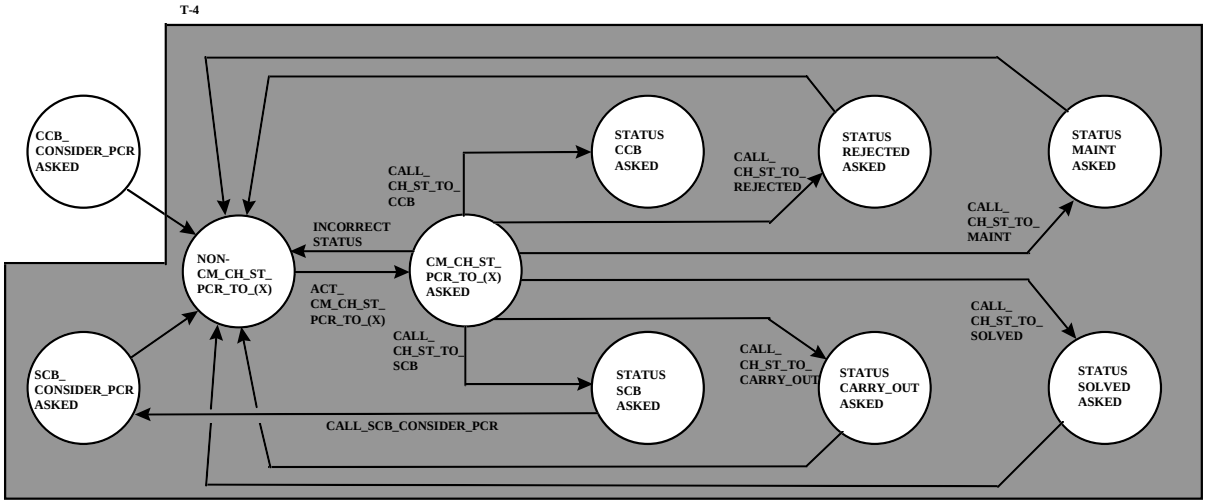


figure 4.131 employee int-cm_ch_st_pcr_to_(x) : subprocess S4

4.3.2.9 Test_Engineer

4.3.2.9.1 Test_Engineer : external behavior-STD

The STD of the external behavior consists of a neutral state in which the test_engineer waits for a call to the only operation it makes available to other objects and of an activation state in which the internal 'te_dt_test' operation is started.

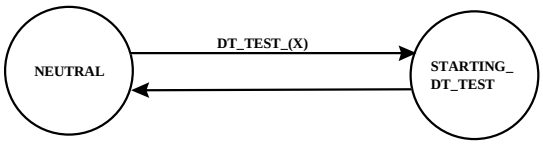


figure 4.132 test_engineer : external behavior STD

The operation takes the id of the configuration_item that has to be tested as its only formal parameter. The test_engineer does not wait until the 'te_dt_test' operation is finished, but returns as soon as possible to its neutral state. The 'te_dt_test' operation can be called again as well as activated while the current execution of 'te_dt_test' is not yet finished.

4.3.2.9.2 Test_Engineer : internal behavior-STDs

The one operation 'te_dt_test' of the test_engineer has the following internal behavior STD.

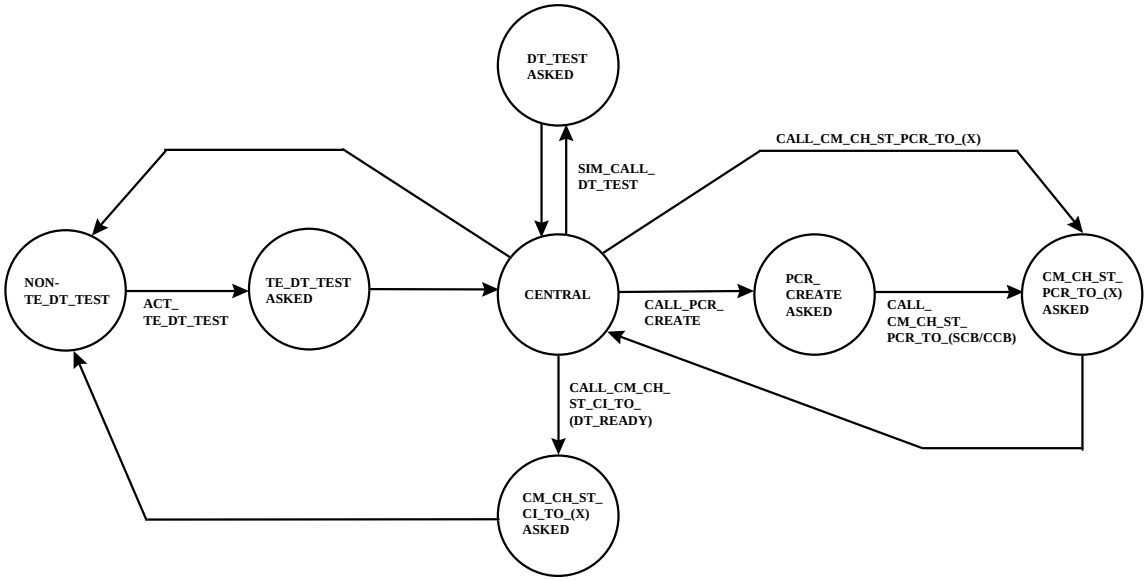


figure 4.133 int-te_dt_test : internal behavior STD

After the operation 'te_dt_test' has started, the test_engineer tests the configuration_item (CI) by executing test_cases. If the test-engineer tests one configuration_item during a test, he executes one call per test_case to the 'dt_test' operation of the CI at the time. If he performs a system level test, he tests many configuration_items at the same time. He must call 'dt_test' simultaneously for all the CIs in the program under test. Both situations are modeled with the 'simultaneous_call'-construct.

If he finds an error he writes a problem_and_change_report (call_pcr_create) which he gives to the configuration manager (call_cm_ch_st_pcr_to_(SCB/CCB)) who will update the status of the problem_and_change_report (PCR). (The configuration_manager will give this PCR either to the software_configuration_board (SCB) or to the configuration_control_board (CCB) for further handling.)

Also during testing the test_engineer checks the (existing) corrected PCRs that apply to the CI(s) under test (if there are any). If a test_case result shows that such a PCR is correctly solved, the test_engineer tells the configuration_manager to change the status of this pcr to ‘solved’ (call_cm_ch_st_pcr_to_(x)).

At the end of the test there are two possibilities. It may be that there are no problems found (no new PCRs were written and all the existing PCRs were solved correctly). The test_engineer then tells the configuration_manager to change the status of the configuration_item to ‘dt_ready’. The test_engineer is now ready with his job and goes to the state ‘non-te_dt_test’.

If the test was not succesfull (PCRs were written or not correctly solved), then the test_engineer takes no status-changing action on the CI. The status of the configuration_item will be changed to ‘maintenance’ later on by the technical_project_manager after he has clustered the pcr’s on this CI. The test_engineer is now ready with his job and goes to the state ‘non-te_dt_test’.

4.3.2.9.3 Test_Engineer : manager-STD

The communication between the test_engineer’s operation (callee) and its caller is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

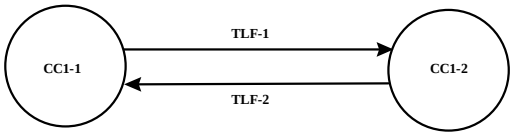


figure 4.134 test_engineer : manager STD

The notation CCx in the STD stands for ‘Caller(s)-Callee’ and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for ‘Traps Logical Formula’ and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

The CC’s and the TLF’s for this manager are the combinations for ‘te_dt_test’ and ‘tpm_dt_test’ :

CC1-1 = {S1, S3}
TLF-1 = T-1 and T-3

CC1-2 = {S2, S4}
TLF-2 = T-2 and T-4

4.3.2.9.4 Test_Engineer : employee-STDs

The manager STD has the following 2 employee STDs : the own internal callee ‘te_dt_test’ and its caller ‘tpm_dt_test’ of the class ‘technical_project_manager’.

The first employee is the callee operation ‘te_dt_test’. This employee has two subprocesses S1 and S2 and two traps T1 and T2 according to the caller_callee-construct.

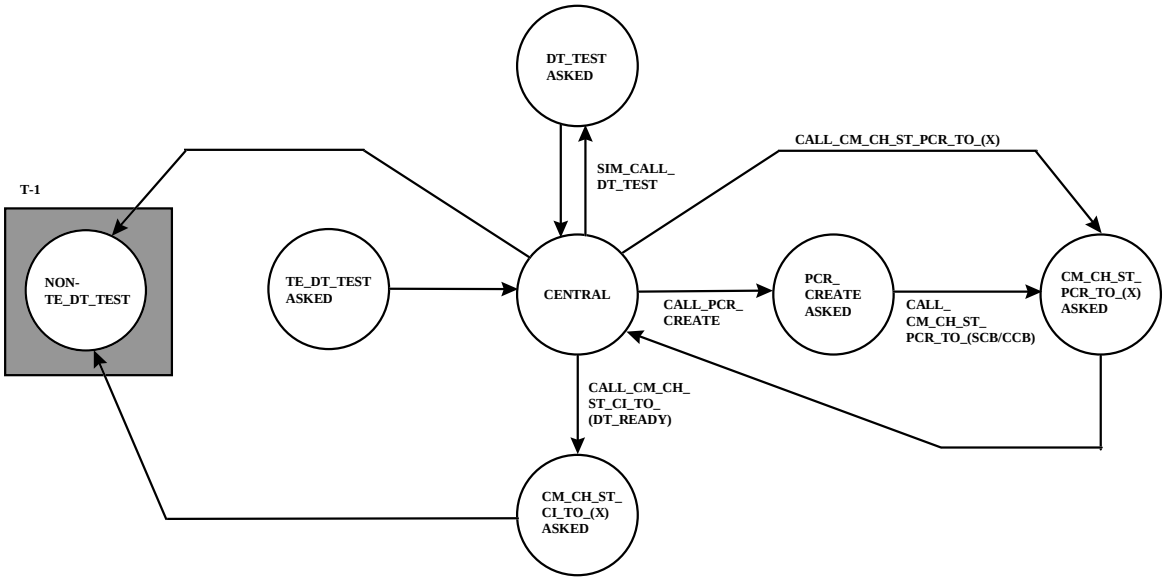


figure 4.135 employee int-te_dt_test : subprocess S1

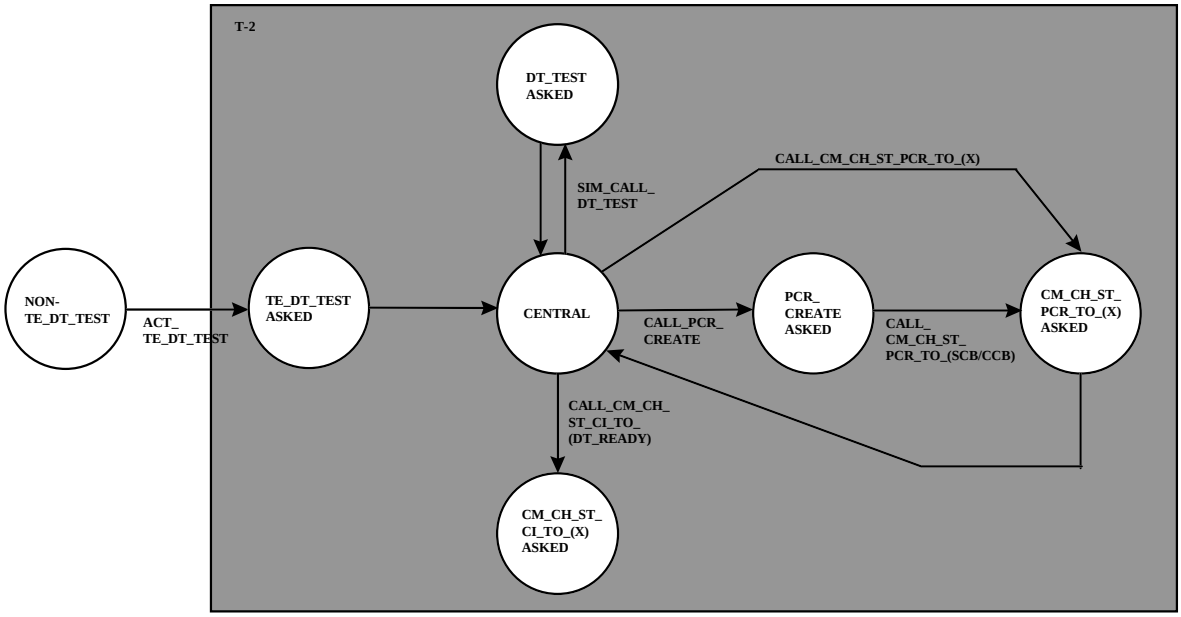


figure 4.136 employee int-te_dt_test : subprocess S2

The second employee is the caller operation ‘tpm_dt_test’. This employee has two subprocesses S3 and S4 and two traps T3 and T4 according to the caller_callee-construct.

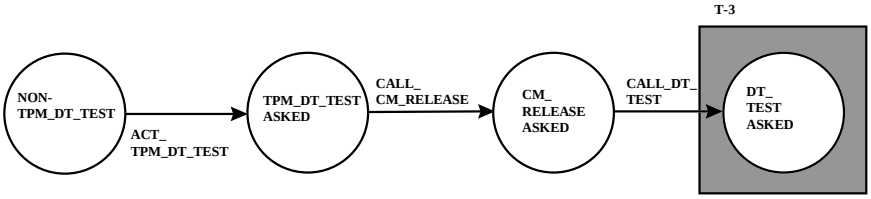


figure 4.137 employee int-dt_test : subprocess S3

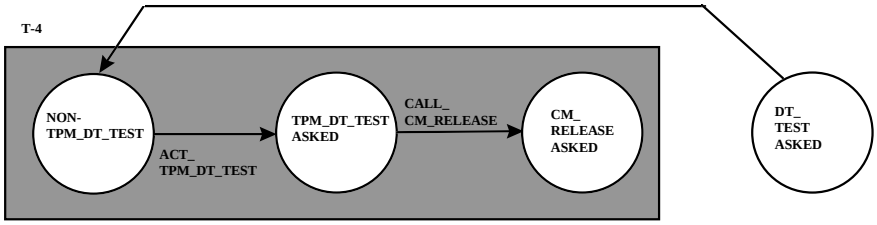


figure 4.138 employee int-dt_test : subprocess S4

4.3.2.10 Customer

4.3.2.10.1 Customer : external behavior-STD

The STD of the external behavior consists of three states. One neutral state in which the customer waits for a call to one of its operations and two activation states in which the internal operations are started.

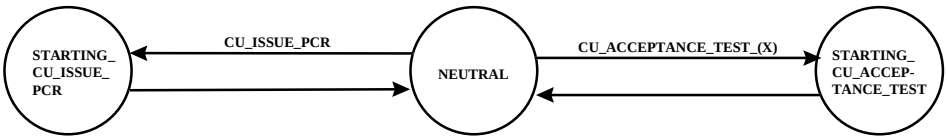


figure 4.139 customer : external behavior STD

As soon as it has started one of its internal operations, the customer returns to the neutral state ready to service another call to its operations. The formal parameter of the operation ‘cu_acceptance_test’ is the id of the configuration_item under test. The customer shows some autonomous behavior by starting his own operation ‘cu_issue_pcr’ without it being called from another object. The customer performs this autonomous behavior when he detects a problem in, or wants a change of, a configuration_item with the status ‘baseline’. (That is a CI of a system that is in operational use.)

4.3.2.10.2 Customer : internal behavior-STD

The 2 operations ‘cu_acceptance_test’ and ‘cu_issue_pcr’ of the customer have the following internal behavior STDs.

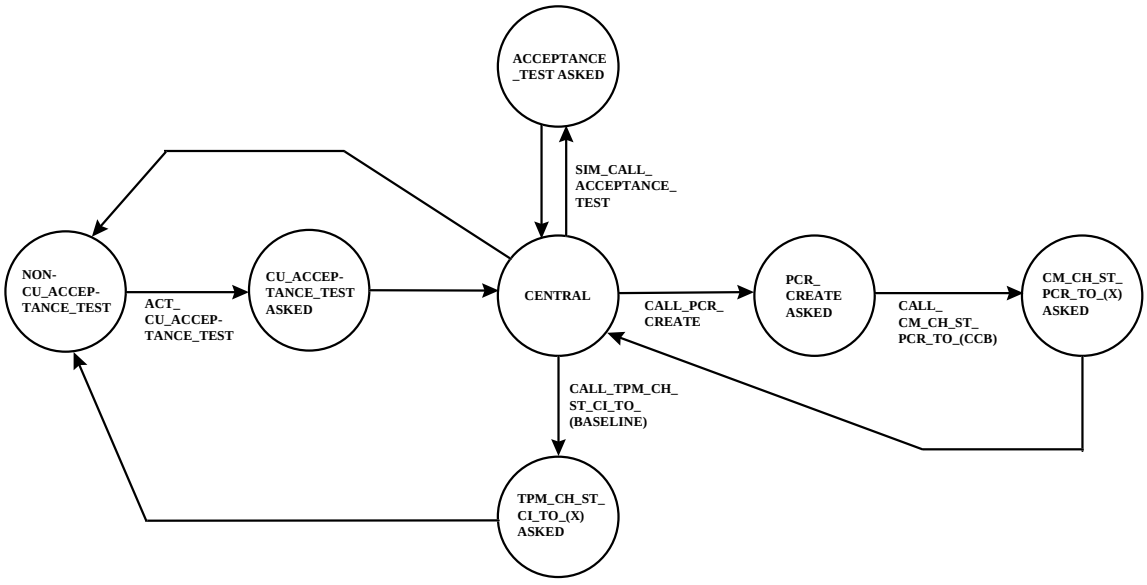


figure 4.140 int-cu_acceptance_test : internal behavior STD

After the operation ‘cu_acceptance_test’ has started, the customer tests the configuration_item by executing test_cases. The customer performs a system level test, he tests many configuration_items at the same time. He must call ‘acceptance_test’ simultaneously for all the ci’s in the program under test. This is modeled with the ‘simultaneous_call’-construct.

If he finds an error he writes an problem_and_change_report which he gives to the configuration_manager. The configuration_manager offers this pcr to the configuration_control_board. If the test has ended and there are no problems found, the customer informs the technical_project_manager of this result by letting him change the status of the configuration_item to ‘baseline’. If the test is not succesfull the customer takes no further action. The status of the configuration_item will be changed to ‘maintenance’ later on by the technical_project_manager after he has clustered the pcr’s on this CI(s).

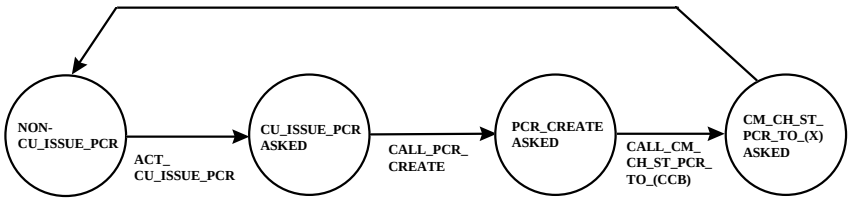


figure 4.141 int-cu_issue_pcr : internal behavior STD

The operation ‘cu_issue_pcr’ is an autonomous operation of the customer. After a system is delivered to the customer, it is in ‘operational’ use. It is now at the location of the customer. The customer uses it. The customer can still find some ‘bugs’ in the system that were not detected during the acceptance test. The customer will issue a problem_and_change_ report on any error he detects. He gives this problem_and_change_report to the configuration_manager. The configuration_manager gives this pcr to the configuration_control_board for evaluation.

Also the customer may find, during operational use of the system, that he wants some changes done. Most of the time a customer finds out only during operational use, if the system really fits his operational or ‘user-interface’ needs. The customer then issues a problem_and_change_report to request the implementation of these changes. He gives this problem_and_change_report to the configuration_manager. The configuration_manager gives this pcr to the configuration_control_board for evaluation.

In both cases (‘errors’ and ‘changes’ during operational use of the system) the customer uses his autonomous operation ‘cu_issue_pcr’.

4.3.2.10.3 Customer : manager-STD

The communication between the customer’s operations (callees) and its callers is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

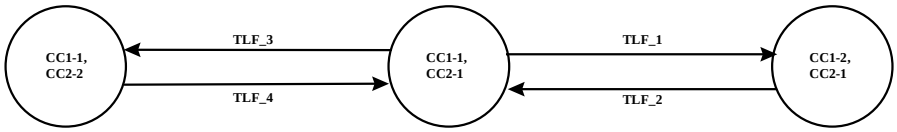


figure 4.142 customer : manager STD

The notation CCx in the STD stands for ‘Caller(s)-Callee’ and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for ‘Traps Logical Formula’ and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

The CC’s and the TLF’s for this manager are the combinations for ‘cu_acceptance_test’ and ‘tpm_ac_test’ and the autonomous operation ‘cu_issue_pcr’ :

CC1-1 = {S1, S3}
TLF-1 = T-1 and T-3

CC1-2 = {S2, S4}
TLF-2 = T-2 and T-4

CC2-1 = {S5}
TLF-3 = T-5

CC2-2 = {S6}
TLF-4 = T-6

4.3.2.10.4 Customer : employee-STDs

The manager STD has the following 3 employee STDs : the own internal callee ‘cu_acceptance_test’ and its caller ‘tpm_ac_test’ of the class ‘technical_project_manager’. And the autonomous operation ‘cu_issue_pcr’.

The first employee is the callee operation ‘cu_acceptance_test’. This employee has two subprocesses S1 and S2 and two traps T1 and T2 according to the caller_callee-construct.

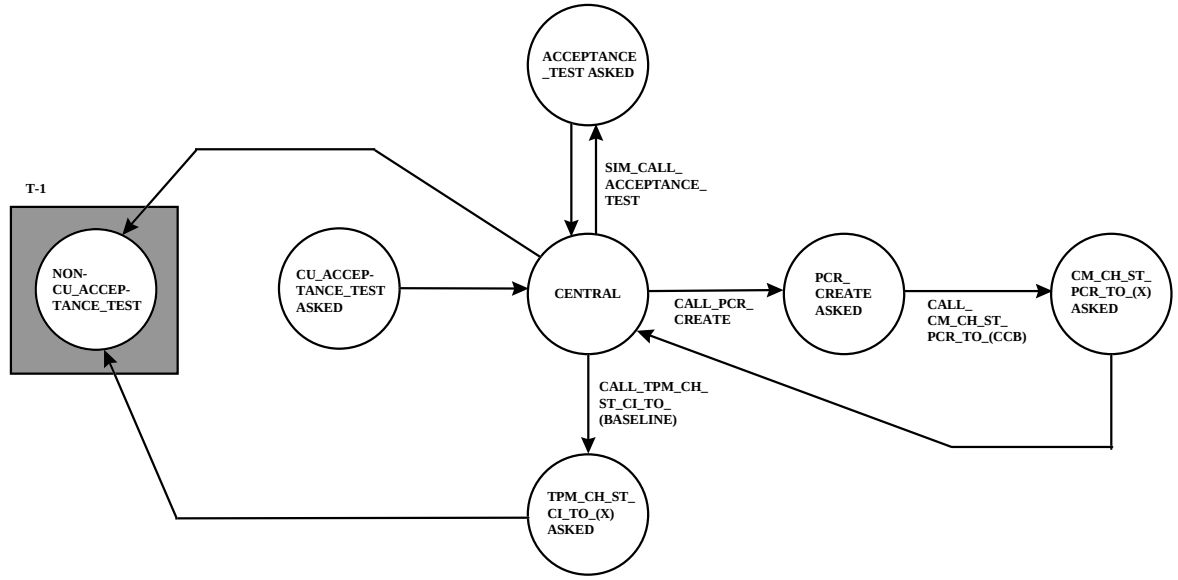


figure 4.143 employee int-cu_acceptance_test : subprocess S1

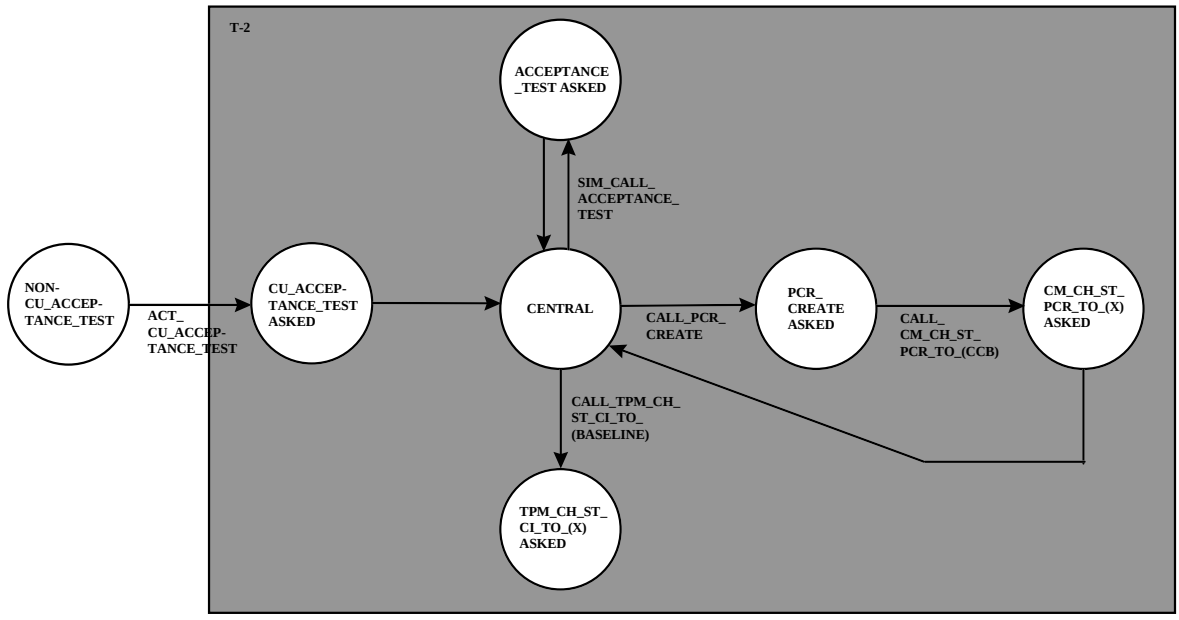


figure 4.144 employee int-cu_acceptance_test : subprocess S2

The second employee is the caller operation ‘tpm_ac_test’. This employee has two subprocesses S3 and S4 and two traps T3 and T4 according to the caller_callee-construct.

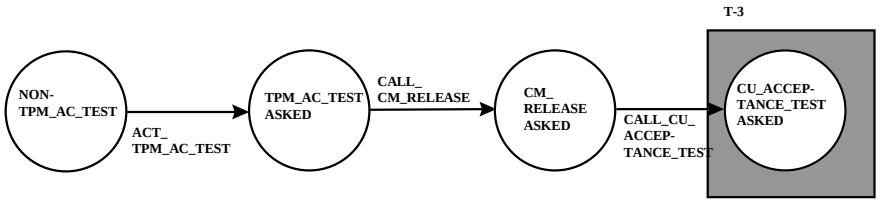


figure 4.145 employee int-tpm_ac_test : subprocess S3

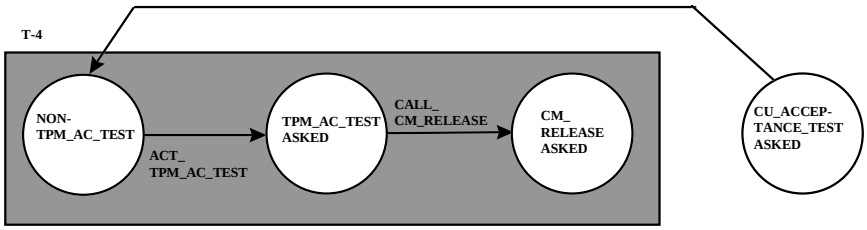


figure 4.146 employee int-tpm_ac_test : subprocess S4

The third employee is the autonomous operation ‘cu_issue_pcr’. This employee has two subprocesses S5 and S6 and two traps T5 and T6 according to the act-construct.

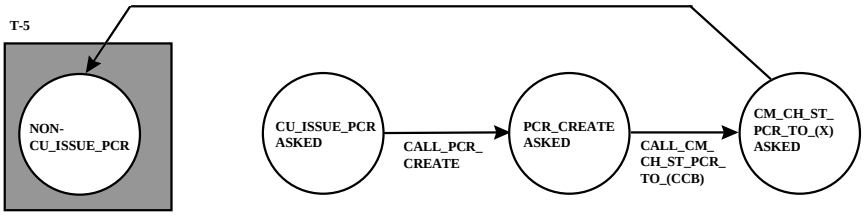


figure 4.147 employee int-cu_issue_pcr : subprocess S5

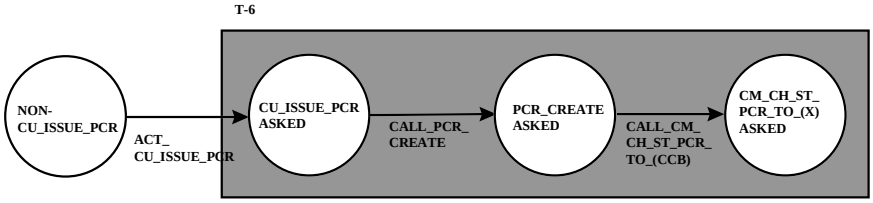


figure 4.148 employee int-cu_issue_pcr : subprocess S6

4.3.2.11 Release_Note

4.3.2.11.1 Release_Note : external behavior-STD

The STD of the external behavior consists of a neutral state in which the release_note waits for a call to the only operation it makes available to other objects and of an activation state in which the internal 'rn_create_(x)' operation is started.

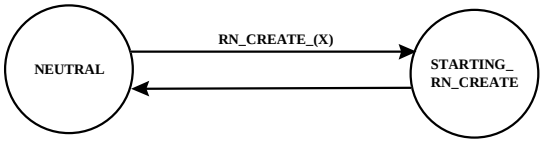


figure 4.149 release_note : external behavior STD

The release_note does not wait until the 'rn_create_(x)' operation is finished, but returns as soon as possible to its neutral state.

4.3.2.11.2 Release_Note : internal behavior-STDs

The release_note has 1 operation : 'rn_create_(x)' with the following internal behavior STDs.

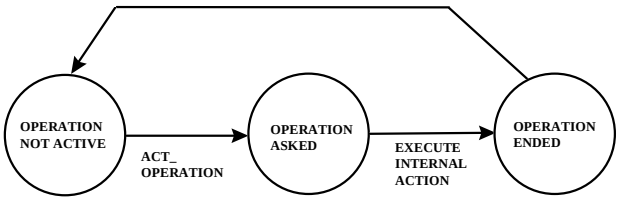


figure 4.150 only_internal_action-construct : internal behavior STD

The operation can be modeled by the 'only_internal_action-construct' since its interaction with other objects is not shown. The internal action consists of creating the release_note object and modifying its 'date' and 'baseline' attributes thereby instantiating the 'baselines'-association, which indicates the CI(s) covered by this release_note.
The operation has one formal parameter. The value of this parameter is put in the 'content'-attribute of the created release_note object.

4.3.2.11.3 Release_Note : manager-STD

The communication between the release_note's operation (callee) and its caller is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

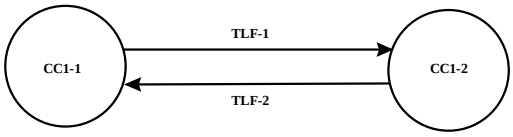


figure 4.151 release_note : manager STD

The notation CCx in the STD stands for 'Caller(s)-Callee' and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for 'Traps Logical Formula' and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

The CC's and the TLF's for this manager are the combinations for 'rn_create_(x)' and 'cm_release_note' :

CC1-1 = {S1, S3}
TLF-1 = T-1 and T-3

CC1-2 = {S2, S4}
TLF-2 = T-2 and T-4

4.3.2.11.4 Release_Note : employee-STDs

The manager STD has the following 2 employee STDs : the own internal callee 'rn_create_(x)' and its caller 'cm_release_note' of the class 'configuration_manager'.

The first employee is the callee operation 'rn_create_(x)' (shown as an 'only_internal_action'-template STD). This employee has two subprocesses S1 and S2 and two traps T1 and T2 according to the caller_callee-construct.

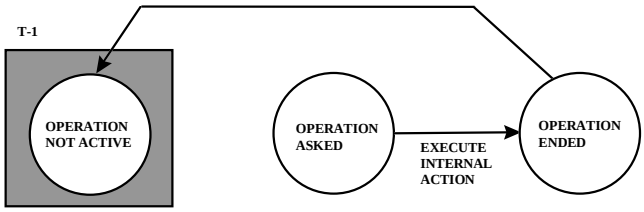


figure 4.152 employee int-rn_create_(x) : subprocess S1

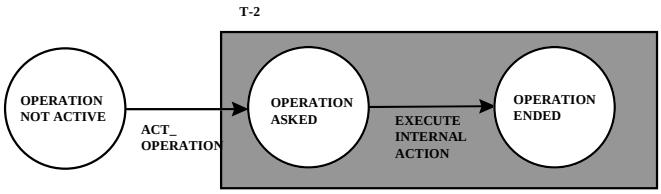


figure 4.153 employee int-rn_create_(x) : subprocess S2

The second employee is the caller operation 'cm_release_note'. This employee has two subprocesses S3 and S4 and two traps T3 and T4 according to the caller_callee-construct.

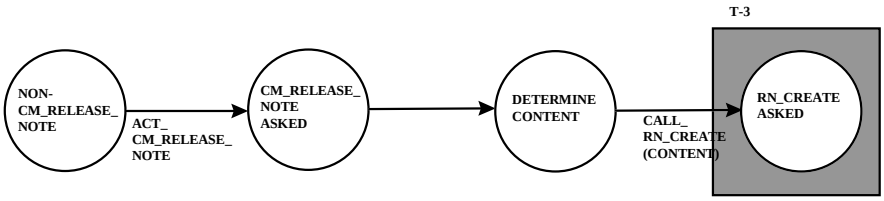


figure 4.154 employee int-cm_release_note : subprocess S3

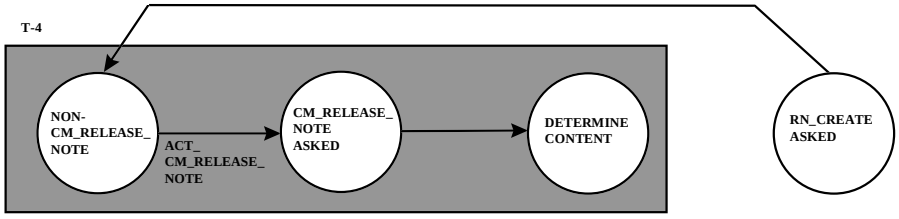


figure 4.155 employee int-cm_release_note : subprocess S4

SOFTWARE PROCESS MODELING in SOCCA

student : H.G.Brugman
studentnr : 9105506
version : 0.10

5. Key Process Area 'Software Project Planning'

5.1 Introduction

In this chapter part of the 'Software Project Planning'-process of the software development organization 'Waco Business Unit' (WBU) of the Dutch Ministry of Defense is modeled using the SOCCA process modeling language.

The 'Software Project Planning'-process consists of three process fragments : 'writing project management documents', 'closing project' and 'changing project mangement documents'.

Due to time limitations only the SOCCA model of the process fragment 'writing project management documents' will be given in this thesis.

To look into the question of the 'scaleability' of SOCCA (i.e can a larger SOCCA model be constructed from separate smaller sub-models), the modeling of the process fragment 'writing project management documents' is done by splitting the process fragment into four smaller process fragments. These smaller process fragments are modeled indepently of each other. Next the sub-models of the four smaller process fragments are integrated into one SOCCA model of the process fragment 'writing project management documents'.

The four smaller process fragments (phases) are : 'phase 1, recognizing customer requirements', 'phase 2, writing and consultation', 'phase 3, approval' and 'phase 4, resource allocation'. The SOCCA sub-models of these four 'phases' are described in paragraph 5.3. First the class diagrams are given and then the state transition diagrams.

Paragraph 5.3.1 describes the 'class diagrams' that are valid for all four 'phases'. Paragraph 5.3.2 describes the 'import-export'-diagram of phase 1. Paragraph 5.3.3 describes the 'import-export'-diagram of phase 2. Paragraph 5.3.4 describes the 'import-export'-diagram of phase 3. Paragraph 5.3.5 describes the 'import-export'-diagram of phase 4.

Paragraph 5.3.6 describes the 'state transition diagrams' of phase 1. Paragraph 5.3.7 describes the 'state transition diagrams' of phase 2. Paragraph 5.3.8 describes the 'state transition diagrams' of phase 3. Paragraph 5.3.9 describes the 'state transition diagrams' of phase 4.

The integration of the four sub-models into one SOCCA model of the process fragment 'writing project management documents' is described in chapter 6.

Also the usefullness of a SOCCA model as a process description is investigated in this chapter. This is done by checking if the SOCCA model of the process fragment 'writing project management documents' of the 'Software Project Planning'-process can be used as input for a process audit. As audit method is chosen the 'Capability Maturity Model'-assessment. The process audit is described in paragraph 5.2.

5.2 CMM Assessment

In this paragraph the usefullness of a SOCCA model as a process description is investigated. This is done by checking the implementation of the CMM 'Key Practices' by the WBU organization while using the SOCCA model of the process fragment 'writing project management documents' of the 'Software Project Planning'-process as a reference instead of the real process.

The purpose of Software Project Planning (SPP) is to establish reasonable plans for performing the software engineering and for managing the software project [CMM].

Software Project Planning involves developing estimates for the work to be performed, establishing the necessary commitments, and defining the plan to perform the work.

The software planning begins with a statement of the work to be performed and other constraints and goals that define and bound the software project. The software planning process includes steps to estimate the size of the software work products and the resources needed, produce a schedule, identify and assess software risks, and negotiate commitments.

Iterating through these steps may be necessary to establish the plan for the software project (i.e., the software development plan). This plan provides the basis for performing and managing the software project's activities and addresses the commitments to the software project's customer according to the resources, constraints, and capabilities of the software project.

The Capability Maturity Model (CMM) states the following goals for this Key Process Area (KPA) :

Goal 1 : Software estimates are documented for use in planning and tracking the software project
Goal 2 : Software project activities and commitments are planned and documented
Goal 3 : Affected groups and individuals agree to their commitments related to the software project

Key Process Areas in CMM are divided into Key Practices (KP). A Key Practice describes at the lowest level 'what' has to be done, but not 'how' it should be done. For a correct fulfillment of the goals of the KPA all its Key Practices have to be satisfied by the software development organization.

Key Practices are organized into five groups (processes). A group (process) is called 'common features' in CMM. The five groups are : 'Commitment to perform', 'Ability to perform', 'Activities performed', 'Measurement and analysis' and 'Verifying implementation'.

1. 'Commitment to perform' describes the actions an organization must take to ensure that the SPP process is established and will endure. Typically this involves the codifying of organizational policies (in manuals) and senior management commitment to this policies. This codifying process is not modeled in this thesis. The result of this process is the 'Manual for Technical Project Management' now in use in the Waco Business Unit (WBU) organization.

2. 'Ability to perform' describes the preconditions that must exists in an organization to implement the SPP process correctly. Typically this involves resources, organizational structures and training. The organizational structure as applied to the SPP process is modeled by the class diagrams of the next chapter.

3. 'Activities performed' describes the roles and procedures necessary to implement the Key Process Area. Typically this involves establishing plans and procedures, performing the work, tracking it and taking corrective actions as necessary. The SPP process is modeled by the state transition diagrams of the next chapter.

4. 'Measurement and analysis' describes the need to measure the SPP process and analyzes the measurements. Typically this involves measurements that could be taken to determine the status and effectiveness of the 'Activities performed'. This process is not implemented in the Waco Business Unit (WBU).

5. 'Verifying implementation' describes the steps to ensure that the activities are performed in compliance with the process that has been established. Typically this involves reviews and audits by management and software quality assurance. The reviewing and auditing of the software project planning is modeled by the state transition diagrams of the next chapter.

The SOCCA model of the process fragment 'writing project management documents' of the Software Project Planning-process as given in the next paragraph, models the organizational structure with class diagrams. It models the implemented SPP-process with state transition diagrams.

The following table lists all the Key Practices of this KPA in the left column. In the right column is indicated if and how the Waco Business Unit (WBU) has satisfied a particular Key Practice.

Commitment to perform

no	**Key Practice**	**WBU implementation**
1	A software manager is designated to be responsible for negotiating commitments and developing the project's software development plan	The procedure is documented by the behavior part (STDs) of the SOCCA model of this KPA. In phase 1 a 'technical project manager' is tasked by the 'make or buy meeting' to write a proposal, which includes writing the software development plan.
2	The project follows a written organizational policy for planning a software project	The SPP process is described in the 'Manual for Technical Project Management'

Ability to perform

no	Key Practice	WBU implementation
1	A documented and approved statement of work exists for the software project	See the class diagram of the SOCCA model of this KPA. The class 'project contract' together with the class 'software development plan' constitute the 'statement of work'. Both documents are agreed on by the WBU management and the customer.
2	Responsibilities for developing the software development plan are assigned	The procedure is documented by the behavior part (STDs) of the SOCCA model of this KPA. In phase 1 a 'technical project manager' is tasked by the 'make or buy meeting' to write a proposal, which includes writing the software development plan
3	Adequate resources and funding are provided for planning the software project	Refer to the class diagram and the STDs of the SOCCA model of this KPA. - time for writing a proposal is allocated. See attribute 'allocated hours' of the class 'project form'. - the responsible technical project manager is aided by a second estimator - an automated project planning tool is available for the technical project manager
4	The software managers, software engineers, and other individuals involved in the software project planning are trained in the software estimating and planning procedures applicable to their areas of responsibility.	- Technical project managers can follow the in-house training course 'Function Point Analysis' - Technical project managers can follow the in-house trainig course 'project planning-tool'

Activities performed

no	Key Practice	WBU implementation
1	The software engineering group participate on the project proposal team	Not implemented. There exists no software engineering group within the WBU organization. Consequently it does not appear in the SOCCA model of this KPA.
2	Software project planning is initiated in the early stages of, and in parallel with, the overall project planning.	Software project planning is an integral part of the overall project planning. They come into being at the same time.
3	The software engineering group participates with other affected groups in the overall project planning throughout the project's life.	Not implemented. There exists no software engineering group within the WBU organization.
4	Software project commitments made to individual groups external to the organization are reviewed with senior management according to a documented procedure.	All 'project contracts' are reviewed by senior management in the 'project meeting minus'. This procedure is documented by the STDs of the SOCCA model of this KPA.
5	A software life cycle with predefined stages of manageable size is identified or defined.	See the sub-models of phase 1 until 4 of the SOCCA model of this KPA. A project has a life cycle consisting of the stages 'writing project management documents', 'changing project management documents' and closing project'. The stage 'writing project management documents' is subdivided in the phases 'recognizing customer requirements', 'writing and consultation', 'approval' and 'resource allocation'.
6	The project's software development plan is	The procedure is documented by the behavior part

no	Key Practice	WBU implementation
	developed according to a documented procedure.	(STDs) of the SOCCA model of this KPA. - This procedure is documented by the operation 'tpm_write_proposal_(x)' of the class 'technical project manager' - the software development plan itself is written by tailoring a template document
7	The plan for the software project is documented	Refer to the class diagram of the SOCCA model of this KPA. A particular software development plan is an object of the class 'software development plan'. This class has an attribute 'content', the value of which is the text of the plan.
8	Software work products that are needed to establish and maintain control of the software project are identified.	The Software Configuration Management plan is part of the software development plan. See the sub-attribute 'SCM-plan' of the class 'software development plan' of the SOCCA model of this KPA. (This is also the implementation of Key Practice 'Activities performed' no 1 and 2, of the KPA 'Software Configuration Management'.)
9	Estimates for the size of the software work product (or change of the size of software work product) are derived according to a documented procedure.	The procedure is documented by the behavior part (STDs) of the SOCCA model of this KPA. -The method 'tpm-write_proposal_(x)' of the class 'technical project manager' in phase 2 documents the procedure which is followed in estimating the size of a work product. This includes the use of a second estimator. - The estimating technique that is used is 'Function Point Analysis'
10	Estimates for the software project's effort and costs are derived according to a documented procedure.	- Their is no historical database (yet) within the WBU to relate the size of the software work product in 'Function Points' to the required effort in 'person-hours'. - Also their is no procedure within the WBU to allocate 'overhead' to a project. - The WBU is an internal software department of the Dutch MoD and it is allocated a yearly budget in 'person-hours'.
11	Estimates for the project's critical computer resources are derived according to a documented procedure	Not implemented
12	The project's software schedule is derived according to a documented procedure.	The 'Planning' (in the form of a Gantt chart = network planning) is part of the software development plan. See the sub-attribute 'Planning' of the class 'software development plan' of the SOCCA model of this KPA. As such, the 'Planning' of a project is derived and reviewed along with the 'software development plan' via the procedure as documented by the behavior part (STDs) of the SOCCA model of this KPA. The 'Planning' is derived with an automated 'project planning'-tool.
13	The software risks associated with the cost,	The 'Risk Analysis'-result is part of the software

no	Key Practice	WBU implementation
	resource, schedule, and technical aspects of the project are identified, assessed, and documented.	development plan. See the sub-sub-attribute 'Risk Analysis' of the class 'software development plan' of the class diagram of the SOCCA model of this KPA. - an automated 'Risk Analysis'-tool is used by the technical project manager to assess the risk of the project.
14	Plans for the project's software engineering facilities and support tools are prepared	The software engineering facilities and support tools for the project are planned for via the 'Internal Resources Allocation'-document. See the class diagram of the SOCCA model of this KPA. The preparation of this 'IRA"-document is modeled in phase 2 with the operation 'tpm_write _proj_man_doc_(x)' of the class 'technical project manager'.
15	Software planning data are recorded	The 'Planning' (in the form of a Gantt chart = network planning) is part of the software development plan. See the sub-attribute 'Planning' of the class 'software development plan' of the SOCCA model of this KPA.

Measurement and analysis

no	Key Practice	WBU implementation
1	Measurements are made and used to determine the status of the software planning activities	not implemented

Verifying implementation

no	Key Practice	WBU implementation
1	The activities for software project planning are reviewed with senior management on a periodic basis	not implemented
2	The activities for software project planning are reviewed with the project manager on both a periodic and event-driven basis	The 'Planning' is part of the 'software development plan (SDP)'. See the sub-attribute 'Planning' of the class 'software development plan' of the class diagram of the SOCCA model of this KPA. As such, the 'Planning' activities of a project are reviewed by the 'head production section' of the 'technical project manager'. This review procedure is documented by the behavior part (STDs) of the SOCCA model of this KPA (see operation 'tpm_write_proposal_(x)') of Phase 2.
3	The software quality assurance group reviews and /or audits the activities and work products for software project planning and reports the results	The 'Planning' is part of the 'software development plan (SDP)'. See the sub-attribute 'Planning' of the class 'software development plan' of the class diagram of the SOCCA model of this KPA. As such, the 'Planning' work product of a project is audited by the 'quality assurance adviser (QAA)'. This auditing by the QAA is documented by the behavior part (STDs) of the SOCCA model of this KPA (see operation 'tpm_write

no	**Key Practice**	**WBU implementation**
		proposal(x)') of Phase 2.

5.3 SOCCA model of process fragment 'writing project management documents'

The SOCCA model depicts the process fragment 'writing project management documents' of the 'Software Project Planning (SPP)'-process that is in use within the Waco Business Unit (WBU). This process is described in the 'Manual for Technical Project Management', chapter 1 'Project management', section 1 : 'Procedure writing project management documents' [MTP].

To look into the question of the 'scaleability' of SOCCA (i.e can a larger SOCCA model be constructed from separate smaller sub-models), the modeling of the process fragment 'writing project management documents' is done by splitting the process fragment into four smaller process fragments. These smaller process fragments are modeled indepently of each other. In chapter 6 these sub-models of the four smaller process fragments are then integrated into one SOCCA model of the process fragment 'writing project management documents'.

The four smaller process fragments (phases) are : 'phase 1, recognizing customer requirements', 'phase 2, writing and consultation', 'phase 3, approval' and 'phase 4, resource allocation'. First the class diagrams are given and then the state transition diagrams.

Paragraph 5.3.1 describes the 'class diagrams' that are valid for all four 'phases'. Paragraph 5.3.2 describes the 'import-export'-diagram of phase 1. Paragraph 5.3.3 describes the 'import-export'-diagram of phase 2. Paragraph 5.3.4 describes the 'import-export'-diagram of phase 3. Paragraph 5.3.5 describes the 'import-export'-diagram of phase 4.

Paragraph 5.3.6 describes the 'state transition diagrams' of phase 1. Paragraph 5.3.7 describes the 'state transition diagrams' of phase 2. Paragraph 5.3.8 describes the 'state transition diagrams' of phase 3. Paragraph 5.3.9 describes the 'state transition diagrams' of phase 4.

5.3.1 Class Diagrams

This paragraph describes the 'class diagrams' that are valid for all four 'phases'. That is to say all the SOCCA class diagrams minus the import-export diagrams. The import-export diagrams are given seperately, per phase, in the paragraphs following this paragraph.

The data perspective of the four phases of the process fragment 'writing project management documents' is modeled with class diagrams. The class diagrams in this paragraph show the classes and subclasses with their operations, attributes and associations (aggregation associations and the general associations) as they pertain to all four phases.

The figures below show the classes and subclasses (via generalization/ specialization associations) and the aggregation associations between them. The notation convention of 'superclass/subclass', 'aggregation association' and 'the multiplicity of an association (cardinality ratio constraint)' in SOCCA is the same as in UML [UML], [FOW].

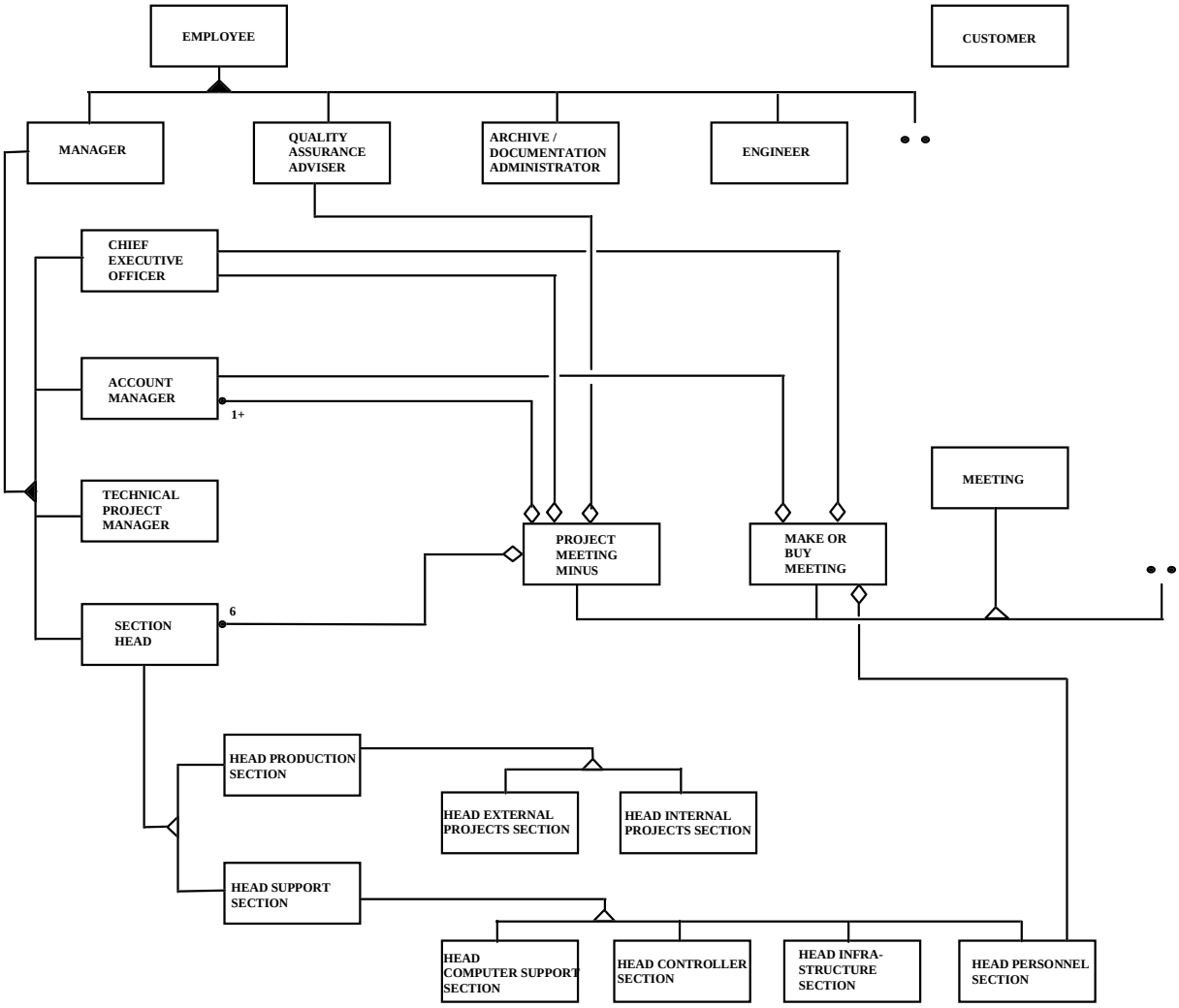


figure 5.1 Class diagram: classes, subclasses and aggregation associations (part 1)

The class diagram showing the classes, subclasses and aggregation associations is divided into two parts. The first part (figure above) shows the employees, meetings and customer involved in the software project planning process. The class EMPLOYEE has the disjoint subclasses MANAGER, QUALITY ASSURANCE ADVISER, ARCHIVE / DOCUMENTATION ADMINISTRATOR and ENGINEER (among others, i.e. these other subclasses are not relevant for the SPP process. They are indicated by the 'ellipsis', the double dot).

The classes CHIEF EXECUTIVE OFFICER, ACCOUNT MANAGER, TECHNICAL_PROJECT_MANAGER and SECTION HEAD are overlapping subclasses of the superclass MANAGER. The class SECTION HEAD is specialized into two disjoint subclasses HEAD PRODUCTION SECTION and HEAD SUPPORT SECTION.

A HEAD PRODUCTION SECTION is either a HEAD EXTERNAL PROJECTS SECTION or a HEAD INTERNAL PROJECTS SECTION.

A HEAD SUPPORT SECTION is either a HEAD COMPUTER SUPPORT SECTION, a HEAD CONTROLLER SECTION, a HEAD INFRASTRUCTURE SECTION or a HEAD PERSONNEL SECTION.

The class MEETING is has the classes PROJECT MEETING MINUS and MAKE OR BUY MEETING as its subclasses (among others, i.e. these other subclasses are not relevant for the SPP process. They are indicated by the 'ellipsis', the double dot).

The MAKE OR BUY MEETING is an aggregation af the employees involved and consists of the (1) CHIEF EXECUTIVE OFFICER, one ACCOUNT MANAGER and the (1) HEAD PERSONNEL SECTION.

The PROJECT MEETING MINUS is an aggregation af the employees involved and consists of the (1) CHIEF EXECUTIVE OFFICER, one or more ACCOUNT MANAGERs, the (1) QUALITY ASSURANCE ADVISER and all (6) SECTION HEADs.

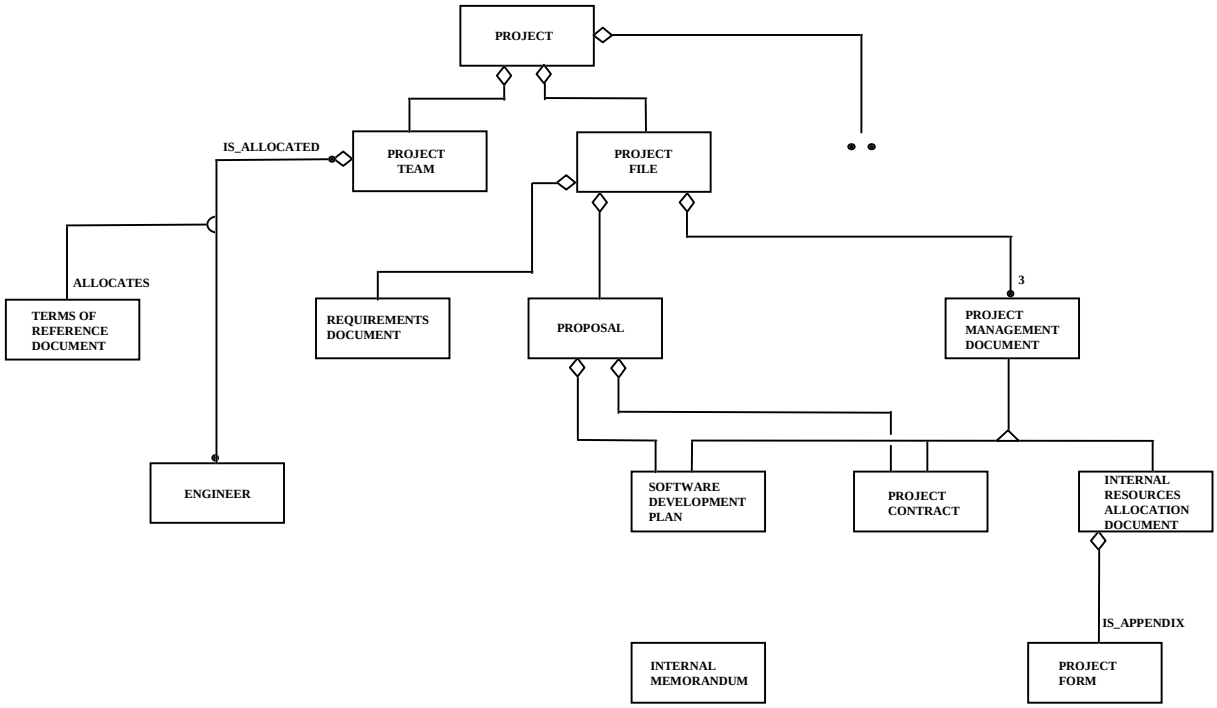


figure 5.2 Class diagram : classes, subclasses and aggregation associations (part 2)

As already mentioned, the class diagram showing the classes, subclasses and aggregation associations is divided into two parts. The second part (figure above) shows the class PROJECT. The class PROJECT is an aggregation of the classes PROJECT FILE and PROJECT TEAM (among others, i.e. these other subclasses are not relevant for the SPP process. They are indicated by the 'ellipsis', the double dot).

The class PROJECT TEAM consists of ENGINEERs who are allocated to the project via a TERMS OF REFERENCE DOCUMENT. N.B. the association IS_ALLOCATED is modeled here as a class. This is indicated in the diagram by the semi-circle symbol connecting the class TERMS OF REFERENCE DOCUMENT with the association IS_ALLOCATED.

The class PROJECT FILE is an aggregation of classes REQUIREMENTS DOCUMENT (1x), PROPOSAL (1x) and PROJECT MANAGEMENT DOCUMENT (3x).

A PROPOSAL consists of a SOFTWARE DEVELOPMENT PLAN (1x) and a PROJECT CONTRACT (1x). In fact the preliminary (unsigned) version of the PROJECT CONTRACT with the SOFTWARE DEVELOPMENT PLAN as appendix is called a PROPOSAL.

A PROJECT MANAGEMENT DOCUMENT is either a SOFTWARE DEVELOPMENT PLAN, a PROJECT CONTRACT or an INTERNAL RESOURCES ALLOCATION DOCUMENT.

The INTERNAL RESOURCES ALLOCATION DOCUMENT has a PROJECT FORM as an appendix.

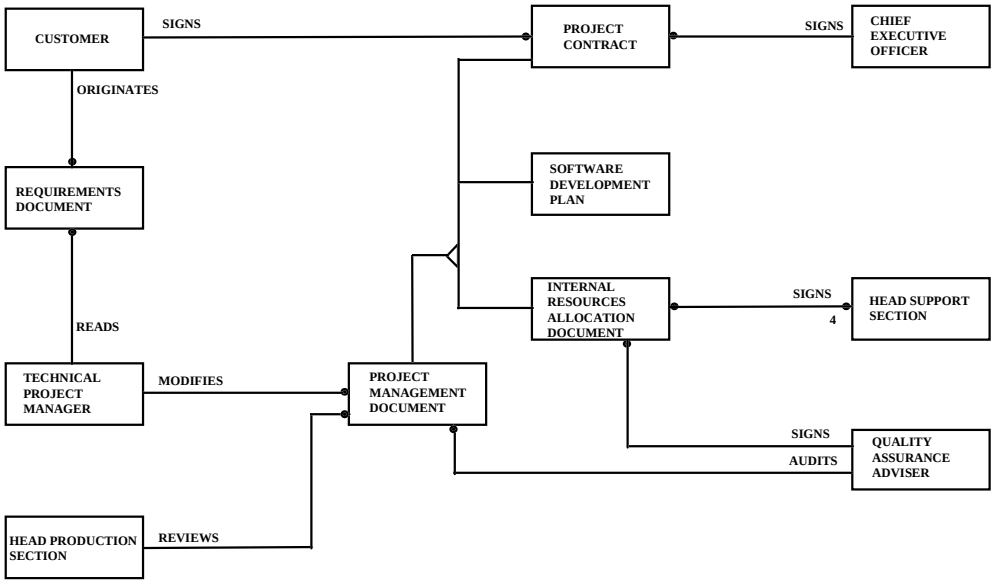


figure 5.3 Class diagram : classes, subclasses and general associations

The figure above shows with the general associations between the classes and subclasses. The notation convention of '(general) association' is the same in SOCCA as it is in UML with the exception of the notation convention of 'constraint between associations' (see the explanation in the 'class diagrams'-paragraphs of the chapter 'Key Process Area 'Software Configuration Management').

The names and the multiplicity of the general associations are also given in the diagram. An association is inherently bidirectional. I.e it can be traversed in both directions. The name of a binary association reads in a particular direction. This direction is called the forward direction. The opposite direction is called the inverse direction. When an association is traversed in the inverse direction, the association name is 'inverted'. E.g. MODIFIES becomes IS_MODIFIED. The class diagram only shows one name per association.

A customer ORIGINATES zero or more requirements documents. An requirements document IS_ORIGINATED by one customer. A technical project manager READS zero or more requirements documents. An requirements document IS_READ by one technical project manager.

A technical project manager MODIFIES zero or more project management documents. A project management document IS_MODIFIED by one technical project manager. A head of a production section REVIEWS zero or more project management documents. A project management document IS_REVIEWED by one production section head.

A customer SIGNS zero or more project contracts. The chief executive officer SIGNS zero or more project contracts. A projects contract IS_SIGNED by one customer and by one (the) chief executive officer.

A head of a support section SIGNS zero or more 'internal resources allocation' documents. The quality assurance adviser SIGNS zero or more 'internal resouces allocation' documents. An 'internal resources allocation' document IS_SIGNED by 4 support section heads (there are 4 support sections) and one (the) quality assurance adviser.

The quality assurance adviser AUDITS zero or more project management documents. A project management document IS_AUDITED by one (the) quality assurance adviser.

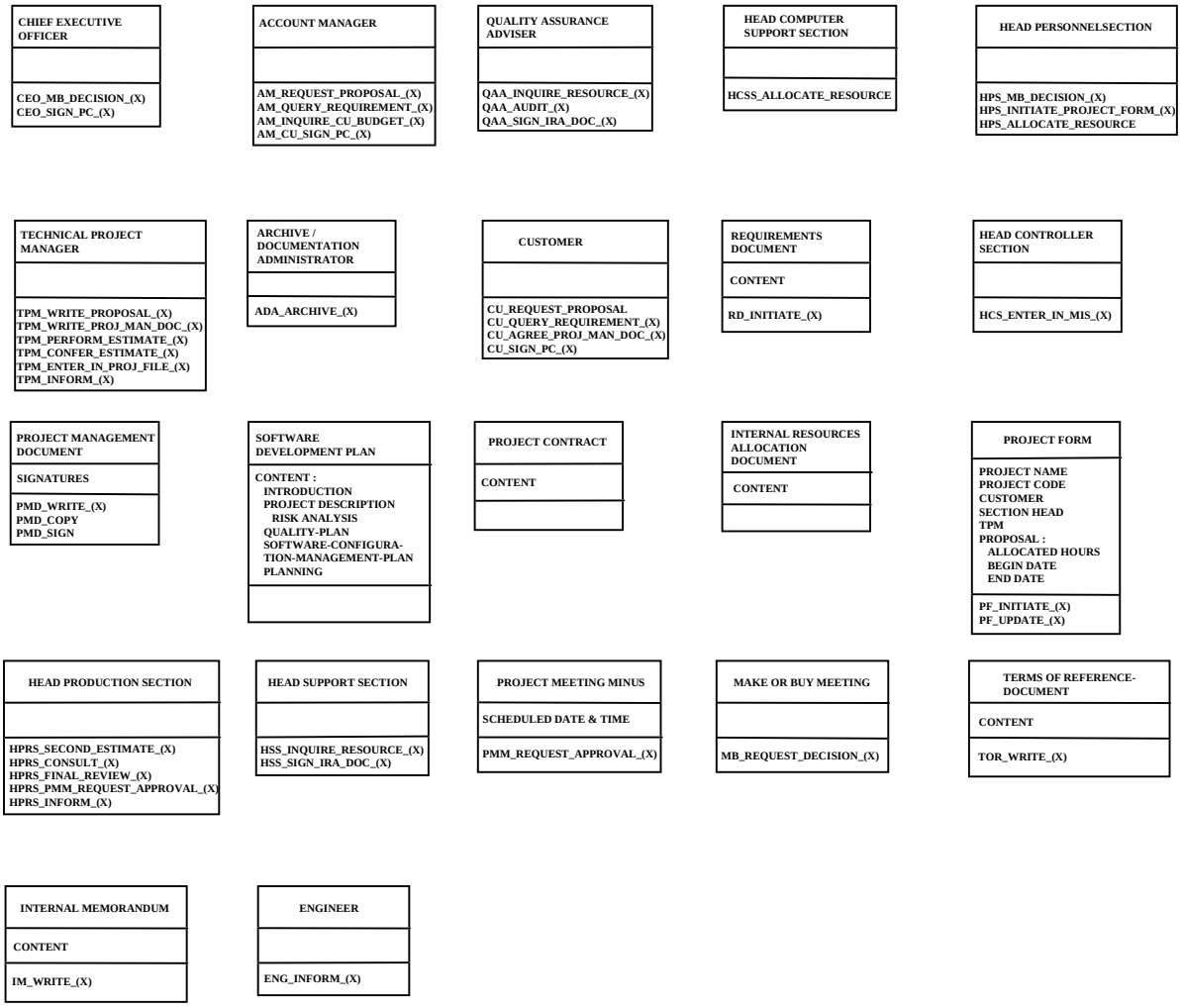


figure 5.4 Class diagram : classes, operations and attributes

The figure above shows the classes with their operations and attributes. The notation convention is the same in SOCCA as in UML. A class is depicted as a rectangle with three compartments. The top compartment holds the class name. The middle compartment shows the attributes. The bottom compartment shows the operations.

(The classes EMPLOYEE, MANAGER, SECTION HEAD, HEAD EXTERNAL PROJECTS SECTION, HEAD INTERNAL PROJECTS SECTION, HEAD INFRASTRUCTURE SECTION, MEETING, PROJECT TEAM, PROJECT FILE and PROPOSAL have no operations or attributes that are relevant for the SPP process. So these classes are not shown in the above class diagram. The class PROJECT is used during the integration of the four phases. The then relevant operations will be given in chapter 6 ‘Integration of process fragment ‘writing project management documents’’.)

The operations will be explained in the ‘internal behavior STDs’-subparagraphs of the ‘State Transition Diagrams’-paragraph of this chapter. The meaning of the attributes can be readily deduced from their name.

The next paragraphs give the import-export diagrams seperately for each phase. These diagrams identify which operations are imported by which classes. Within the importing classes the importing operations are identified. This is done by the SOCCA specific binary association ‘uses’. This association has the attribute ‘import_list’ that has as its domain a list of imported operations together with the operations that import them. The style guideline for this association is a solid line with an arrow at one and. The arrow indicates the exporting class. For a comparison between the SOCCA notation and the UML notation : see the ‘class diagrams’-chapter of the KPA ‘Software Configuration Management’.

The import-export diagrams are showing the communication between the classes at the highest level and are constructed as a step towards the constructing of the state transition diagrams (STDs).

5.3.2 Import-export diagram - Phase 1

The first import-exports diagram is the diagram of phase 1, ‘recognizing customer requirements’, of the process fragment ‘writing project management documents’.

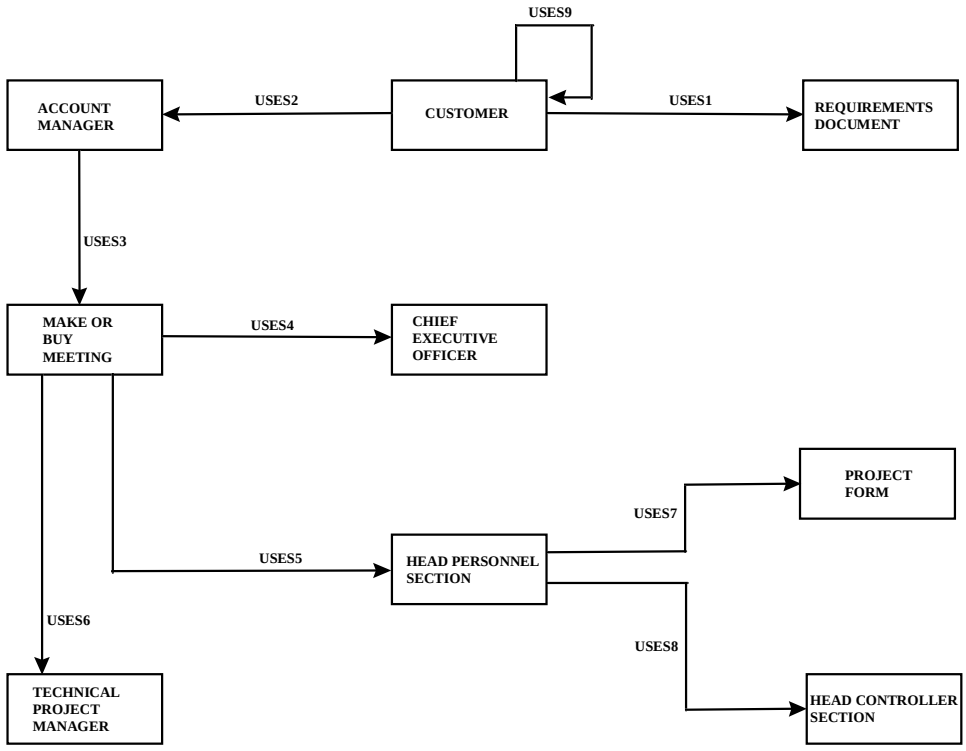


figure 5.5 Class diagram : classes and uses associations of phase 1 of process fragment ‘writing project management documents’

The customer is the one who initiates phase 1 of the process fragment by his autonomous behavior. Since the phases of the process fragment take place sequentially, the customer therefore initiates the whole process fragment.

The customer perceives a need to automate (part of) his business process. He may possibly be guided in this perception by the account manager, but this is not part of the model. The customer invokes his own operation cu_request_proposal. With this operation he writes a requirements document. He then aks the account manager to give him a proposal based on this requirements document.

The account manager then discusses this request for proposal in a ‘make or buy’ meeting with the chief executive officer and the head of the personnel section of the Waco Business Unit (WBU).

Depending on the estimated project size and the projection of the available (manpower) resources, a decision is reached either to do the project in-house (make) or to procure it from an outside vendor (buy). A combination of these two is also possible. For example to do the information analysis phase of the project in-house and outsource the system design and coding.

If the decision is to ‘buy’, the ‘Software Subcontract Management’ process is initiated. This process also includes the writing of project management documents. This process however is not modeled further here. The model depicts the process when the decision is to ‘make’.

If the decision is to ‘make’, the head of the personnel section initiates a project form for the project. He enters on the project form : the customer, the project code, the technical project manager responsible for the project, the section head of the tpm, the allocated hours for making the proposal and the time frame for making the proposal. This form is given to the head of the controller section. The data of this form is then entered by the controller section in the Management Information System (MIS) of the Waco Business Unit (WBU).

Finally the assigned technical project manager tasked with writing the proposal (the tpm's operation 'tpm_write_ proposal_(x)' is called). He does this by writing (a preliminary version of) a software development plan, a 'internal resources allocation'-document and a project contract. The calling of the tpm's operation 'tpm_write_proposal_(x)' starts phase 2,'writing and consultation', of the process fragment 'writing project management documents'.

In terms of the values of the 'import-list'-attributes of the uses associations this amounts to the following :

	imported operation	imported by
uses1 :	rd_initiate_(x)	cu_request_proposal
uses2 :	am_request_proposal_(x)	cu_request_proposal
uses3 :	mb_request_decision_(x)	am_request_proposal_(x)
uses4 :	ceo_mb_decision_(x)	mb_request_decision_(x)
uses5 :	hps_mb_decision_(x)	mb_request_decision_(x)
	hps_initiate_project_form_(x)	mb_request_decision_(x)
uses6 :	tpm_write_proposal_(x)	mb_request_decision_(x)
uses7 :	pf_initiate_(x)	hps_initiate_project_form_(x)
uses8 :	hcs_enter_in _mis_(x)	hps_initiate_project_form_(x)
uses9 :	cu_request_proposal	autonomous operation

5.3.3 Import-export diagram - Phase 2

The second import-exports diagram is the diagram of phase 2, ‘writing and consultation’, of the process fragment ‘writing project management documents’. This phase 2 is started by the calling of the tpm’s operation ‘tpm_write_proposal_(x)’ in phase 1.

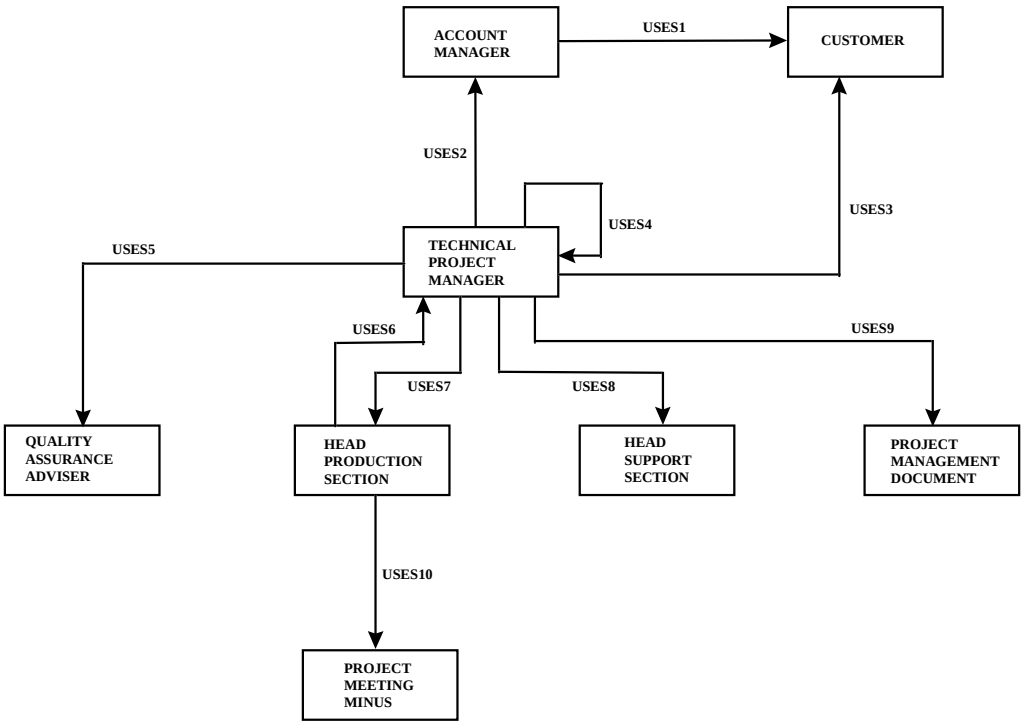


figure 5.6 Class diagram : classes and uses associations of phase 2 of process fragment ‘writing project management documents’

Before the technical project manager actually starts writing he first checks the requirements document on consistency and completeness. If he has any questions in this respect, he queries the customer via the account manager. He then performs a risk analysis and estimates the necessary person-hours for the project.

Then the technical project manager informs his section head (head production section) to inform him that he needs a second estimate. The head production section tasks another technical project manager (tpm) to perform the second estimate. This second estimate is returned to the first tpm. The first tpm then confers with the second tpm to match both their estimates.

When consensus is reached by the technical project managers about a final estimate, the first manager proceeds by writing a software development plan, a ‘internal resources allocation’-document and a project contract. During this process he consults (informs) his section head regularly. When writing the software development plan and the project contract the tpm has to check with the account manager if the budget of the customer is sufficient for the project. The tpm also determines regularly if the customer agrees with the software development plan and the project contract. This applies both for the intermediate versions and the final version of the software development plan and the project contract.

When writing the ‘internal resources allocation’-document the technical project manager checks with the heads of all the support sections and with the quality assurance adviser to detemine if the needed internal resources are available within the time frame of the project. Internal resources consist of : (software)engineers, computer equipment, support software, person-hours of the computer support section, work locations, office equipment, person-hours of the controller section and person-hours of the quality assurance adviser.

The technical project manager presents the final version of all three project management documents to his section head (head production section) for review. If the section head has still some comments at this stage, the tpm updates the documents accordingly and consults again (when necesary) with account manager, customer or support section heads.

If the section head of the tpm approves of the documents, the technical project manager presents them for a quality audit to the quality assurance adviser. The technical project manager updates the documents according to the comments of the quality assurance adviser (if any). He then gives the project management documents to his section head for their approval by the management (call hprs_request_approval_(x)).

The section head of the tpm then requests the approval of the documents in the next 'project meeting minus' (the opeartion 'pmm_request_approval_(x)' of the 'project meeting minus'-class is called by the section head). The calling of the operation 'pmm_ request_approval_(x)' starts phase 3, 'approval', of the process fragment 'writing project management documents'.

In terms of the values of the 'import-list'-attributes of the uses associations this amounts to the following :

	imported operation	imported by
uses1 :	cu_query_requirement_(x)	am_query_requirement_(x)
uses2 :	am_query_requirement_(x)	tpm_write_proposal_(x)
	am_inquire_cu_budget_(x)	tpm_write_proj_man_doc_(x,y)
uses3 :	cu_agree_proj_man_doc_(x)	tpm_write_proj_man_doc_(x,y)
uses4 :	tpm_perform_estimate_(x)	tpm_write_proposal_(x)
	tpm_confer_estimate_(x)	tpm_write_proposal_(x)
	tpm_write_proj_man_doc_(x,y)	tpm_write_proposal_(x)
uses5 :	qaa_inquire_resource_(x)	tpm_write_proj_man_doc_(x,y)
	qaa_audit_(x)	tpm_write_proposal_(x)
uses6 :	tpm_perform_estimate_(x)	hprs_second_estimate_(x)
uses7 :	hprs_second_estimate_(x)	tpm_write_proposal_(x)
	hprs_consult_(x)	tpm_write_proj_man_doc_(x,y)
	hprs_final_review_(x)	tpm_write_proposal_(x)
	hprs_pmm_request_approval_(x)	tpm_write_proposal_(x)
uses8 :	hss_inquire_resource_(x)	tpm_write_proj_man_doc_(x,y)
uses9 :	pmd_write_(x)	tpm_write_proj_man_doc_(x,y)
uses10 :	pmm_request_approval_(x)	hprs_pmm_request_approval_(x)

5.3.4 Import-export diagram - Phase 3

The third import-exports diagram is the diagram of phase 3, ‘approval’, of the process fragment ‘writing project management documents’. This phase 3 is started by the calling of the ‘project meeting minus’’s operation ‘pmm_request_approval_(x)’ in phase 2.

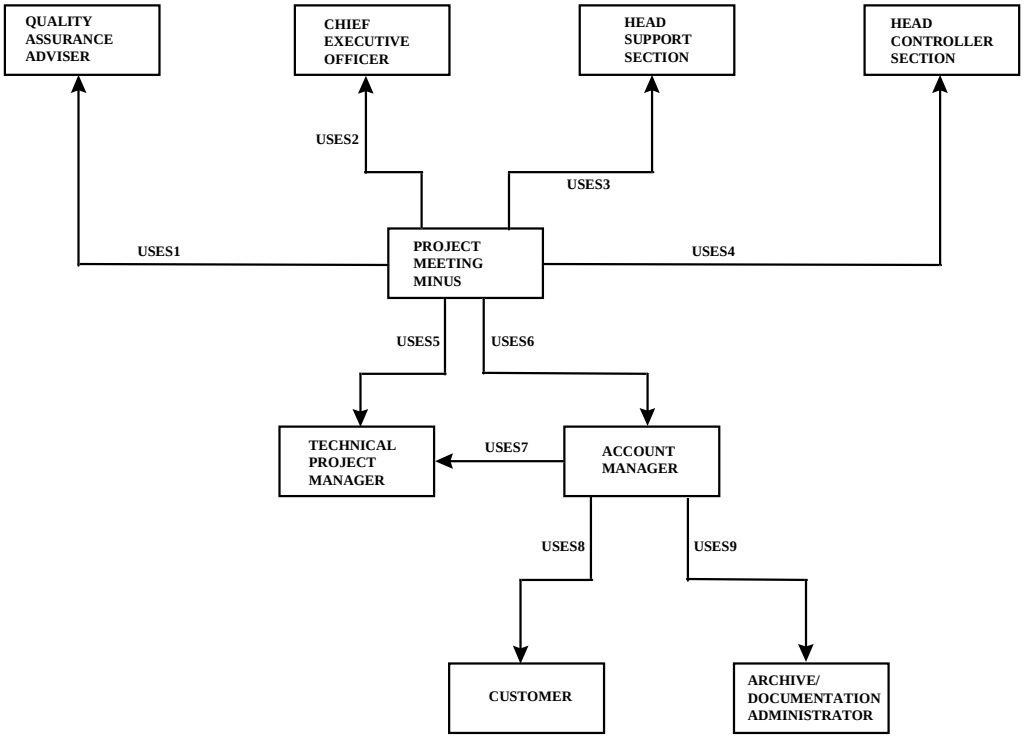


figure 5.7 Class diagram : classes and uses associations of phase 3 of process fragment ‘writing project management documents’

The project management documents that are handled by the ‘project meeting minus’ are a project contract, a software development plan and a ‘internal resources allocation’-document (ira-document). The ira-document contains the project form as an appendix.

The functionality of this phase is as follows. First 2 copies of the project contract are presented to the chief executive officer (ceo). He signs both copies of the contract document. Since the contract consists of a tailored template and is already reviewed by the ‘head of the production section’ during the writing of the document, the signing does takes place without the chief executive officer commenting on it.

Then the ‘internal resources allocation’-document is presented to the heads of the support section (hss’s) and also to the quality assurance adviser (qaa). They all sign the ‘ira’-document. Again the ‘ira’-document is already reviewed by the heads of the support sections and the quality assurance adviser during the writing of the document. In this stage the document is signed without commenting on it. By signing of the ‘ira’-document the hss’s and the qaa give their commitment that the resources stipulated in the ‘ira’-document will be delivered by them and/or their sections.

Next the head of the controller section receives the updated project form. The project form is an appendix to te ‘ira’-document and has been updated by the technical project manager during the writing of the ‘ira’- document. The updated data in the project form is entered by the (personnel of the) controller section into the Management Information System (MIS) of the Waco Business Unit (WBU).

Then the ‘project meeting minus’ gives the two copies of the project contract to the account manager. Appended to each copy of the project contract is the software development plan. At this stage the project contract contains the signature of the ceo and needs to be signed by the customer.

The signed ‘ira’-document is given to the technical project manager. He files this in ‘the project file’ of the project.

The account manager presents the two copies of the project contract (plus appended software development plan) to the customer for his signature. The customer signs both copies. He keeps one copy for himself and returns the other. Since the customer has already reviewed the project contract and the software development plan during the writing stage of these documents, the documents are signed without comment.

The returned copy is now duplicated. The original is filed by ‘archive/documentation administrator’ in the central archive. The duplicate is filed by the technical project manager in the ‘project file’ of the project.

In terms of the values of the ‘import-list’-attributes of the uses associations this amounts to the following :

	imported operation	imported by
uses1 :	qaa_sign_ira_doc_(x)	pmm_request_approval_(x)
uses2 :	ceo_sign_pc_(x)	pmm_request_approval_(x)
uses3 :	hss_sign_ira_doc_(x)	pmm_request_approval_(x)
uses4 :	hcs_enter_in_mis_(x)	pmm_request_approval_(x)
uses5 :	tpm_enter_in_proj_file_(x)	pmm_request_approval_(x)
uses6 :	am_cu_sign_pc_(x)	pmm_request_approval_(x)
uses7 :	tpm_enter_in_proj_file_(x)	am_cu_sign_pc_(x)
uses8 :	cu_sign_pc_(x)	am_cu_sign_pc_(x)
uses9 :	ada_archive_(x)	am_cu_sign_pc_(x)

5.3.5 Import-export diagram - Phase 4

The fourth import-export diagram is the diagram of phase 4, ‘resources allocation’, of the process fragment ‘writing project management documents’. In this phase 4 the ‘head of the personnel section’ and the ‘head of the computer support section’ allocate resources to a project as specified in the ‘internal resources allocation’-document of that project.

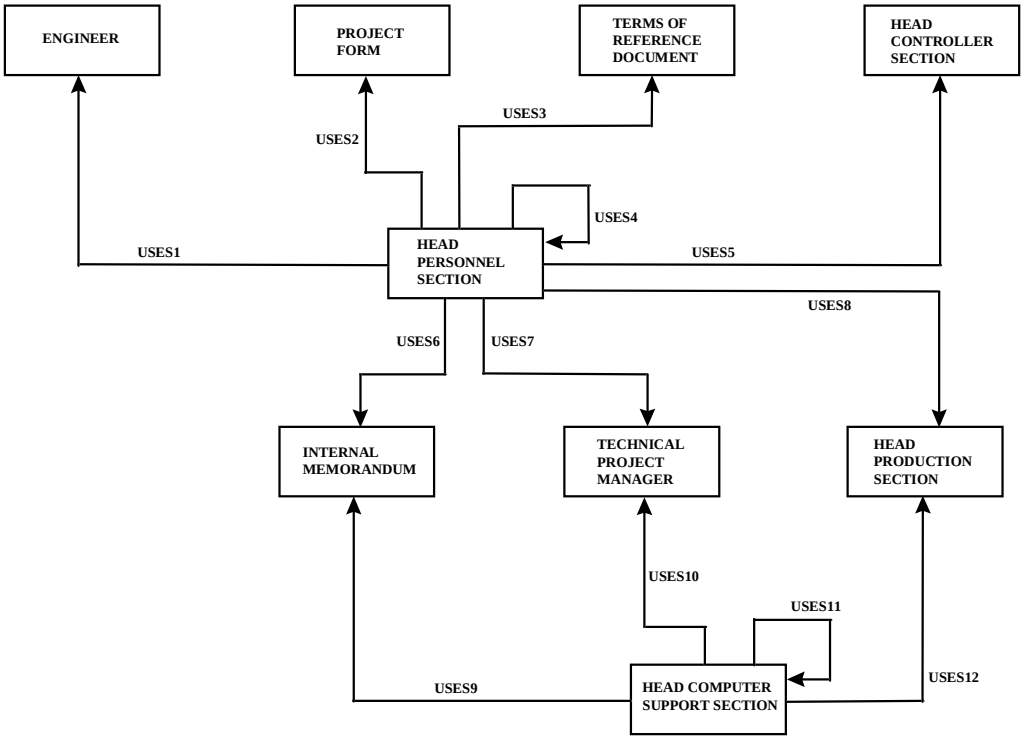


figure 5.8 Class diagram : classes and uses associations of phase 4 of process fragment ‘writing project management documents’

The ‘head of the personnel section’ (hps) allocates manpower to the project. He writes a ‘terms of reference’-document (tor-document). This document specifies for an engineer to which project he is allocated and for how many hours per year.

The hps also updates the project form accordingly. The updated data in the project form is entered by the (personnel of the) controller section into the Management Information System (MIS) of the Waco Business Unit (WBU).

The technical project manager, his head production section, and the engineer involved, and his head production section, are given the tor-document (plus an internal memorandum). In this way they are informed of the allocation of the engineer to the project.

The ‘head of the computer support section’ writes an internal memorandum to indicate the allocation of computer support personnel to a project. This memorandum is then communicated to the technical project manager and his head of production section.

In terms of the values of the ‘import-list’-attributes of the uses associations this amounts to the following :

	imported operation	imported by
uses1 :	eng_inform_(x)	hps_allocate_resource
uses2 :	pf_update_(x)	hps_allocate_resource
uses3 :	tor_write_(x)	hps_allocate_resource

	imported operation	imported by
uses4 :	hps_allocate_resource	autonomous operation
uses5 :	hcs_enter_in_mis_(x)	hps_allocate_resource
uses6 :	im_write_(x)	hps_allocate_resource
uses7 :	tpm_inform_(x)	hps_allocate_resource
uses8 :	hprs_inform_(x)	hps_allocate_resource
uses9 :	im_write_(x)	hcss_allocate_resource
uses10 :	tpm_inform_(x)	hcss_allocate_resource
uses11 :	hcss_allocate_resource	autonomous operation
uses12 :	hprs_inform_(x)	hcss_allocate_resource

5.3.6 State Transition Diagrams - Phase 1

The behavior perspective of ‘phase 1’ of the process fragment ‘writing project management documents’ is modeled in SOCCA by STDs (State Transition Diagrams). The behavior perspective comprises both the visible behavior of classes (and thus of objects of that classes) and the behavior of operations (methods) of the classes.

The visible behavior of a class is modeled by an external STD. This STD shows the possible sequences of starting the execution of the class’ (object’s) operations. Usually there is one external STD per class. The UML equivalent of an external STD is a statechart describing the states an object can be in and the transitions it can make in response to received stimuli.

The behavior of an operation is modeled by an internal STD. It describes the functionality of the operation. This comprises the operation’s start and end, the calling of other operations by this operation, and ‘local steps’ (local functionality). In UML methods are also described by statecharts.

The communication between objects (calling of exported operations) is modeled by manager STDs and employee STDs. External STDs become manager STDs and internal STDs become employee STDs. This mechanism is explained in the SOCCA chapter of this document. This way of modeling communication is unique for SOCCA.

STD’s can show different views of the same process. An STD is called a ‘communicative view’ if it models any communication details between one or more caller and its callee(s). An STD is called an ‘organizational view’ otherwise.

Phase 1, ‘recognizing customer requirements’, of the process fragment ‘writing project management documents’ (partly) models the behavior of the following classes :

- customer
- requirements document
- account manager
- make or buy-meeting
- chief executive officer
- head personnel section
- project form
- head controller section
- technical project manager

5.3.6.1 Customer

5.3.6.1.1 Customer : external behavior-STD, organizational view

The external behavior STD shows the possible sequences in which the exported operation(s) can be started. The class 'customer' has only one operation relevant for the process fragment 'writing project management documents', phase1. The organizational view of the external STD does not show any communication details. Consequently the STD consists of one state 'neutral' in which the customer waits for a call to its exported operation 'cu_request_proposal'. If the call has taken place, the customer can make the transition labeled 'cu_request_proposal'. The customer then comes again in the state 'neutral'. The possible starting sequence specified by this STD is thus 'cu_request_proposal', 'cu_request_ proposal', etc.

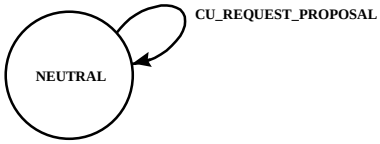


figure 5.9 customer : external behavior STD, organizational view

5.3.6.1.2 Customer : external behavior-STD, communicative view

The communicative view of the external behavior STD shows the communication details between the external STD and the called operation(s). It consists of a neutral state in which the customer waits for a call to its operation 'cu_request_ proposal' and of a state in which he starts this operations. Typically the customer does not wait in this 'starting' state until the called operation is finished, but returns as soon as possible to its neutral state to allow handling of its other operations. These other operations are not shown here because they are not relevant to the process fragment 'writing project management documents', phase 1.

The customer exihibits autonomous behavior by starting his own operation without being called from another object.

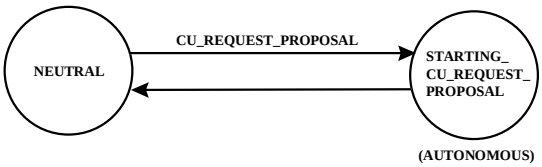


figure 5.10 customer : external behavior STD, communicative view

5.3.6.1.3 Customer : internal behavior-STDs

The 1 operation 'cu_request_proposal' of the customer has the following internal behavior STD.

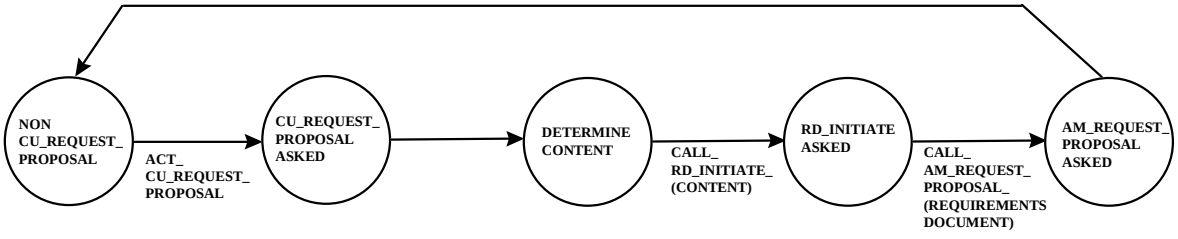


figure 5.11 int-cu_request_proposal : internal behavior STD

With the operation 'cu_request_proposal' the customer considers his requirements (= determines the contents of the requirements document). Then he creates a requirements document and initiates it with his requirements (call_

rd_initiate_(content)). With this requirements document he approaches the account manager with a request for a proposal based on the requirements (call_am_request_proposal_(requirements document)).

5.3.6.1.4 Customer : manager-STD

The communication between the customer's operation (callee) and its caller is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

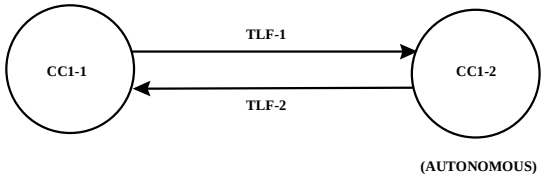


figure 5.12 customer : manager STD

The notation CCx in the STD stands for 'Caller(s)-Callee' and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for 'Traps Logical Formula' and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

The caller-callee combination for 'cu_request_proposal' consist only of prescribed subprocesses of the callee 'cu_ request_proposal'. This is because 'cu_request_proposal' is an automous operation.

In the state 'neutral' the CC and the TLF for the transition leaving the state are :

CC1-1 = {S1}
TLF-1 = T-1 (corresponds with 'cu_request_proposal' transition in extern STD)

In the state 'starting_cu_request_proposal' the CC and the TLF for the transition leaving the state are :

CC1-2 = {S2}
TLF-2 = T-2

5.3.6.1.5 Customer : employee-STDs

Internal operations are normally started with the caller_callee-construct. In the case of autonomous operations the caller_callee-construct consists only of the callee-part (i.e. the act-construct). The caller_callee-construct, the act-construct and autonomous behavior are described in more detail in the chapter explaining the SOCCA concepts.

This manager STD has 1 employee, the autonomous operation 'cu_request_proposal'. This operation is started with the act-construct.

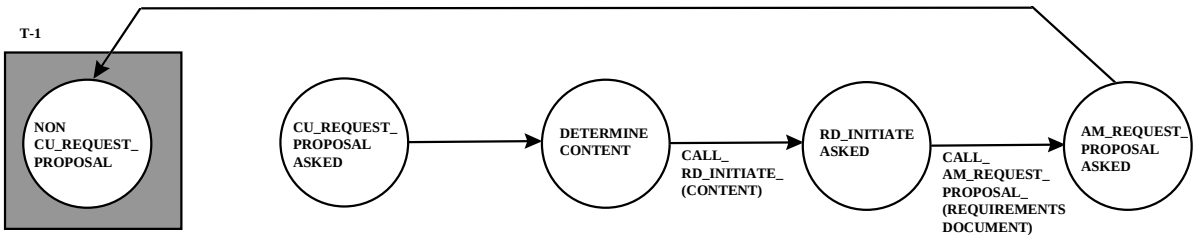


figure 5.13 employee int-cu_request_proposal : subprocess S1

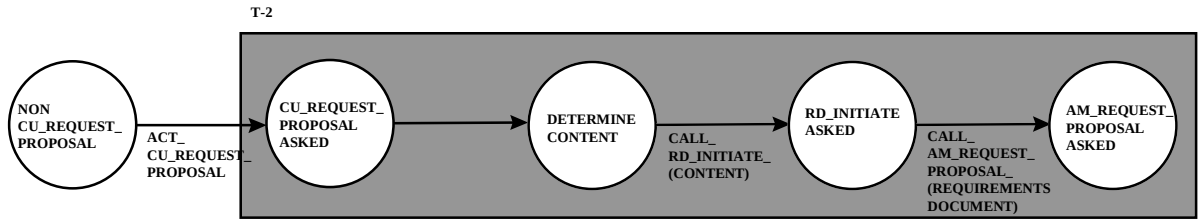


figure 5.14 employee int-cu_request_proposal : subprocess S2

When the act-construct is applied to the employee, the result is the following. The employee has two subprocesses S1 and S2, and two traps T-1 and T-2. In its neutral state the manager prescribes the subprocess S1 for the employee. The manager can transit to the state ‘starting_cu_request_proposal’ when the subprocess S1 is in trap T-1 and the customer decides autonomously to perform the operation ‘cu_request_proposal’. In the state ‘starting_cu_request_proposal’ the manager prescribes the subprocess S2. This means that the employee can make its transition ‘act_cu_request_proposal’ which means that the operation can start executing. As soon as the subprocess S2 enters in trap T-2, the manager can transit back to the neutral state.

5.3.6.2 Requirements document

5.3.6.2.1 Requirements document : external behavior-STD, organizational view

The external behavior STD shows the possible sequences in which the exported operation(s) can be started. The class 'requirements document' has only one operation relevant for the process fragment 'writing project management documents', phase1. The organizational view of the external STD does not show any communication details. Consequently the STD consists of one state 'neutral' in which the requirements document waits for a call to its exported operation 'rd_initiate_(x)'. If the call has taken place, the requirements document can make the transition labeled 'rd_initiate_(x)'. The requirements document then comes again in the state 'neutral'. The possible starting sequence specified by this STD is thus 'rd_initiate_(x)', 'rd_initiate_(x)', etc.

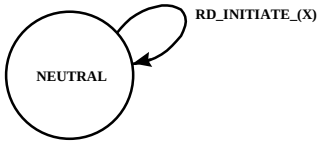


figure 5.15 requirements document : external behavior STD, organizational view

5.3.6.2.2 Requirements document : external behavior-STD, communicative view

The communicative view of the external behavior STD shows the communication details between the external STD and the called operation(s). It consists of a neutral state in which the requirements document waits for a call to its operation 'rd_initiate_(x)' and of a state in which he starts this operation. Typically the requirements document does not wait in this 'starting' state until the called operation is finished, but returns as soon as possible to its neutral state. It can then handle another call to the operation 'rd_initiate_(x)' before the current execution of the operation has finished.

The external STD still has another state, 'requirements document created'. The reason for this state is explained in the 'requirements document : manager-STD' chapter.

The callers of the operations of this class can be found in the import-export diagram. They are given in the the 'import_list' attribute of the 'uses association'.

.

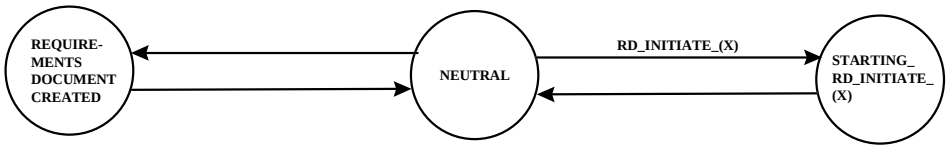


figure 5.16 requirements document : external behavior STD, communicative view

5.3.6.2.3 Requirements document : internal behavior-STDs

The 1 operation 'rd_initiate_(x)' of the requirements document has the following internal behavior STD.

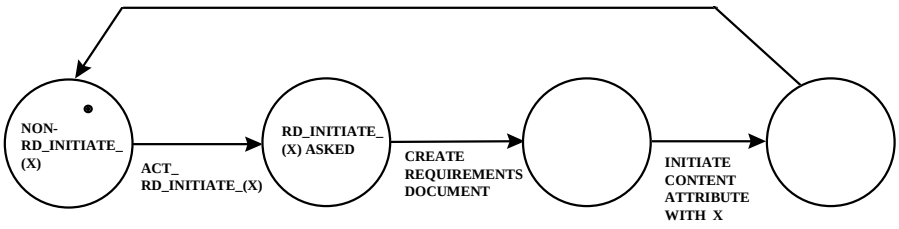


figure 5.17 int-rd_initiate_(x) : internal behavior STD

The operation ‘rd_inititate_(x)’ has one formal parameter ‘x’. The multiplicity of concurrent executing STD instances is zero or more. This is indicated by the solid circle inside the state ‘non rd_initiate_(x)’. A multiplicity of zero or more means that the manager STD can start another instance of the internal STD to execute concurrently with an already executing instance of the internal STD.

The formal parameter of the operation ‘rd_initiate_(x)’ is the content with which the newly created requirements document is initialized. The operation first creates a new requirements document object and then initializes its attribute ‘content’ with the actual parameter value of ‘x’.

5.3.6.2.4 Requirements document : manager-STD

The communication between the requirements document’s operations (callees) and their callers is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

The caller of the operation ‘rd_initiate_(x)’ has to wait until the operation ‘rd_initiate_(x)’ has progressed far enough for the new requirements document object to be in existence . This is because the caller (the customer operation ‘cu_request_proposal’) needs the requirements document object (id) in its next action. It uses it as a parameter value in its ‘call_am_request_proposal_(x)’. For this reason the manager STD has the state CC1-3. This state also appears in the external STD as the state ‘requirements document created’.

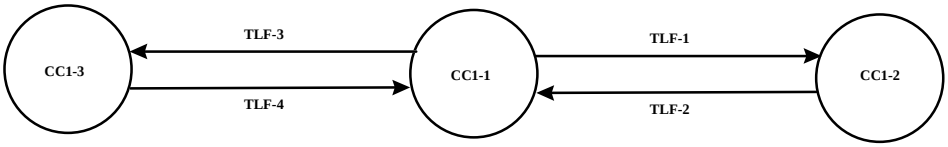


figure 5.18 requirements document : manager STD

The notation CCx in the STD stands for ‘Caller(s)-Callee’ and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for ‘Traps Logical Formula’ and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

There is only one caller-callee combination, the callee ‘rd_initiate_(x)’ and its caller ‘cu_request_proposal’ of the class customer. Because the caller has to wait for the result produced by the callee, the call is modeled by the ‘caller waits’-variant of the caller-callee construct (see SOCCA chapter for this construct). This results in the extra state in the manager STD and the fact that in the state ‘neutral’ either S1 or S2 is prescribed depending on the state the STD was in before it entered the state ‘neutral’.

In the state ‘neutral’ CC and the TLF for the transition leaving the state are :

CC1-1 = {S1 or S2, S3}
TLF-1 = T-1 and T-3 (corresponds with ‘rd_initiate_(x)’ transition in extern STD)
TLF-3 = T-2b

In the state ‘starting_rd_inititate_(x)’ the CC and the TLF for the transition leaving the state are :

CC1-2 = {S2, S3}
TLF-2 = T-2a

In the state ‘requirements document created’ the CC and the TLF for the transition leaving the state are :

CC1-3 = {S1, S4}
TLF-4 = T-4

5.3.6.2.5 Requirements document : employee-STDs

The manager STD has 2 employee STDs. These are the callee ‘rd_initiate_(x)’ and its caller ‘cu_request_proposal’ of the class customer. The calling of ‘rd_initiate_(x)’ is modeled by the ‘caller waits’-variant of the caller-callee construct.

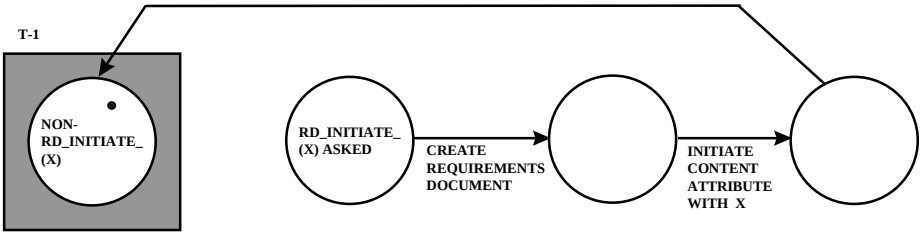


figure 5.19 employee int-rd_initiate_(x) : subprocess S1

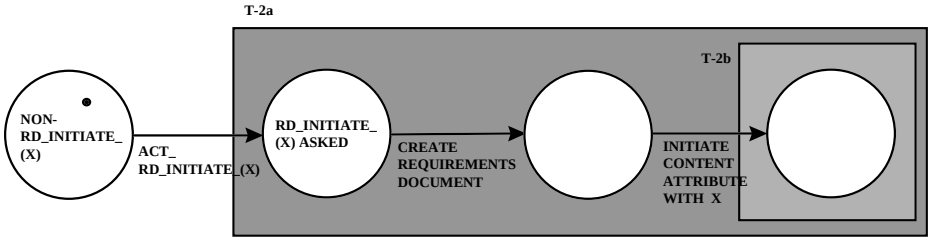


figure 5.20 employee int-rd_initiate_(x) : subprocess S2

The first employee is the own internal operation ‘rd_initiate_(x)’. This employee has two subprocesses S1 and S1 and three traps T-1, T-2a and T-2b. The trap T-2b is to determine that in fact the operation ‘rd_initiate_(x)’ has created and initialized a new requirements document.

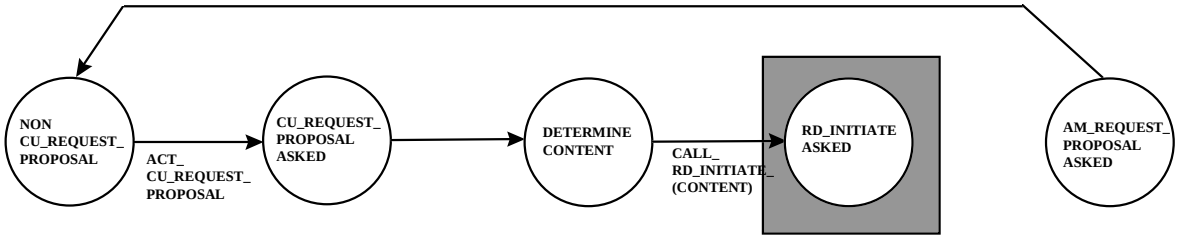


figure 5.21 employee int-cu_request_proposal : subprocess S3

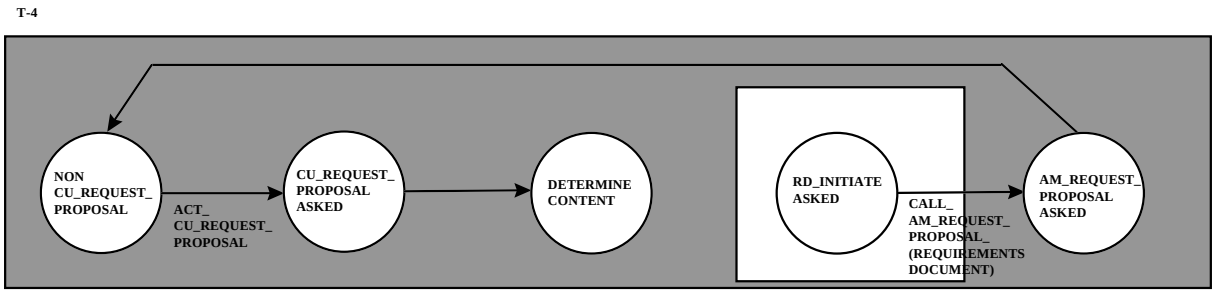


figure 5.22 employee int-cu_request_proposal : subprocess S4

The second employee is the caller operation ‘cu_request_proposal’. This employee has two subprocesses S3 and S4 and two traps T-3 and T-4.

In its neutral state the manager prescribes initially subprocess S1 for the callee and S3 for the caller. When the caller enters its trap T-3, i.e. excutes the call (and the callee is ready in its trap T-1) the manager can make the transition to the state

‘starting_rd_initiate_(x). Here it prescribes S2 for the callee and S3 for the caller (i.e the caller stays in subprocess S3). When the callee has entered trap T-2a the manager can transit back to its neutral state. Now it prescribes S2 (instead of S1) for the callee and S3 for the caller. If the callee has finished its action, i.e it has entered trap T2-b, the manager can transit to state ‘requirements document created’. Here it prescribes S1 for the callee and S4 for the caller (i.e. the caller is allowed to proceed). When the caller enters T-4, the manager can transit back to its neutral state. Here it prescribes S1 again for the callee.

5.3.6.3 Account manager

5.3.6.3.1 Account manager : external behavior-STD, organizational view

The external behavior STD shows the possible sequences in which the exported operation(s) can be started. The class 'account manager' has only one operation relevant for the process fragment 'writing project management documents', phase1. The organizational view of the external STD does not show any communication details. Consequently the STD consists of one state 'neutral' in which the account manager waits for a call to its exported operation 'am_request_proposal_(x)'. If the call has taken place, the account manager can make the transition labeled 'am_request_proposal_(x)'. The account manager then comes again in the state 'neutral'. The possible starting sequence specified by this STD is thus 'am_request_proposal_(x)', 'am_request_proposal_(x)', etc.

figure 5.23 account manager : external behavior STD, organizational view

5.3.6.3.2 Account manager : external behavior-STD, communicative view

The communicative view of the external behavior STD shows the communication details between the external STD and the called operation(s).

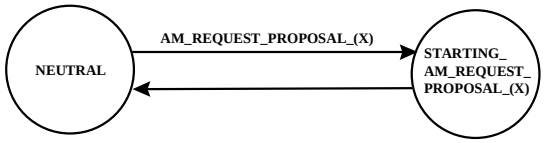


figure 5.24 account manager : external behavior STD, communicative view

This external behavior STD has two states. In the state neutral the account manager waits until a call has been placed to its operation 'am_request_proposal_(x). If a call has been made, the manager STD can (and eventually will) transit to the state 'starting_am _request_proposal_(x)'. If the operation has been started the manager can transit back to the state neutral.

The callers of the operations of this class can be found in the import-export diagram. They are given in the the 'import_list' attribute of the 'uses association'.

5.3.6.3.3 Account manager : internal behavior-STDs

The account manager has 1 operation : 'am_request_proposal_(x)'. This operation has the following internal behavior STD.

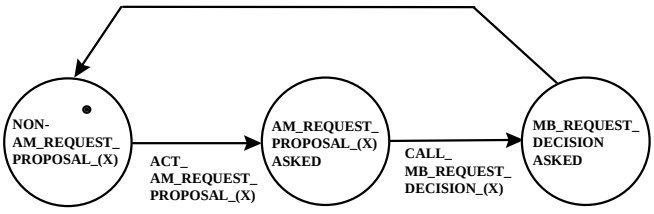


figure 5.25 int-am_request_proposal_(x) : internal behavior STD

The operation ‘am_request_proposal_(x)’ has one formal parameter ‘x’. The multiplicity of concurrent executing STD instances is zero or more. This is indicated by the solid circle inside the state ‘non am_request_proposal_(x)’. A multiplicity of zero or more means that the manager STD can start another instance of the internal STD to execute concurrently with an already executing instance of the internal STD.

The formal parameter of the operation ‘am_request_proposal_(x)’ is the requirements document on the basis of which the requested proposal has to be made. The account manager presents this requirements document to the ‘make or buy’ meeting’. This meeting will come to a decision to either do the project in-house (make) or to subcontract it to an outside vendor (buy). If the decision is to ‘make’ then the meeting (i.e. the participants in the meeting) will take care that a proposal is made based on the requirements document.

5.3.6.3.4 Account manager : manager-STD

The communication between the account manager’s operations (callees) and their callers is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

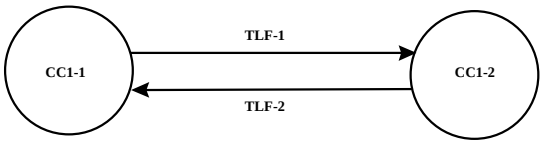


figure 5.26 account manager : manager STD

The notation CCx in the STD stands for ‘Caller(s)-Callee’ and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for ‘Traps Logical Formula’ and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

There is only one caller-callee combination, the callee ‘am_request_proposal_(x)’ and its caller ‘cu_request_proposal’ of the class customer. The calling is modeled by the ‘caller does not wait’-variant of the caller-callee construct (see SOCCA chapter for this construct). In the state ‘neutral’ CC and the TLF for the transition leaving the state are :

CC1-1 = {S1, S3}
TLF-1 = T-1 and T-3 (corresponds with ‘am_request_proposal_(x)’ transition in extern STD)

In the state ‘starting_am_request_proposal_(x)’ the CC and the TLF for the transition leaving the state are :

CC1-2 = {S2, S4}
TLF-2 = T-2 and T-4

5.3.6.3.5 Account manager : employee-STDs

The manager STD has 2 employee STDs. These are the callee ‘am_request_proposal_(x)’ and its caller ‘cu_request_proposal’ of the class customer.

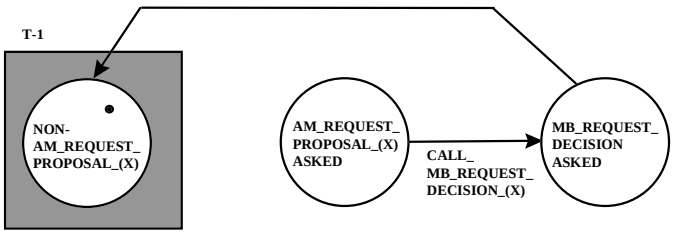


figure 5.27 employee int-am_request_proposal_(x) : subprocess S1

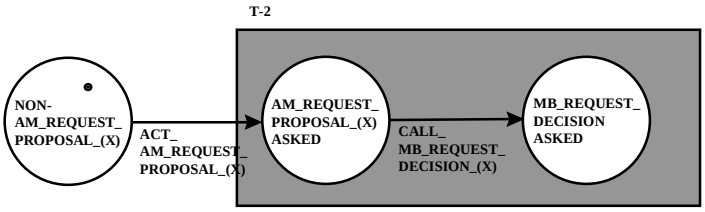


figure 5.28 employee int-am_request_proposal_(x) : subprocess S2

The first employee is the own internal operation ‘am_request_proposal_(x)’. This employee has two subprocesses S1 and S1 and two traps T-1 and T-2.

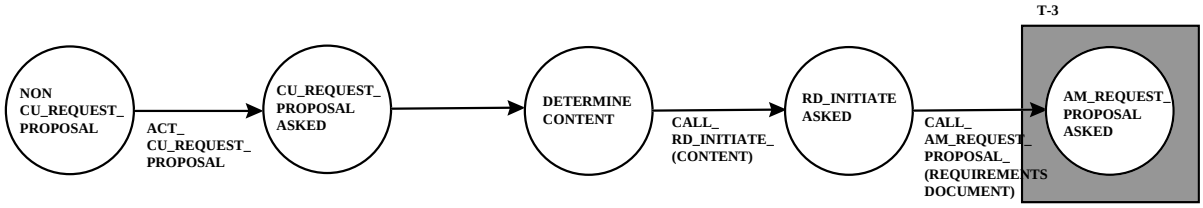


figure 5.29 employee int-cu_request_proposal : subprocess S3

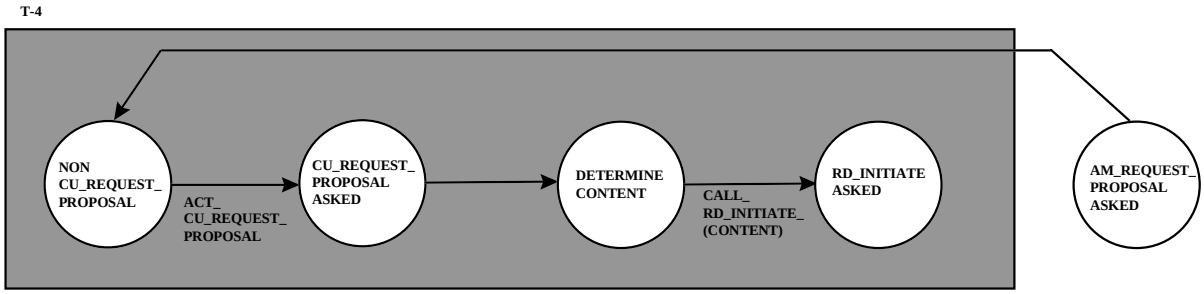


figure 5.30 employee int-cu_request_proposal : subprocess S4

The second employee is the caller operation ‘cu_request_proposal’. This employee has two subprocesses S3 and S4 and two traps T-3 and T-4.

The manager prescribes initially subprocess S3 for the calling employee, it is waiting for the call. When the caller enters its trap T-3, i.e. excutes the call (and the callee is ready in its trap T-1) the manager can make the transition to the next state. Here it prescribes S4 for the caller, thereby allowing it to proceed in its next subprocess. This has the effect that the caller does not wait for the result of the called operation but proceeds right away after the manager has started the called operation.

When the callee has entered its trap T-2 and the caller has entered his trap T-4, the manager can transit back to the state ‘neutral’.

5.3.6.4 Make or buy-meeting

5.3.6.4.1 Make or buy-meeting : external behavior-STD, organizational view

The external behavior STD shows the possible sequences in which the exported operation(s) can be started. The class ‘make or buy-meeting’ has only one operation relevant for the process fragment ‘writing project management documents’, phase1. The organizational view of the external STD does not show any communication details. Consequently the STD consists of one state ‘neutral’ in which the make or buy-meeting waits for a call to its exported operation ‘mb_request_decision_(x)’. If the call has taken place, the make or buy-meeting can make the transition labeled ‘mb_request_decision_(x)’. The make or buy-meeting then comes again in the state ‘neutral’. The possible starting sequence specified by this STD is thus ‘mb_request_decision_(x)’, ‘mb_request_decision_(x)’, etc.

figure 5.31 make or buy-meeting : external behavior STD, organizational view

5.3.6.4.2 Make or buy-meeting : external behavior-STD, communicative view

The communicative view of the external behavior STD shows the communication details between the external STD and the called operation(s).

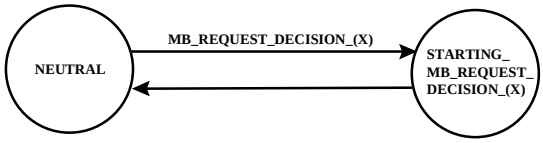


figure 5.32 make or buy-meeting : external behavior STD, communicative view

This external behavior STD has two states. In the state neutral the make or buy-meeting waits until a call has been placed to its operation ‘mb_request_decision_(x). If a call has been made, the manager STD can (and eventually will) transit to the state ‘starting_mb_request_decision_(x)’. If the operation has been started the manager can transit back to the state neutral. It can then start another instance of the operation ‘mb_request_decision_(x)’ if there has been another call to this operation. This instance can excute concurrently with the one(s) already excuting.

The callers of the operations of this class can be found in the import-export diagram. They are given in the the ‘import_list’ attribute of the ‘uses association’.

5.3.6.4.3 Make or buy-meeting : internal behavior-STDs

The make or buy-meeting has 1 operation : ‘mb-request_decision_(x)’. This operation has the following internal behavior STD.

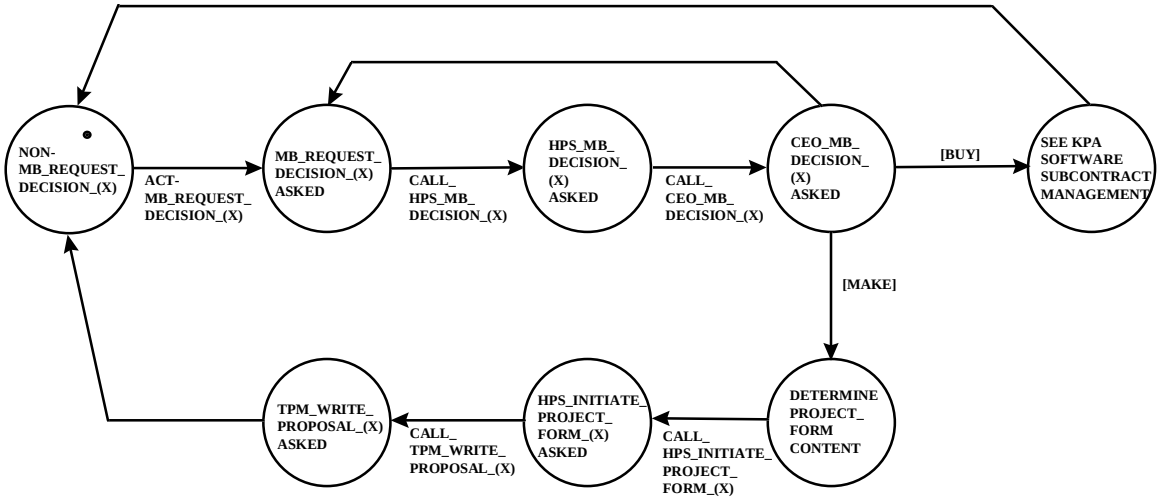


figure 5.33 mb_request_decision_(x) : internal behavior STD

The operation ‘mb_request_decision_(x)’ has one formal parameter ‘x’. The multiplicity of concurrent executing STD instances is zero or more. This is indicated by the solid circle inside the state ‘non mb_request_decision_(x)’. A multiplicity of zero or more means that the manager STD can start another instance of the internal STD to execute concurrently with an already executing instance of the internal STD.

The formal parameter of the operation ‘mb_request_decision_(x)’ is the requirements document on the basis of which the ‘make’ or ‘buy’ decision has to be made by the make or buy-meeting. The make or buy-meeting first asks the head of the personnel section for his decision. Then it asks the chief executive officer for his decision. If both decisions are different, the head of the personnel section is asked to reconsider his decision. Then the chief executive officer is asked again for his decision. This process repeats itself until both decisions are the same, or until the chief executive officer makes his decision final. If the decision is to buy from an outside vendor, the STD transits to the state ‘see KPA Software Subcontract Management’. This part of the process belongs to the Key Process Area (KPA) Software Subcontract Management and is not modeled further here.

If the decision is to develop the system in house, the project is give a name and a project code. A technical project manager is chosen to do the proposal. A time budget for this proposal writing task is allocated together with a start date and a due date. This infomation constitutes the initial content of the project form. The head of the personnel section initiates a project form and thereby starts the project. The chosen technical project manager is then tasked to write the proposal based on the requirements document.

5.3.6.4.4 Make or buy-meeting : manager-STD

The communication between the ‘make or buy-meeting’‘s operations (callees) and their callers is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

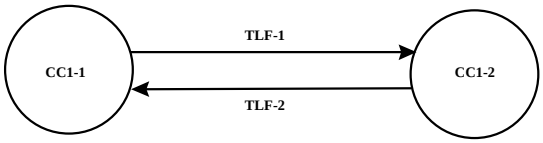


figure 5.34 make or buy-meeting : manager STD

The notation CCx in the STD stands for ‘Caller(s)-Callee’ and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for ‘Traps Logical Formula’ and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

There is only one caller-callee combination, the callee ‘mb_request_decision_(x)’ and its caller ‘am_request_proposal _(x)’ of the class account manager. The call is modeled by the ‘caller does not wait’-variant of the caller-callee construct (see SOCCA chapter for this construct).

In the state ‘neutral’ CC and the TLF for the transition leaving the state are :

CC1-1 = {S1, S3}
TLF-1 = T-1 and T-3 (corresponds with ‘mb_request_decision_(x)’ transition in extern STD)

In the state ‘starting_mb_request_decision_(x)’ the CC and the TLF for the transition leaving the state are :

CC1-2 = {S2, S4}
TLF-2 = T-2 and T-4

5.3.6.4.5 Make or buy-meeting : employee-STDs

The manager STD has the following 2 employee STDs. The first is its own internal operation the callee ‘mb_request_ decision_(x)’. The second is the caller ‘am_request_proposal_(x)’ from the class account manager.

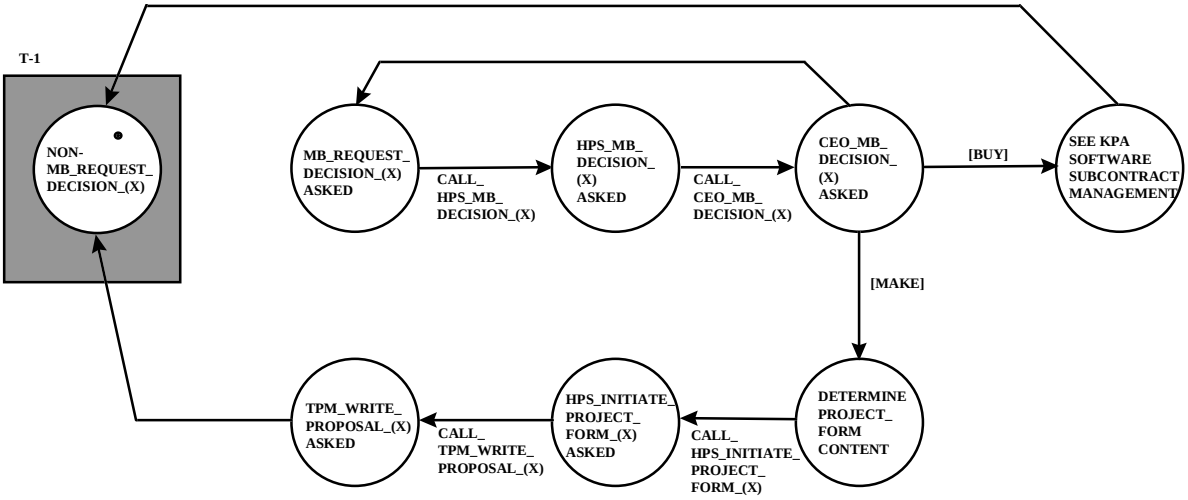


figure 5.35 employee int-mb_request_decision_(x) : subprocess S1

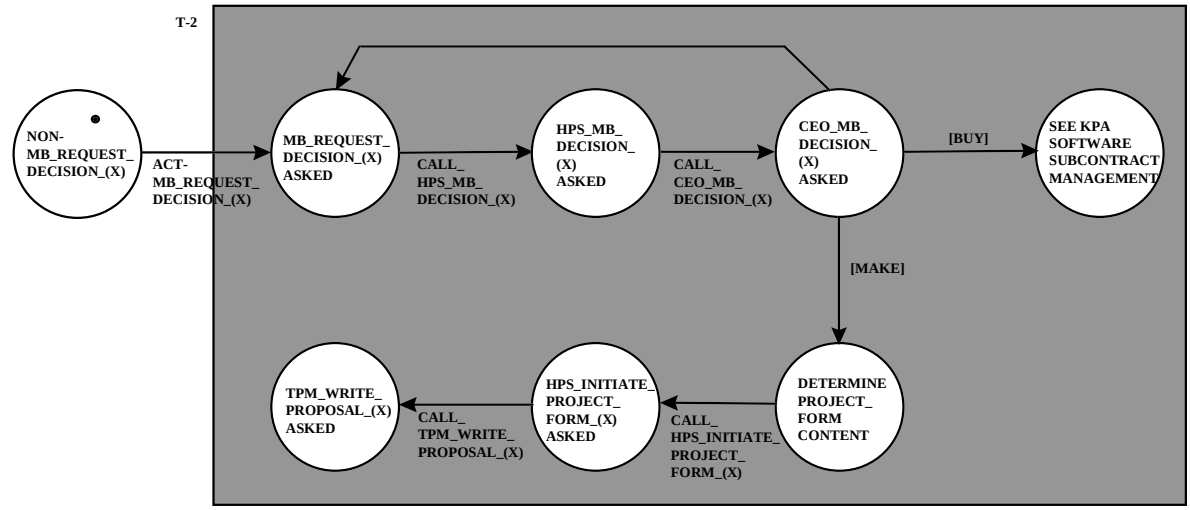


figure 5.36 employee int-mb_request_decision_(x) : subprocess S2

The first employee is the own internal operation ‘mb_request_decision_(x)’. This employee has two subprocesses S1 and S1 and two traps T-1 and T-2.

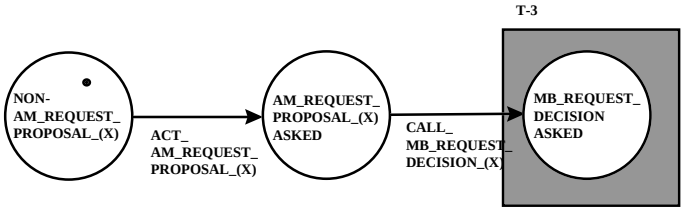


figure 5.37 employee int-am_request_proposal_(x) : subprocess S3

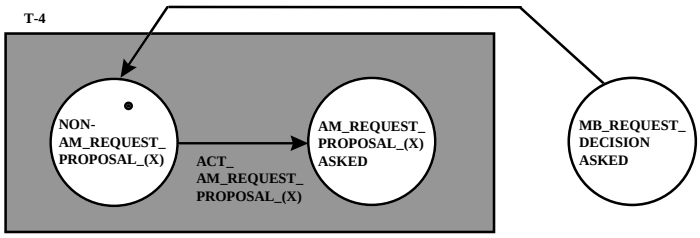


figure 5.38 employee int-am_request_proposal_(x) : subprocess S4

The second employee is the caller operation ‘am_request_proposal_(x)’. This employee has two subprocesses S3 and S4 and two traps T-3 and T-4.

The manager prescribes initially subprocess S3 for the calling employee, it is waiting for the call. When the caller enters its trap T-3, i.e. excutes the call (and the callee is ready in its trap T-1) the manager can make the transition to the next state. Here it prescribes S4 for the caller, thereby allowing it to proceed in its next subprocess. This has the effect that the caller does not wait for the result of the called operation but proceeds right away after the manager has started the called operation.

When the callee has entered its trap T-2 and the caller has entered his trap T-4, the manager can transit back to the state ‘neutral’.

5.3.6.5 Chief Executive Officer

5.3.6.5.1 Chief Executive Officer : external behavior-STD, organizational view

The external behavior STD shows the possible sequences in which the exported operation(s) can be started. The class 'chief executive officer' has only one operation relevant for the process fragment 'writing project management documents', phase1. The organizational view of the external STD does not show any communication details. Consequently the STD consists of one state 'neutral' in which the chief executive officer waits for a call to its exported operation 'ceo_mb_decision_(x)'. If the call has taken place, the chief executive officer can make the transition labeled 'ceo_mb_decision_(x)'. The chief executive officer then comes again in the state 'neutral'. The possible starting sequence specified by this STD is thus 'ceo_mb_decision_(x)', 'ceo_mb_decision_(x)', etc.

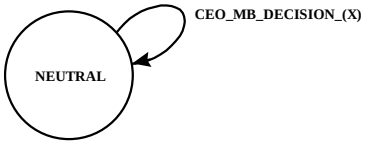


figure 5.39 chief executive officer : external behavior STD, organizational view

5.3.6.5.2 Chief Executive Officer : external behavior STD, communicative view

The communicative view of the external behavior STD shows the communication details between the external STD and the called operation(s).

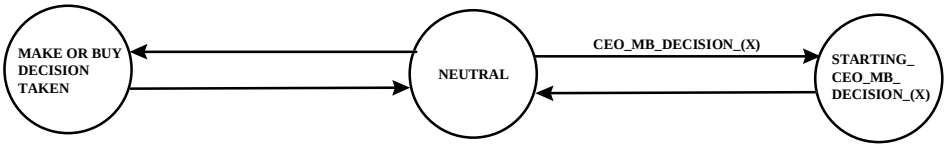


figure 5.40 chief executive officer : external behavior STD, communicative view

It consists of a neutral state in which the chief executive officer waits for a call to its operation 'ceo_mb_decision_(x)' and of a state in which he starts this operations. Typically the chief executive officer does not wait in this 'starting' state until the called operation is finished, but returns as soon as possible to its neutral state. It can then handle another call to the operation 'ceo_mb_decision_(x)' before the current execution of the operation has finished.

The external STD still has another state, 'make or buy decision taken'. The reason for this state is explained in the 'chief executive officer : manager-STD' paragraph.

The callers of the operations of this class can be found in the import-export diagram. They are given in the the 'import_list' attribute of the 'uses association'.

5.3.6.5.3 Chief Executive Officer : internal behavior-STDs

The 1 operation 'ceo_mb_decision' of the chief executive officer has the following internal behavior STD.

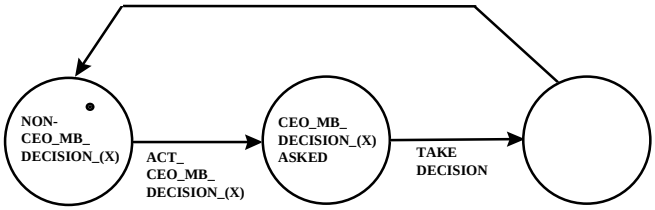


figure 5.41 int-ceo_mb_decision_(x) : internal behavior STD

The operation has one formal parameter ‘x’. The multiplicity of concurrent executing STD instances is zero or more. This is indicated by the solid circle inside the state ‘non ceo_mb_decision_(x)’. A multiplicity of zero or more means that the manager STD can start another instance of the internal STD to execute concurrently with an already executing instance of the internal STD.

The formal parameter of the operation ‘ceo_mb_decision_(x)’ is the requirements document on the basis of which the ‘make’ or ‘buy’ decision has to be made by the chief executive officer. After being started the operation (=chief executive officer) takes this decision as an ‘internal action’. The decision is then communicated back to the caller of this operation. This means that the caller has to wait for the decision after he has placed the call.

5.3.6.5.4 Chief Executive Officer : manager-STD

The communication between the chief executive officer’s operation (callee) and its caller is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

The caller of the operation ‘ceo_mb_decision_(x)’ has to wait until the operation ‘ceo_mb_decision_(x)’ has progressed to the point where the decision has been taken. For this reason the manager STD has the state CC1-3. This state also appears in the external STD as the state ‘make or buy decision taken’.

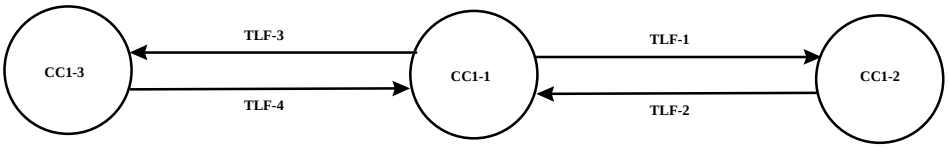


figure 5.42 chief executive officer : manager STD

The notation CCx in the STD stands for ‘Caller(s)-Callee’ and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for ‘Traps Logical Formula’ and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

There is only one caller-callee combination, the callee ‘ceo_mb_decision_(x)’ and its caller ‘mb_request_decision_(x)’ of the class ‘make or buy-meeting’. Because the caller has to wait for the result produced by the callee, the call is modeled by the ‘caller waits’-variant of the caller-callee construct (see SOCCA chapter for this construct). This results in the extra state in the manager STD and the fact that in the state ‘neutral’ either S1 or S2 is prescribed depending on the state the STD was in before it entered the state ‘neutral’.

In the state ‘neutral’ CC and the TLF for the transition leaving the state are :

CC1-1 = {S1 or S2, S3}
TLF-1 = T-1 and T-3 (corresponds with ‘ceo_mb_decision_(x)’ transition in extern STD)
TLF-3 = T-2b

In the state ‘starting_ceo_mb_decision_(x)’ the CC and the TLF for the transition leaving the state are :

CC1-2 = {S2, S3}
TLF-2 = T-2a

In the state ‘make or buy decision taken’ the CC and the TLF for the transition leaving the state are :

CC1-3 = {S1, S4}
TLF-4 = T-4

5.3.6.5.5 Chief Executive Officer : employee-STDs

The manager STD has the following 2 employee STDs. The first is its own internal operation the callee ‘ceo_mb_ decision_(x)’. The second is the caller ‘mb_request_decision_(x)’ from the class make or buy-meeting.

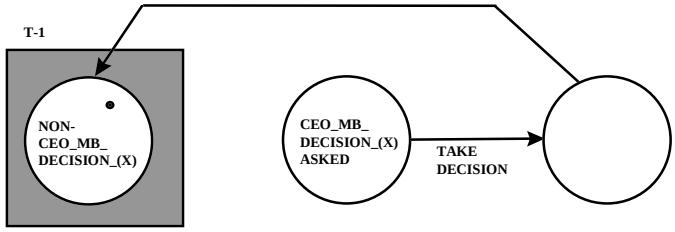


figure 5.43 employee int-ceo_mb_decision_(x) : subprocess S1

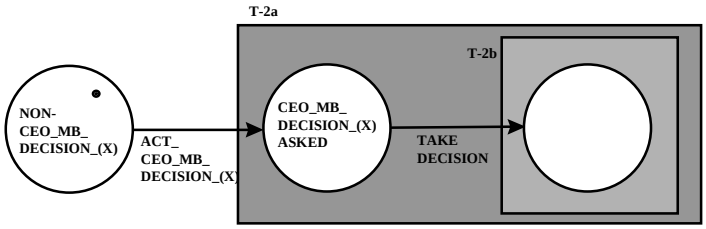


figure 5.44 employee int-ceo_mb_decision_(x) : subprocess S2

The first employee is the own internal operation ‘ceo_mb_decision_(x)’. This employee has two subprocesses S1 and S1 and three traps T-1, T-2a and T-2b. The trap T-2b is to determine that in fact the operation ‘ceo_mb_decision_(x)’ has progressed to the point where a decision has been taken.

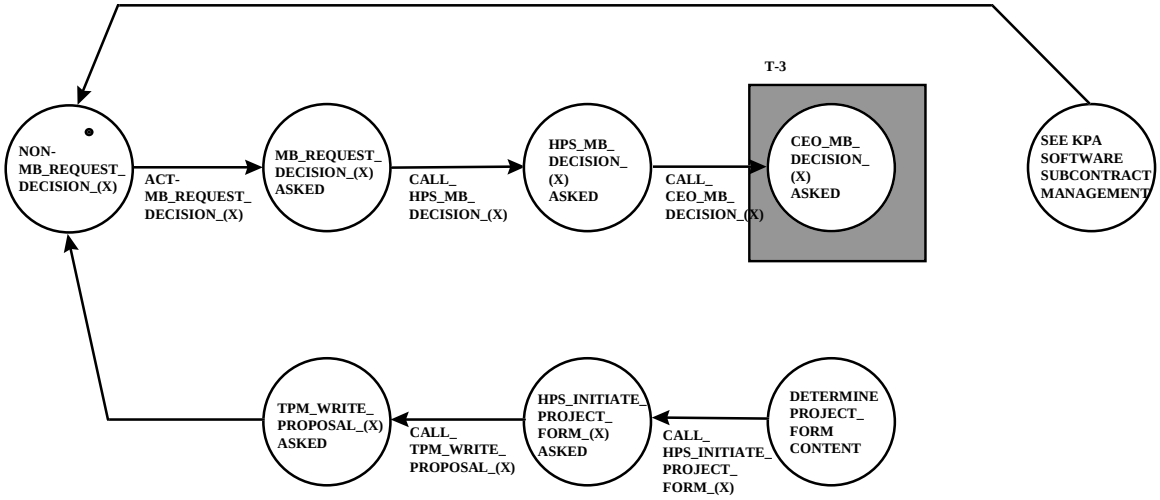


figure 5.45 employee int-mb_request_decision_(x) : subprocess S3

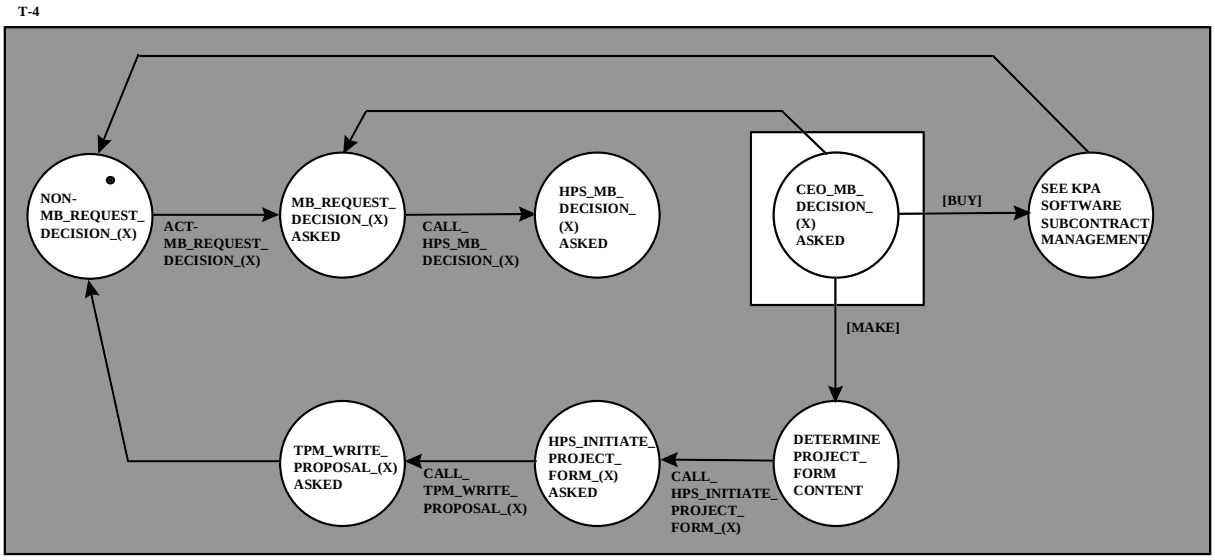


figure 5.46 employee int-mb_request_decision_(x) : subprocess S4

The second employee is the caller operation ‘mb_request_decision_(x)’. This employee has two subprocesses S3 and S4 and two traps T-3 and T-4, according to the caller_callee-construct.

In its neutral state the manager prescribes initially subprocess S1 for the callee and S3 for the caller. When the caller enters its trap T-3, i.e. excutes the call (and the callee is ready in its trap T-1) the manager can make the transition to the state ‘starting_ceo_mb_decision_(x)’. Here it prescribes S2 for the callee and S3 for the caller (i.e the caller stays in subprocess S3). When the callee has entered trap T-2a the manager can transit back to its neutral state. Now it prescribes S2 (instead of S1) for the callee and S3 for the caller. If the callee has finished its action, i.e it has entered trap T2-b, the manager can transit to state ‘make or buy decision taken’. Here it prescribes S1 for the callee and S4 for the caller (i.e. the caller is allowed to proceed). When the caller enters T-4, the manager can transit back to its neutral state. Here it prescribes S1 again for the callee.

5.3.6.6 Head Personnel Section

5.3.6.6.1 Head Personnel Section : external behavior-STD, organizational view

The external behavior STD shows the possible sequences in which the exported operation(s) can be started. The class 'head personnel section' has two operation relevant for the process fragment 'writing project management documents', phase1. The organizational view of the external STD does not show any communication details. Consequently the STD consists of one state 'neutral' in which the head personnel section waits for a call to its operation 'hps_mb_decision_(x)' or a call to its operation 'hps_initiate_project_form_(x) ' If a call has taken place, the head personnel section can make the relevant transition. The head personnel section then comes again in the state 'neutral'. The possible starting sequence specified by this STD is thus any combination of 'hps_mb_decision_(x)'-operations and 'hps_initiate_project_form(x)-operations.

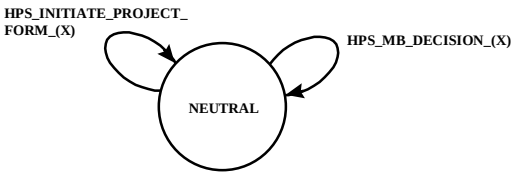


figure 5.47 head personnel section : external behavior STD, organizational view

5.3.6.6.2 Head Personnel Section : external behavior STD, communicative view

The communicative view of the external behavior STD shows the communication details between the external STD and the called operation(s).

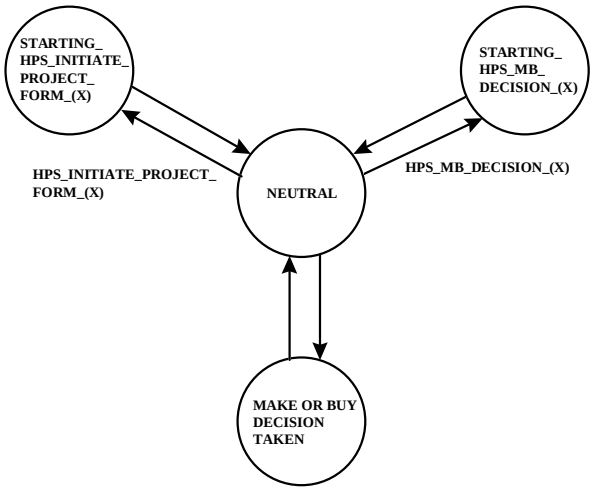


figure 5.48 head personnel section : external behavior STD, communicative view

The STD consists of a neutral state in which the head personnel section waits for a call to one of its two operations 'hps_mb_decision_(x)' and 'hpd_initiate_project_form_(x)'. Furthermore it has two states in which an operation is started. Typically the head personnel section does not wait in a 'starting' state until the called operation is finished, but returns as soon as possible to its neutral state. It can then handle another call, either to its other operation or to the same operation.

The external STD still has another state, 'make or buy decision taken'. The reason for this state is explained in the 'head personnel section officer : manager-STD' paragraph.

The callers of the operations of this class can be found in the import-export diagram. They are given in the the 'import_list' attribute of the 'uses association'.

5.3.6.6.3 Head Personnel Section : internal behavior-STDs

The 2 operations ‘hps_mb_decision’ and ‘hps_initiate_project_form_(x)’ of the head personnel section have the following internal behavior STDs.

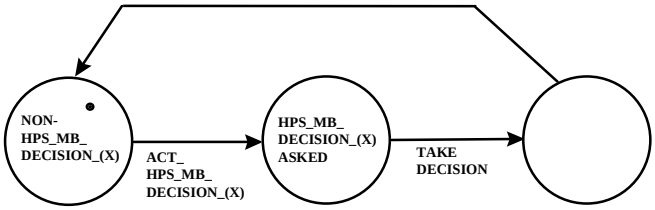


figure 5.49 int-hps_mb_decision_(x) : internal behavior STD

The operation ‘hps_mb_decision_(x)’ has one formal parameter ‘x’. The multiplicity of concurrent executing STD instances is zero or more. This is indicated by the solid circle inside the state ‘non hps_mb_decision_(x)’. A multiplicity of zero or more means that the manager STD can start another instance of the internal STD to execute concurrently with an already executing instance of the internal STD.

The formal parameter of the operation ‘hps_mb_decision_(x)’ is the requirements document on the basis of which the ‘make’ or ‘buy’ decision has to be made by the head presonnel section. After being started the operation (=chief executive officer) takes this decision as an ‘internal action’. The decision is then communicated back to the caller of this operation. This means that the caller has to wait for the decision after he has placed the call.

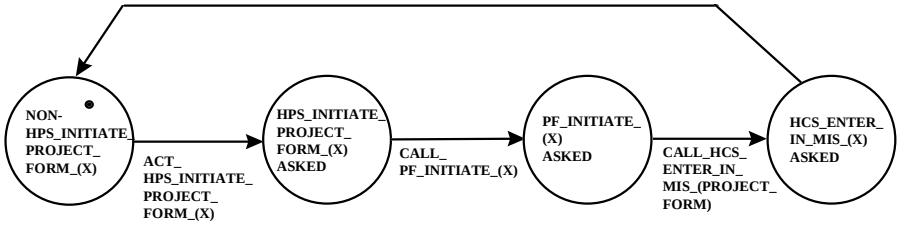


figure 5.50 int-hps_initiate_project_form_(x) : internal behavior STD

The operation ‘hps_initiate_project_form_(x)’ has one formal parameter ‘x’. The multiplicity of concurrent executing STD instances is zero or more.

The formal parameter of the operation ‘hps_initiate_project_form_(x)’ is the content with which the project form is to be initialized. The caller operation has to wait until the project form is created and initialized. The created project form is then given to the head controller section (call_hcs_enter_in_mis(project_form)). The head controller section enters (has it entered by one of its subordinates) the data on the project form in the Management Information System (MIS) of the Waco Business Unit (WBU).

5.3.6.6.4 Head Personnel Section : manager-STD

The communication between the head personnel section’s operation (callee) and its caller is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

The caller of the operation ‘hps_mb_decision_(x)’ has to wait until the operation ‘hps_mb_decision_(x)’ has progressed to the point where the decision has been taken. For this reason the manager STD has the state CPS4. This state also appears in the external STD as the state ‘make or buy decision taken’.

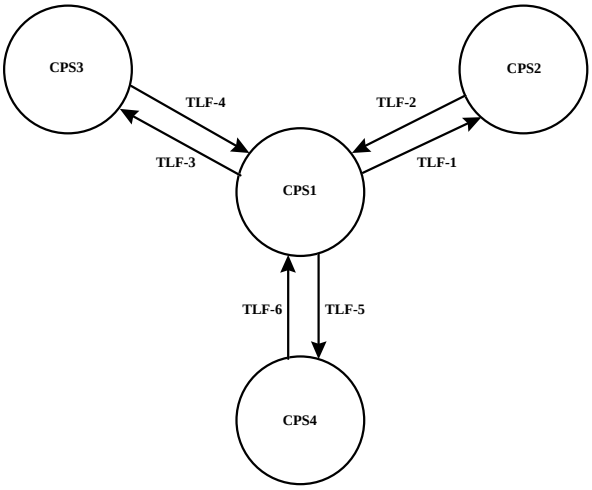


figure 5.51 head personnel section : manager STD

The notation CPSx in the STD stands for Consolidated Prescribed Subprocesses and is a set of CCx's. The notation CCx in the STD stands for 'Caller(s)-Callee' and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for 'Traps Logical Formula' and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

In the state 'neutral' the CPS and the TLFs for the transitions leaving the state are :

CPS1 = {CC1-1, CC2-1}
TLF-1 = T-1 and T-3 (corresponds with 'hps_mb_decision_(x)' transition)
TLF-3 = T-5 and T-7 (corresponds with 'hps_initiate_project_form_x)' transition)
TLF-5 = T-2b

In the state 'starting_hps_mb_decision_(x)' the CPS and the TLFs for the transitions leaving the state are :

CPS2 = {CC1-2, CC2-1}
TLF-2 = T-2a

In the state 'starting_hps_initiate_project_form_(x)_1' the CPS and the TLFs for the transitions leaving the state are :

CPS3 = {CC1-1, CC2-2}
TLF-4 = T-6 and T-8

In the state 'make or buy decision taken' the CPS and the TLFs for the transitions leaving the state are :

CPS4 = {CC1-3, CC2-1}
TLF-6 = T-4

Because the caller 'mb_request_decision' has to wait for the result produced by the callee 'hps_mb_decision_(x)', the call is modeled by the 'caller waits'-variant of the caller-callee construct (see SOCCA chapter for this construct). This results in the fact that in CC1-1 either S1 or S2 is prescribed depending on the state the STD was in before it entered the state 'neutral'.

CC1-1 = {S1 or S2, S3}
CC1-2 = {S2, S3}
CC1-3 = {S1, S4}

In its neutral state the manager prescribes initially subprocess S1 for the callee and S3 for the caller. When the caller enters its trap T-3, i.e. excutes the call (and the callee is ready in its trap T-1) the manager can make the transition to the state 'starting_hps_mb_decision_(x)'. Here it prescribes S2 for the callee and S3 for the caller (i.e the caller stays in subprocess S3). When the callee has entered trap T-2a the manager can transit back to its neutral state. Now it prescribes S2 (instead of S1) for the callee and S3 for the caller. If the callee has finished its action, i.e it has entered trap T2-b, the manager can transit to state 'make or buy decision taken'. Here it prescribes S1 for the callee and S4 for the caller (i.e. the caller is

allowed to proceed). When the caller enters T-4, the manager can transit back to its neutral state. Here it prescribes S1 again for the callee.

The caller-callee combinations for ‘hps_initiate_project_form_(x)’ and its caller ‘mb_request_decision_(x)’ of the class make or buy-meeting are :

CC2-1 = {S5, S7}
CC2-2 = {S6, S8}

At any one time the manager STD of class ‘head personnel section’ will prescribe for its employee ‘mb_request_ decision_(x)’ 2 subprocesses, (S3 or S4) and (S7 or S8). This is because both the callees belong to the same class and are being called by one caller. The actual prescribed subprocess is the intersection of (S3 or S4) and (S7 or S8) (see explanation of intersection of prescribed subprocesses in SOCCA chapter).

5.3.6.6.5 Head Personnel Section : employee-STDs

The manager STD has the following 3 employee STDs. The first is its own internal operation the callee ‘hps_mb_ decision_(x)’. The second is the caller ‘mb_request_decision_(x)’ from the class make or buy-meeting. The third is the own internal operation ‘hps_initiate_project_form_(x). This operation is called by ‘mb_request_decision’. This is the same operation that calls the first employee. So there are only 3 employees. Two callees and one caller.

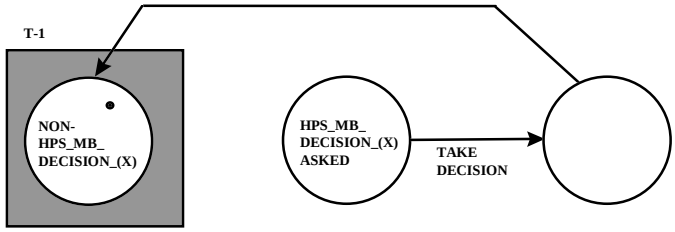


figure 5.52 employee int-hps_mb_decision_(x) : subprocess S1

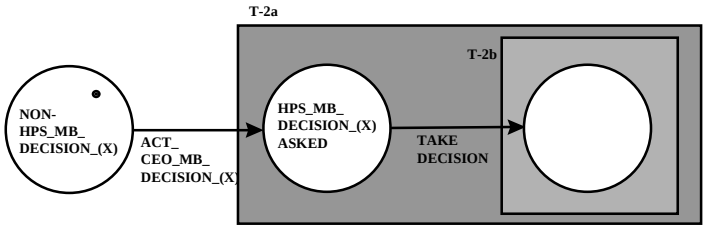


figure 5.53 employee int-hps_mb_decision_(x) : subprocess S2

The first employee is the own internal operation ‘hps_mb_decision_(x)’. This employee has two subprocesses S1 and S1 and three traps T-1, T-2a and T-2b. The trap T-2b is to determine that in fact the operation ‘hps_mb_decision_(x)’ has progressed to the point where a decision has been taken.

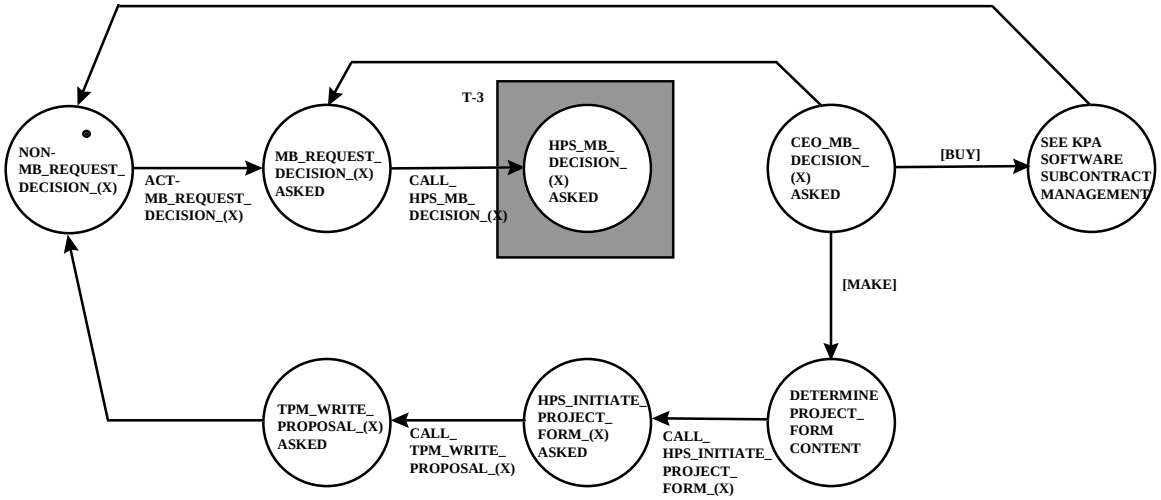


figure 5.54 employee int-mb_request_decision_(x) : subprocess S3

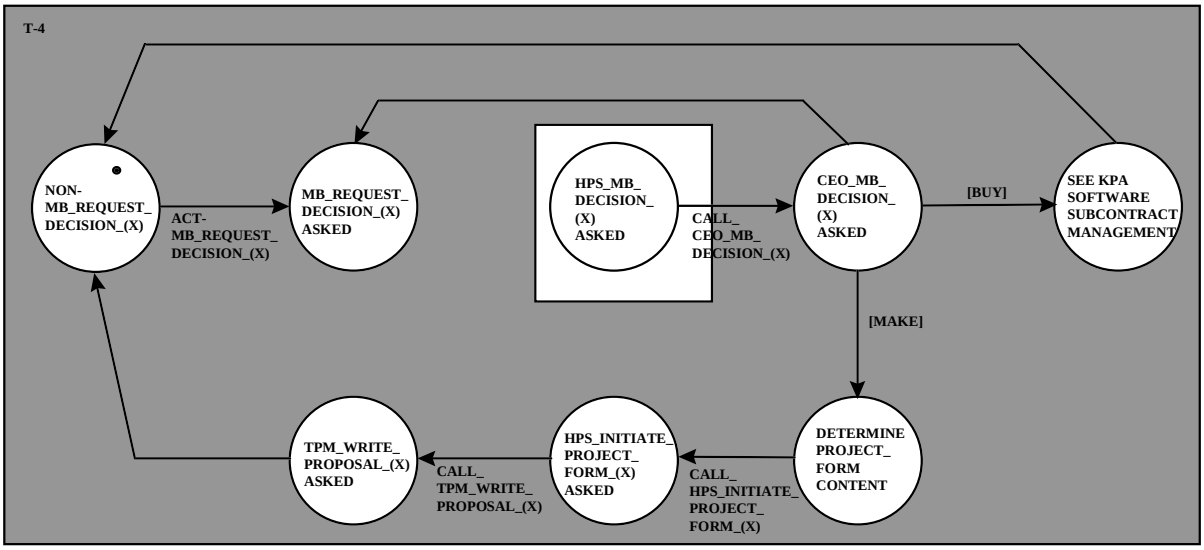


figure 5.55 employee int-mb_request_decision_(x) : subprocess S4

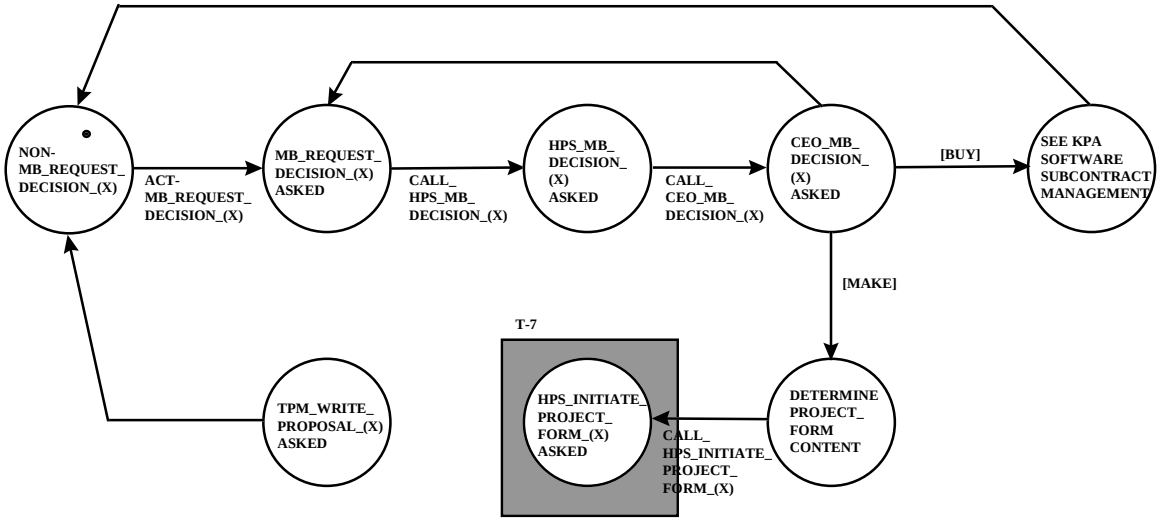


figure 5.56 employee int-mb_request_decision_(x) : subprocess S7

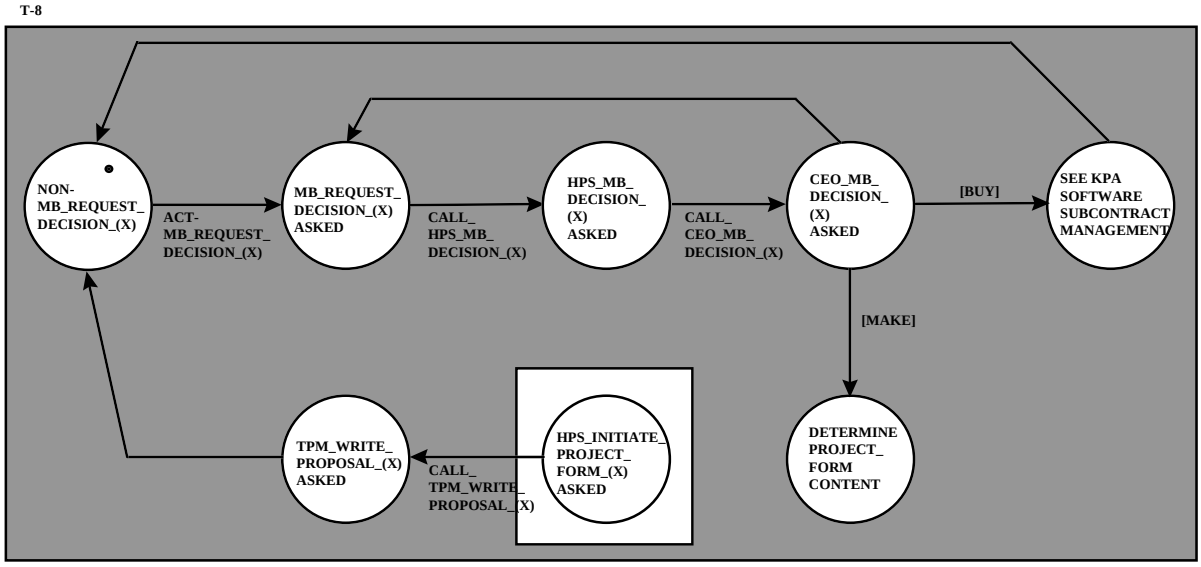


figure 5.57 employee int-mb_request_decision_(x) : subprocess S8

The second employee is the caller operation ‘mb_request_decision_(x)’. This employee has four subprocesses and four traps. The two subprocesses S3 and S4 and the two traps T-3 and T-4 are according to the caller_callee-construct. The callee is in this case ‘hps_mb_decision_(x)’.

The two subprocesses S7 and S8 and the two traps T-7 and T-8 are also according to the caller_callee-construct. The callee is in this case ‘hps_initiate_project_form_(x)’.

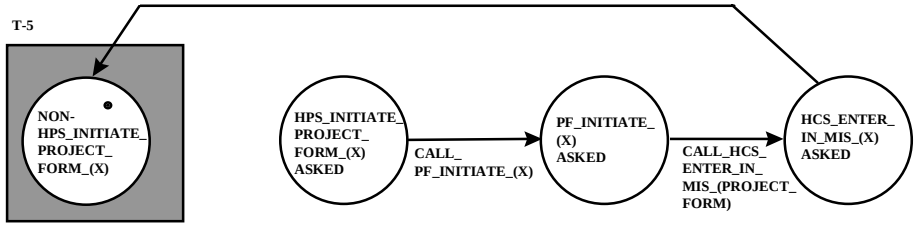


figure 5.58 employee int-hps_initiate_project_form_(x) : subprocess S5

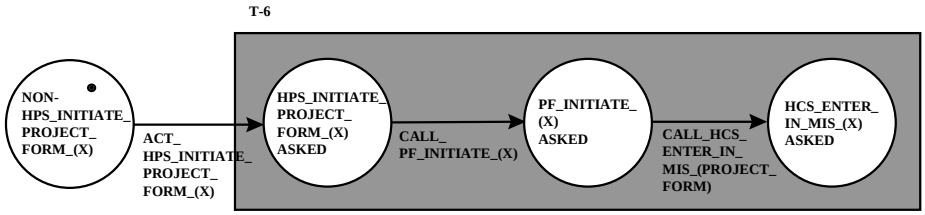


figure 5.59 employee int-hps_initiate_project_form_(x) : subprocess S6

The third employee is the internal callee operation ‘hps_initiate_project_form_(x)’. This employee has two subprocesses S5 and S6 and two traps T-5 and T-6, according to the caller_callee-construct.

5.3.6.7 Project form

5.3.6.7.1 Project form : external behavior-STD, organizational view

The external behavior STD shows the possible sequences in which the exported operation(s) can be started. The class ‘project from’ has only one operation relevant for the process fragment ‘writing project management documents’, phase1. The organizational view of the external STD does not show any communication details. Consequently the STD consists of one state ‘neutral’ in which the project form waits for a call to its exported operation ‘pf_initiate_(x)’. If the call has taken place, the project form can make the transition labeled ‘pf_initiate_(x)’. The project form then comes again in the state ‘neutral’. The possible starting sequence specified by this STD is thus ‘pf_initiate_(x)’, ‘pf_initiate_(x)’, etc.

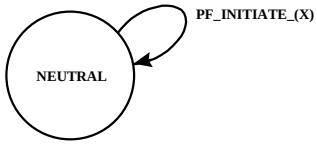


figure 5.60 project form : external behavior STD, organizational view

5.3.6.7.2 Project form : external behavior-STD, communicative view

The communicative view of the external behavior STD shows the communication details between the external STD and the called operation(s). It consists of a neutral state in which the project form waits for a call to its operation ‘pf_initiate_(x)’ and of a state in which he starts this operation. Typically the project form does not wait in this ‘starting’ state until the called operation is finished, but returns as soon as possible to its neutral state. It can then handle another call to the operation ‘pf_initiate_(x)’ before the current execution of the operation has finished.

The external STD still has another state, ‘project form initiated’. The reason for this state is explained in the ‘project form : manager-STD’ chapter.

The callers of the operations of this class can be found in the import-export diagram. They are given in the the ‘import_list’ attribute of the ‘uses association’.

.

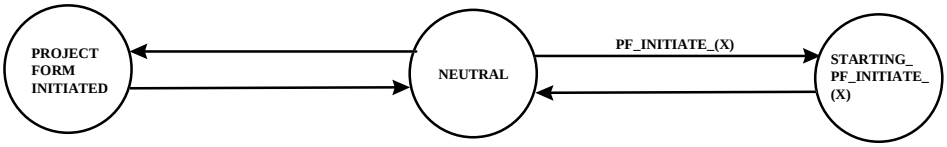


figure 5.61 project form : external behavior STD, communicative view

5.3.6.7.3 Project form : internal behavior-STDs

The 1 operation ‘pf_initiate_(x)’ of the project form has the following internal behavior STD.

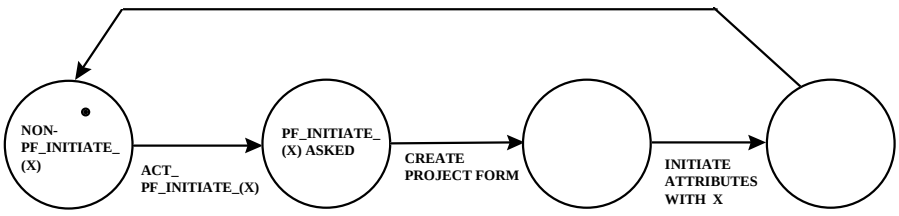


figure 5.62 int-pf_initiate_(x) : internal behavior STD

The operation ‘pf_inititate_(x)’ has one formal parameter ‘x’. The multiplicity of concurrent executing STD instances is zero or more. This is indicated by the solid circle inside the state ‘non pf_initiate_(x)’. A multiplicity of zero or more

means that the manager STD can start another instance of the internal STD to execute concurrently with an already executing instance of the internal STD.

The formal parameter of the operation ‘pf_inititate_(x)’ is the content with which the newly created project form is initialized. The operation first creates a new project form object and then initializes its attributes with the actual parameter value of ‘x’.

5.3.6.7.4 Project form : manager-STD

The communication between the project form’s operations (callees) and their callers is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

The caller of the operation ‘pf_initiate_(x)’ has to wait until the operation ‘pf_initiate_(x)’ has progressed far enough for an initiated project form objrct to be in existance. This is because the caller (the head personnel section operation ‘hps_initiate_project_form_(x)’) needs the project form object (id) in its next action. It uses it as a parameter value in its ‘call_hcs_enter_in_mis_(x)’. For this reason the manager STD has the state CC1-3. This state also appears in the external STD as the state ‘project form initiated’.

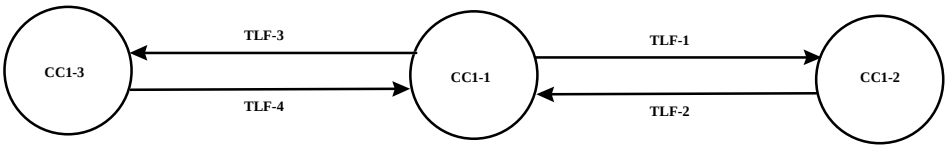


figure 5.63 project form : manager STD

The notation CCx in the STD stands for ‘Caller(s)-Callee’ and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for ‘Traps Logical Formula’ and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

There is only one caller-callee combination, the callee ‘pf_initiate_(x)’ and its caller ‘hps_initiate_project_form_(x)’ of the class head personnel section. Because the caller has to wait for the result produced by the callee, the call is modeled by the ‘caller waits’-variant of the caller-callee construct (see SOCCA chapter for this construct). This results in the extra state in the manager STD and the fact that in the state ‘neutral’ either S1 or S2 is prescribed depending on the state the STD was in before it entered the state ‘neutral’.

In the state ‘neutral’ CC and the TLF for the transition leaving the state are :

CC1-1 = {S1 or S2, S3}
TLF-1 = T-1 and T-3 (corresponds with ‘rd_initiate_(x)’ transition in extern STD)
TLF-3 = T-2b

In the state ‘starting_pf_inititate_(x)’ the CC and the TLF for the transition leaving the state are :

CC1-2 = {S2, S3}
TLF-2 = T-2a

In the state ‘project form initiated’ the CC and the TLF for the transition leaving the state are :

CC1-3 = {S1, S4}
TLF-4 = T-4

5.3.6.7.5 Project form : employee-STDs

The manager STD has 2 employee STDs. These are the callee ‘pf_initiate_(x)’ and its caller ‘hps_initiated_ project_form_(x)’ of the class customer. The calling of ‘pf_initiate_(x)’ is modeled by the ‘caller waits’-variant of the caller-callee construct.

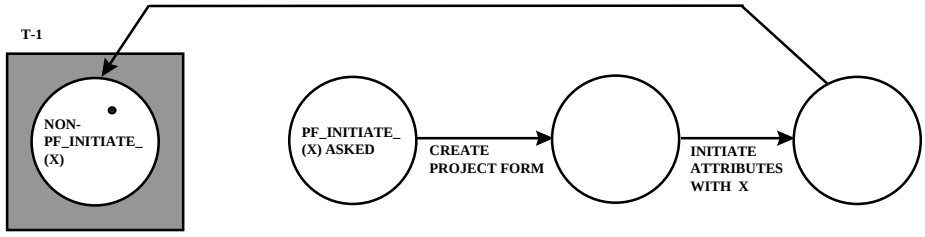


figure 5.64 employee int-pf_initiate_(x) : subprocess S1

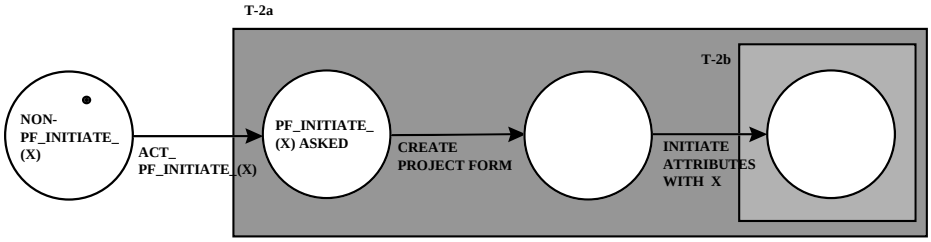


figure 5.65 employee int-pf_initiate_(x) : subprocess S2

The first employee is the own internal operation ‘pf_initiate_(x)’. This employee has two subprocesses S1 and S1 and three traps T-1, T-2a and T-2b. The trap T-2b is to determine that in fact the operation ‘pf_initiate_(x)’ has created and initialized a new project form.

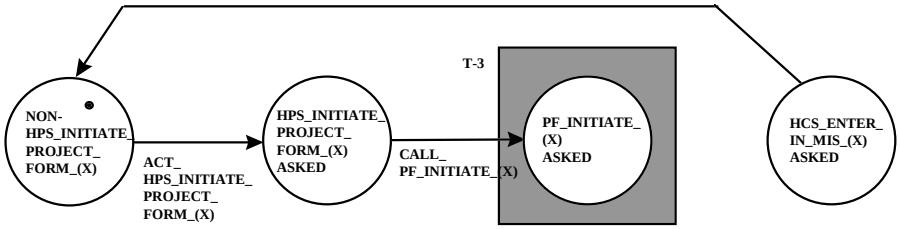


figure 5.66 employee int-hps_initiate_project_form_(x) : subprocess S3

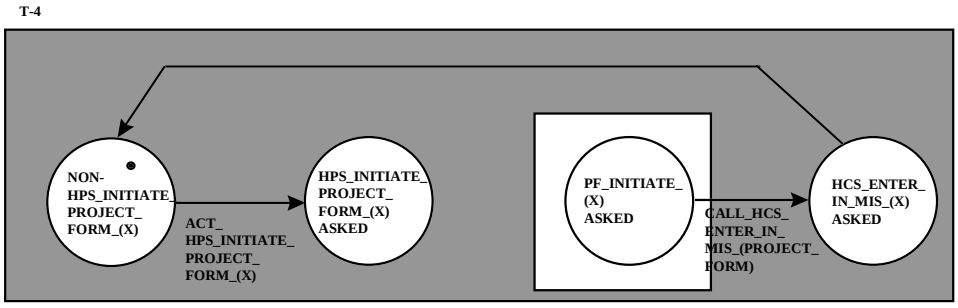


figure 5.67 employee int-hps_initiate_project_form_(x) : subprocess S4

The second employee is the caller operation ‘hps_initiate_project_form_(x)’. This employee has two subprocesses S3 and S4 and two traps T-3 and T-4 according to the caller_callee-construct.

In its neutral state the manager prescribes initially subprocess S1 for the callee and S3 for the caller. When the caller enters its trap T-3, i.e. excutes the call (and the callee is ready in its trap T-1) the manager can make the transition to the state ‘starting_pf_initiate_(x). Here it prescribes S2 for the callee and S3 for the caller (i.e the caller stays in subprocess S3). When the callee has entered trap T-2a the manager can transit back to its neutral state. Now it prescribes S2 (instead of S1) for the callee and S3 for the caller. If the callee has finished its action, i.e it has entered trap T2-b, the manager can transit

to state ‘project form initiated’. Here it prescribes S1 for the callee and S4 for the caller (i.e. the caller is allowed to proceed). When the caller enters T-4, the manager can transit back to its neutral state. Here it prescribes S1 again for the callee.

5.3.6.8 Head controller section

5.3.6.8.1 Head controller section : external behavior-STD, organizational view

The external behavior STD shows the possible sequences in which the exported operation(s) can be started. The class 'head controller section' has only one operation relevant for the process fragment 'writing project management documents', phase1. The organizational view of the external STD does not show any communication details. Consequently the STD consists of one state 'neutral' in which the head controller section waits for a call to its exported operation 'hcs_enter_ in_mis_(x)'. If the call has taken place, the head controller section can make the transition labeled 'hcs_enter_in_mis _(x)'. The head controller section then comes again in the state 'neutral'. The possible starting sequence specified by this STD is thus 'hcs_enter_in_mis_(x)', 'hcs_enter_in_mis_(x)', etc.

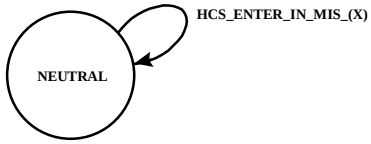


figure 5.68 head controller section : external behavior STD, organizational view

5.3.6.8.2 Head controller section : external behavior-STD, communicative view

The communicative view of the external behavior STD shows the communication details between the external STD and the called operation(s).

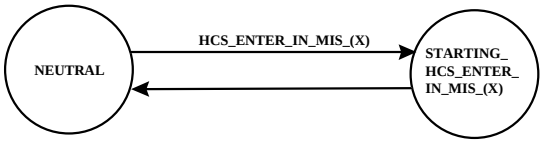


figure 5.69 head controller section : external behavior STD, communicative view

This external behavior STD has two states. In the state neutral the head controller section waits until a call has been placed to its operation 'hcs_enter_in_mis_(x). If a call has been made, the manager STD can (and eventually will) transit to the state 'starting_hcs_enter_in_mis_(x)'. If the operation has been started the manager can transit back to the state neutral.

The callers of the operations of this class can be found in the import-export diagram. They are given in the the 'import_list' attribute of the 'uses association'.

5.3.6.8.3 Head controller section : internal behavior-STDs

The head controller section has 1 operation : 'hcs_enter_in_mis_(x)'. This operation has the following internal behavior STD.

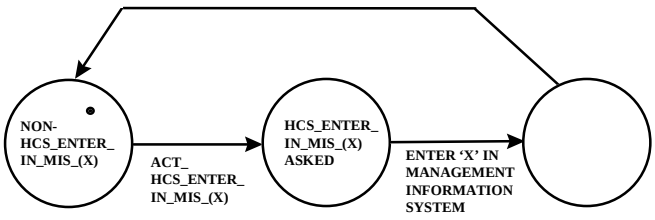


figure 5.70 int-hcs_enter_in_mis_(x) : internal behavior STD

The operation 'hcs_enter_in_mis_(x)' has one formal parameter 'x'. The multiplicity of concurrent executing STD instances is zero or more. This is indicated by the solid circle inside the state 'non hcs_enter_in_mis_(x)'. A multiplicity of

zero or more means that the manager STD can start another instance of the internal STD to execute concurrently with an already executing instance of the internal STD.

The formal parameter of the operation 'hcs_enter_in_mis_(x)' is any information that needs to be entered in the Management Information System (MIS) of the Waco Business Unit (WBU). With this operation the head controller section enters the data in the MIS (or let one of his subordinates do it).

5.3.6.8.4 Head controller section : manager-STD

The communication between the head controller's operations (callees) and their callers is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

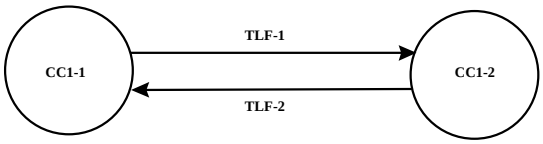


figure 5.71 head controller section : manager STD

The notation CCx in the STD stands for 'Caller(s)-Callee' and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for 'Traps Logical Formula' and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

There is only one caller-callee combination, the callee 'hcs_enter_in_mis_(x)' and its caller 'hps_initiate_project _form_(x)' of the class head personnel section. The calling is modeled by the 'caller does not wait'-variant of the caller-callee construct (see SOCCA chapter for this construct).

In the state 'neutral' CC and the TLF for the transition leaving the state are :

CC1-1 = {S1, S3}
TLF-1 = T-1 and T-3 (corresponds with 'hcs_enter_in_mis_(x)' transition in extern STD)

In the state 'starting_hcs_enter_in_mis_(x)' the CC and the TLF for the transition leaving the state are :

CC1-2 = {S2, S4}
TLF-2 = T-2 and T-4

5.3.6.8.5 Head controller section : employee-STDs

The manager STD has 2 employee STDs. These are the callee 'hcs_enter_in_mis_(x)' and its caller 'hps_initiate_ project_form_(x)' of the class head personnel section.

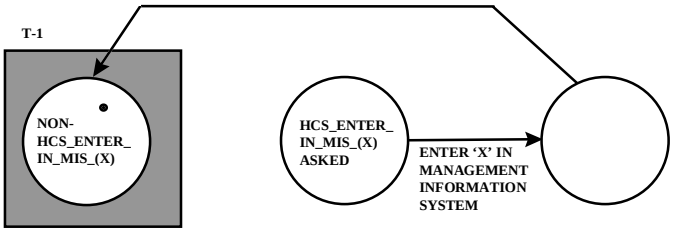


figure 5.72 employee int-hcs_enter_in_mis_(x) : subprocess S1

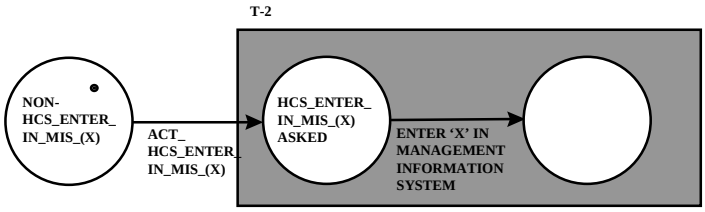


figure 5.73 employee int-hcs_enter_in_mis_(x) : subprocess S2

The first employee is the own internal operation 'hcs_enter_in_mis_(x)'. This employee has two subprocesses S1 and S1 and two traps T-1 and T-2 according to the caller_callee-construct.

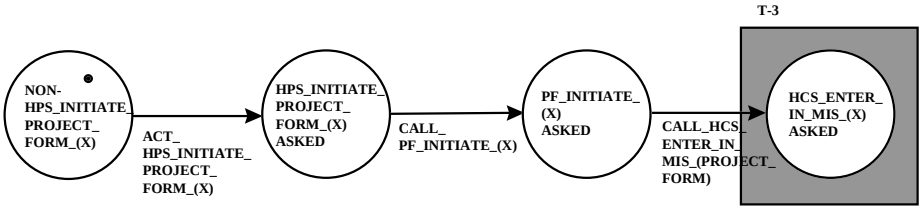


figure 5.74 employee int-hps_initiate_project_form_(x) : subprocess S3

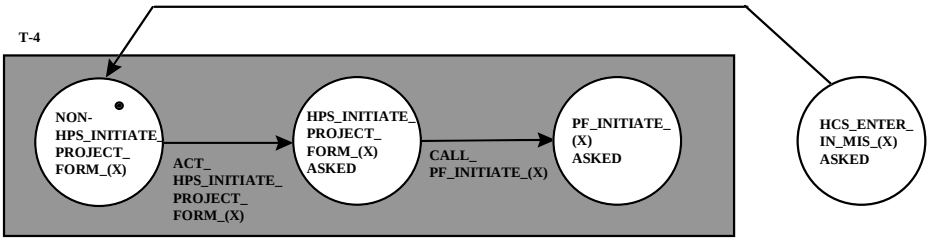


figure 5.75 employee int-hps_initiate_project_form_(x) : subprocess S4

The second employee is the caller operation 'hps_initiate_project_form_(x)'. This employee has two subprocesses S3 and S4 and two traps T-3 and T-4.

The manager prescribes initially subprocess S3 for the calling employee, it is waiting for the call. When the caller enters its trap T-3, i.e. excutes the call (and the callee is ready in its trap T-1) the manager can make the transition to the next state. Here it prescribes S4 for the caller, thereby allowing it to proceed in its next subprocess. This has the effect that the caller does not wait for the result of the called operation but proceeds right away after the manager has started the called operation.

When the callee has entered its trap T-2 and the caller has entered his trap T-4, the manager can transit back to the state 'neutral'.

5.3.6.9 Technical_Project_Manager

5.3.6.9.1 Technical_Project_Manager : external behavior-STD, organizational view

The external behavior STD shows the possible sequences in which the exported operation(s) can be started. The class 'technical project manager' has only one operation relevant for the process fragment 'writing project management documents', phase 1. The organizational view of the external STD does not show any communication details. Consequently the STD consists of one state 'neutral' in which the technical project manager waits for a call to its exported operation 'tpm_write_proposal_(x)'. If the call has taken place, the technical project manager can make the transition labeled 'tpm_write_proposal_(x)'. The technical project manager then comes again in the state 'neutral'. The possible starting sequence specified by this STD is thus 'tpm_write_proposal_(x)', 'tpm_write_proposal_(x)', etc.

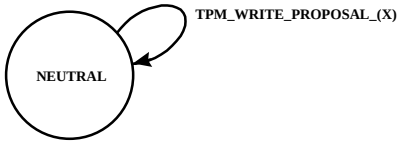


figure 5.76 technical_project_manager : external behavior STD, organizational view

5.3.6.9.2 Technical_Project_Manager : external behavior-STD, communicative view

The communicative view of the external behavior STD shows the communication details between the external STD and the called operation(s).

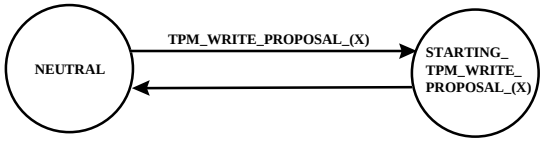


figure 5.77 technical_project_manager : external behavior STD, communicative view

This external behavior STD has two states. In the state neutral the technical project manager waits until a call has been placed to its operation 'tpm_write_proposal_(x)'. If a call has been made, the manager STD can (and eventually will) transit to the state 'starting_tpm_write_proposal_(x)'. If the operation has been started the manager can transit back to the state neutral.

The callers of the operations of this class can be found in the import-export diagram. They are given in the the 'import_list' attribute of the relevant 'uses association'.

5.3.6.9.3 Technical_Project_Manager : internal behavior-STDs

The technical project manager has 1 operation relevant for phase 1 of the process fragment 'writing project management documents' : 'tpm_write_proposal_(x)'. Only a 'view' of the internal STD is shown. The full STD is described in the modeling of phase 2. A 'view' on an STD shows only the transitions and (aggregated) states that are important for a certain 'user' of that STD. (See also the explanation on 'views' in the SOCCA chapter). The view shown here is the view of the external STD on this internal STD. Only the one transition 'act_tpm_write_proposal_(x)' is relevant to the external STD. And only two states are seen by the external STD. Either the STD is non-active or the STD is executing.

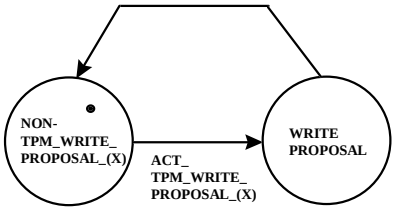


figure 5.78 int-tpm_write_proposal_(x) : internal behavior STD, view

The operation ‘tpm_write_proposal_(x)’ has one formal parameter ‘x’. The multiplicity of concurrent executing STD instances is zero or more. This is indicated by the solid circle inside the state ‘non tpm_write_proposal_(x)’. A multiplicity of zero or more means that the manager STD can start another instance of the internal STD to execute concurrently with an already executing instance of the internal STD.

The formal parameter of the operation ‘tpm_write_proposal_(x)’ is the requirements document. Based on this requirements document the technical project manager writes a (preliminary version of) a Software Development Plan (SDP), a Project Contract (PC) and an Internal Resources Allocation document (IRA document). The writing of the SDP, PC and IRA document is modeled in phase 2 of the process fragment ‘writing project management documents’.

The preliminary version of the SDP and PC constitute the proposal.

5.3.6.9.4 Technical_Project_Manager : manager-STD

The communication between the technical project manager’s operations (callees) and their callers is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

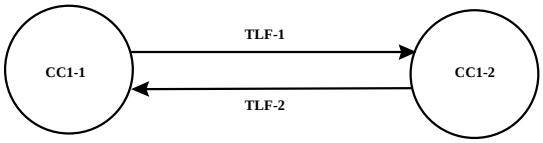


figure 5.79 technical_project_manager : manager STD

The notation CCx in the STD stands for ‘Caller(s)-Callee’ and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for ‘Traps Logical Formula’ and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

There is only one caller-callee combination, the callee ‘tpm_write_proposal_(x)’ and its caller ‘mb_request_ decision_(x)’ of the class ‘make or buy-meeting’. The calling is modeled by the ‘caller does not wait’-variant of the caller-callee construct (see SOCCA chapter for this construct).

In the state ‘neutral’ the CC and the TLF for the transition leaving the state are :

CC1-1 = {S1, S3}
TLF-1 = T-1 and T-3 (corresponds with ‘tpm_write_proposal_(x)’ transition in extern STD)

In the state ‘starting_tpm_write_proposal_(x)’ the CC and the TLF for the transition leaving the state are :

CC1-2 = {S2, S4}
TLF-2 = T-2 and T-4

5.3.6.9.5 Technical_Project_Manager : employee-STDs

The manager STD has 2 employee STDs. These are the callee ‘tpm_write_proposal_(x)’ and its caller ‘mb_request_ decision_(x)’ of the class make or buy-meeting.

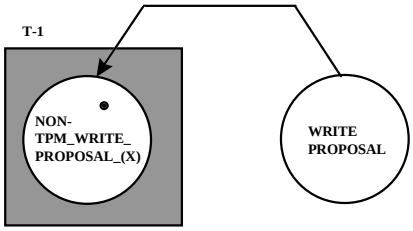


figure 5.80 employee int-tpm_write_proposal_(x) : subprocess S1

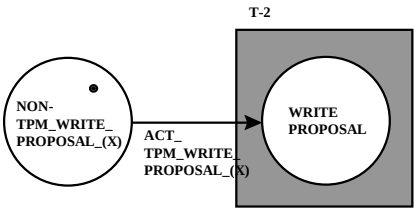


figure 5.81 employee int-tpm_write_proposal_(x) : subprocess S2

The first employee is the own internal operation ‘tpm_write_proposal_(x)’. This employee has two subprocesses S1 and S2 and two traps T-1 and T-2 according to the caller_callee-construct. The internal STD can be a long time in its subprocess S2. This is the actual ‘writing’ of the proposal by the technical project manager.

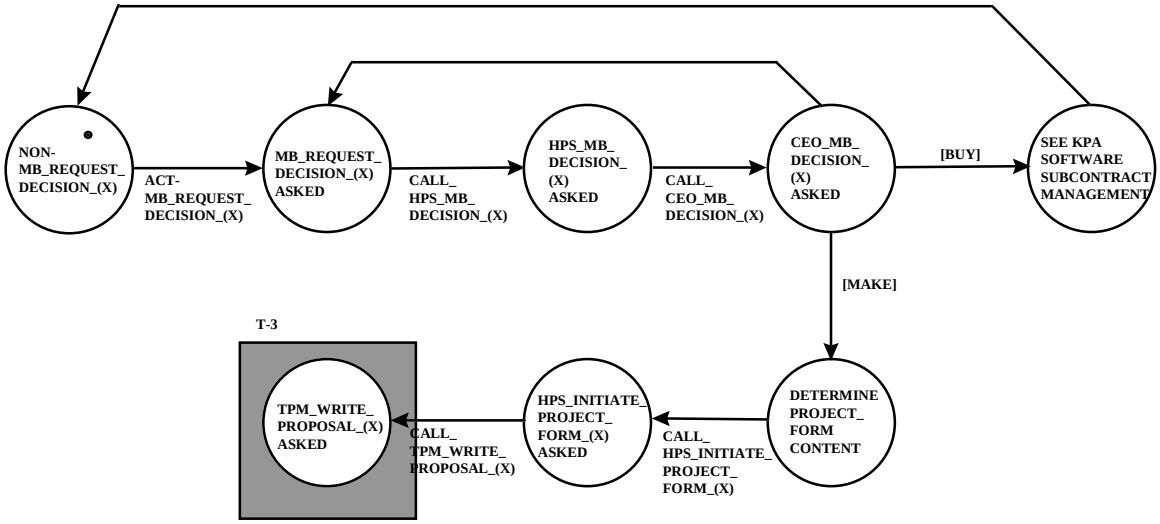


figure 5.82 employee int-mb_request_decision_(x) : subprocess S3

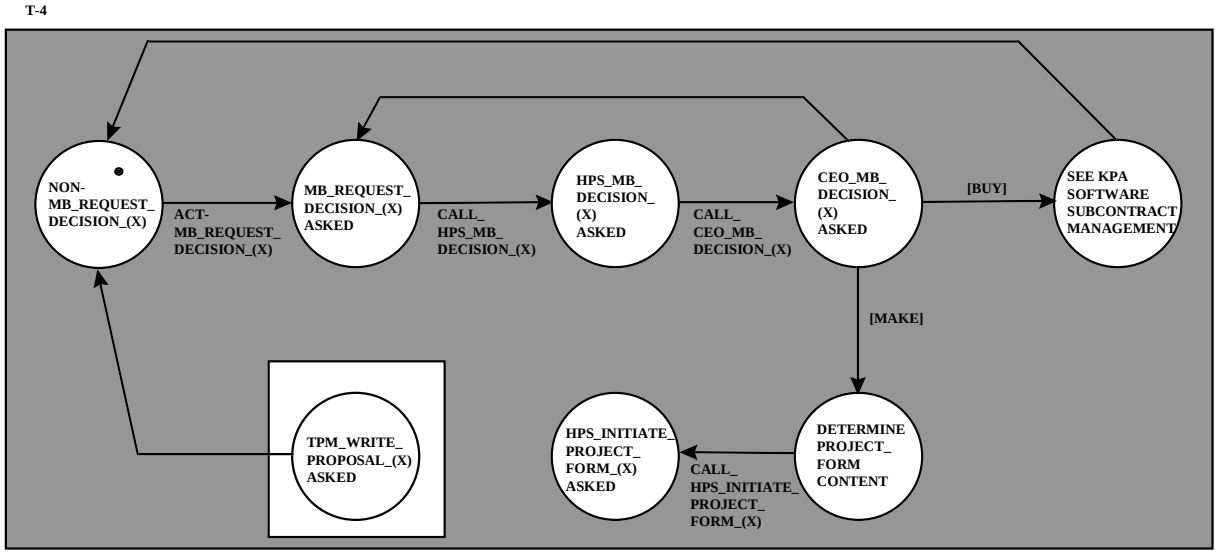


figure 5.83 employee int-mb_request_decision_(x) : subprocess S4

The second employee is the caller operation ‘mb_request_decision_(x)’. This employee has two subprocesses S3 and S4 and two traps T-3 and T-4 according to the caller_callee-construct.

The manager prescribes initially subprocess S3 for the calling employee, it is waiting for the call. When the caller enters its trap T-3, i.e. excutes the call (and the callee is ready in its trap T-1) the manager can make the transition to the next state. Here it prescribes S4 for the caller, thereby allowing it to proceed in its next subprocess. This has the effect that the caller does not wait for the result of the called operation but proceeds right away after the manager has started the called operation.

When the callee has entered its trap T-2 and the caller has entered his trap T-4, the manager can transit back to the state ‘neutral’.

SOFTWARE PROCESS MODELING in SOCCA

student : H.G.Brugman
studentnr : 9105506
version : 0.10

5. Key Process Area ‘Software Project Planning’

figure 5.1figure 5.2 figure 5.3figure 5.4 figure 5.5figure 5.6 figure 5.7figure 5.8 figure 5.9figure 5.10
figure 5.11figure 5.12 figure 5.13figure 5.14 figure 5.15figure 5.16 figure 5.17figure 5.18 figure 5.19figure 5.20
figure 5.21figure 5.22 figure 5.23figure 5.24 figure 5.25figure 5.26 figure 5.27figure 5.28 figure 5.29figure 5.30
figure 5.31figure 5.32 figure 5.33figure 5.34 figure 5.35figure 5.36 figure 5.37figure 5.38 figure 5.39figure 5.40
figure 5.41figure 5.42 figure 5.43figure 5.44 figure 5.45figure 5.46 figure 5.47figure 5.48 figure 5.49figure 5.50
figure 5.51figure 5.52 figure 5.53figure 5.54 figure 5.55figure 5.56 figure 5.57figure 5.58 figure 5.59figure 5.60
figure 5.61figure 5.62 figure 5.63figure 5.64 figure 5.65figure 5.66 figure 5.67figure 5.68 figure 5.69figure 5.70
figure 5.71figure 5.72 figure 5.73figure 5.74 figure 5.75figure 5.76 figure 5.77figure 5.78 figure 5.79figure 5.80
figure 5.81figure 5.82 figure 5.83

5.3.7 State Transition Diagrams - Phase 2

Phase 2, ‘writing and consultation’, of the process fragment ‘writing project management documents’ (partly) models the behavior of the following classes :

- technical project manager
- customer
- account manager
- quality assurance adviser
- head production section
- head support section
- project management document
- project meeting minus

5.3.7.1 Technical_Project_Manager

5.3.7.1.1 Technical_Project_Manager : external behavior-STD, organizational view

The external behavior STD shows the possible sequences in which the exported operation(s) can be started. The class 'technical project manager' has four operations relevant for the process fragment 'writing project management documents', phase 2. The first is the operation 'tpm_write_proposal_(x)'. The calling of this operation is described in phase 1 and will not be duplicated here. The other three operations are 'tpm_write_proj_man_doc_(x,y)', 'tpm_perform_estimate_(x)' and 'tpm_confer_estimate_(x)'. The organizational view of the external STD does not show any communication details. Consequently the STD consists of one state 'neutral' in which the technical project manager waits for a call to its exported operations The possible starting sequence specified by this STD is any combination of 'tpm_write_proj_man_doc_(x)'-operations, 'tpm_perform_ estimate_(x)'-operations and 'tpm_confer_estimate_(x)'-operations.

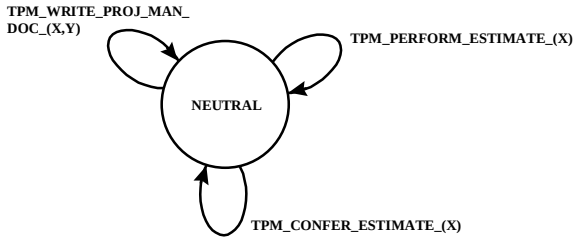


figure 5.84 technical_project_manager : external behavior STD, organizational view

5.3.7.1.2 Technical_Project_Manager : external behavior-STD, communicative view

The communicative view of the external behavior STD shows the communication details between the external STD and the called operation(s).

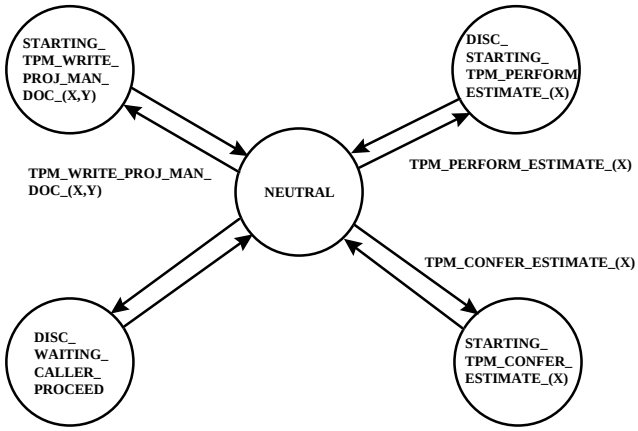


figure 5.85 technical_project_manager : external behavior STD, communicative view

The STD consists of a neutral state in which the technical project manager waits for a call to one of its three operations 'tpm_write_proj_man_doc_(x,y)', 'tpm_perform_estimate_(x)' and 'tpm_confer_estimate_(x)'. Furthermore it has three states in which an operation is started. Typically the technical project manager does not wait in a 'starting' state until the called operation is finished, but returns as soon as possible to its neutral state. It can then handle another call, either to its other operations or to the same operation.

The operation 'tpm_perform_estimate_(x)' can be called by two callers, 'hprs_second_estimate_(x)' and 'tpm_write_ proposal_(x)'. To distinguish between those callers, the starting state for the operation 'tpm_perform_ estimate_(x)' is a 'discriminator' state, 'disc_starting_ tpm_perform_estimate_(x)'. (See the SOCCA chapter for an explanation af the discriminator construct.)

The fifth state is ‘disc_waiting_caller_proceed’. This is an aggregate state in which the manager determines which waiting caller may proceed.

There are four callers for the three callee operations. The operation ‘tpm_perform_estimate_(x)’ has two callers. Both callers have to wait for a result after they have called ‘tpm_perform_estimate_(x)’. If the manager detects that the execution of ‘tpm_perform_estimate_(x)’ has progressed far enough (i.e. it as produced a return result), it can transit to ‘disc_waiting_caller_proceed’. In this state it is determined for wich of the two callers the result is intended. This caller will then be allowed to proceed and the manager transits back to ‘neutral’.

The operation ‘tpm_confer_estimate_(x)’ has one caller, namely the operation ‘tpm_write_proposal_(x)’ of the same tpm-object. After the call the caller may proceed a little further (one state to be precise) and then has to wait until the callee returns a result (i.e the required estimate). If the manager detects that the callee has progressed far enough in its execution, it can and will transit to the state ‘disc_waiting_caller_proceed’ and it will allow the caller to proceed.

The operation ‘tpm_write_proj_man_doc_(x,y)’ can be called up to three times by the same caller ‘tpm_write_ proposal_(x)’. The caller has to wait for the result of all (max 3) concurrently running instances. Initialy the tpm has to write all three project management documents. Then there will be three instances of the operation ‘tpm_write_proj_man _doc_(x,y)’ running concurrently. Later on it may be that the tpm has to correct one or more of the documents. Then 1, 2 or 3 instances of the operation ‘tpm_write_proj_man_doc_(x,y)’ will run concurrently.

There is only one physical external STD and consequently only one manager STD. The manager STD prescribes in its states subprocesses for the internal STDs. These subprocesses apply to only one instance of an internal STD and its caller. Nevertheless the manager can prescribe a different behavior restriction for every instance of a callee and also for its caller. This phenomenon, which is not readily expressed in the current notation conventions of SOCCA, is called the ‘role’ of a manager STD. A manager STD can play different ‘roles’ with respect to different instances of the same internal STD which are called by the same caller. In every role it prescribes a subprocess to its employees. The actual subprocess of the caller employee is then the intersection of the subprocesses prescribed by the different roles of the manager. In fact just if there were more than one manager for this employee.

If there are for example 3 concurrent executing instances of the operation ‘tpm_write_proj_man_doc_(x,y)’, the manager has three roles. If the manager detects that one of the instances has produced a result, it will transit to the state ‘disc_waiting_caller_proceed’. Then it decides for which ‘role’ of the manager a change in precribed subprocesses is in order. It will aplly this change. Thus in one role of the manager the caller is allowed to proceed and in the two other roles it still has to wait. Consequently the caller does not proceed. Only after all three ‘roles’ of the manager allow the caller to proceed, it can actually do so.

5.3.7.1.3 Technical_Project_Manager : internal behavior-STDs

The technical project manager has 4 operations relevant for phase 2 of the process fragment ‘writing project management documents’ : ‘tpm_write_proposal_(x)’, ‘tpm_write_proj_man_doc_(x,y)’, ‘tpm_perform_estimate_(x)’ and ‘tpm_ confer_estimate_(x)’. These operation have the following internal behavior STDs.

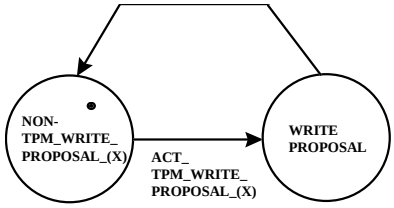


figure 5.86 int-tpm_write_proposal_(x) : internal behavior STD, view

In the description of the operation ‘tpm_write_proposal_(x)’ in phase 1, only a view of its internal STD was given (see figure above). Here the complete STD is given. To indicate how the view corresponds to the full STD, the aggregated state ‘write proposal’ is shown in the full STD (see figure below).

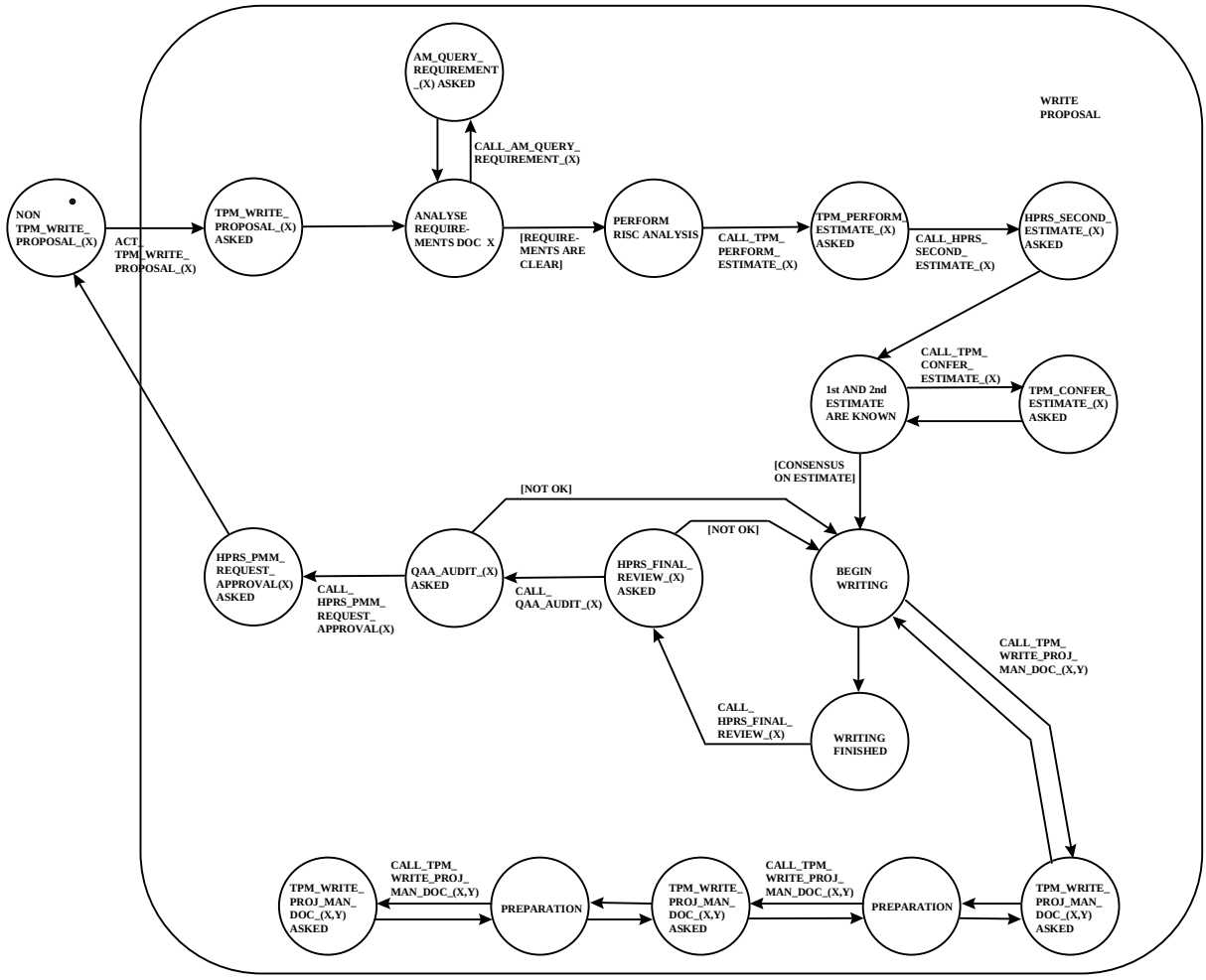


figure 5.87 int-tpm_write_proposal_(x) : internal behavior STD

The operation ‘tpm_write_proposal_(x)’ has one formal parameter ‘x’. The multiplicity of concurrently executing STD instances is zero or more. This is indicated by the solid circle inside the state ‘non tpm_write_proposal_(x)’.

The formal parameter of the operation ‘tpm_write_proposal_(x)’ is the requirements document. Before the technical project manager actually starts writing he first analyses the requirements document on consistency and completeness. If he has any questions in this respect, he queries the customer via the account manager (call_am_query_requirement_(x)). If the requirements are clear to him, he takes the guarded transition ’[requirements are clear]’. He then performs a risk analysis. After that he estimates the necessary person-hours for the project. To do this he calls his own operation ‘tpm_ perform_estimate_(x)’. Immediately thereafter he informs his section head that he needs a second estimate (call_hprs_ second_estimate). The tpm then has to wait for the the result of his own estimate and the result of the second estimate, which is returned to him via his section head.

If the estimates are significantly different, the tpm confers with the second estimator (he calls the operation ‘tpm_ confer_estimate_(x)’ of another tpm-object). When a consensus is reached about a final estimate, the tpm proceeds by writing a software development plan, a ‘internal resources allocation’-document and a project contract. He does this by calling his own internal operation ‘tpm_write_proj_man_doc_(x,y) three times. The parameter ‘x’ is in those three calls the same, the requirements document. The parameter ‘y’ is different for the three calls. ‘y’ signifies the type of project management document that has to be written. ‘y’ is ‘sdp’, ‘ira’ and ‘pc’ respectively. The three instances of the operation ‘tpm_write_proj_man_doc_(x,y)’ run concurrently.

The tpm has to wait until all three instances of the operation ‘tpm_write_proj_man_doc_(x)’ have progressed to a point in their execution where the writing of the respective project management document is finished.

The technical project manager then presents the final version of all three project management documents to his section head for review (call_hprs_final_review_(x)). If the section head has still some comments at this stage (the transition [not ok]), the tpm updates the documents accordingly and presents them again for final review,

If the section head of the tpm approves of the documents, the technical project manager presents them for a quality audit to the quality assurance adviser (call_qaa_audit_(x)). If the quality assurance adviser has any comments (the transition [not ok]), the technical project manager updates the documents accordingly and presents them again for final review to his section head and then again to the quality assurance adviser for a quality audit.

If the audit is ok, then the tpm presents the project management documents to his section head for subsequent approval in the next 'project meeting minus' (call_hprs_pmm_request_approval_(x)). Because all participants in the 'project meeting minus' are already (intensively) consulted by the technical project manager during the writing of the documents, the 'project meeting minus' will always give an OK on these documents.

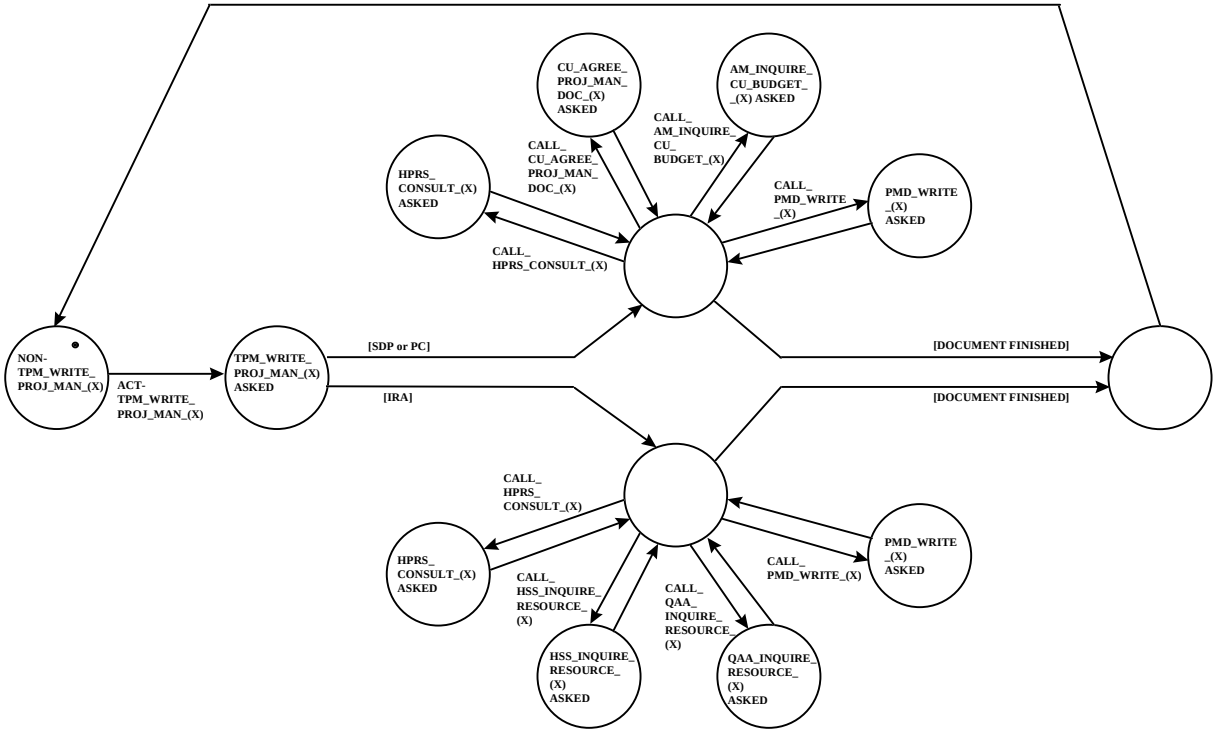


figure 5.88 int-tpm_write_proj_man_doc_(x) : internal behavior STD

The operation 'tpm_write_proj_man_doc_(x,y)' has two formal parameters. The parameter 'x' is the requirements document on which the project management document is to be based. The parameter 'y' indicates which project management document must be produced. If 'y' is 'sdp' then a software development plan must be written. If 'y' is 'pc' then a project contract must be written. If 'y' is 'ira' then an internal resources allocation-document must be made. The multiplicity of concurrent executing STD instances is zero or more. This is indicated by the solid circle inside the state 'non tpm_write_proposal_(x)'.

The STD has two distinct parts. One is for writing a software development plan (SDP) or a project contract (PC). The other part deals with the making of an internal resources allocation-document (IRA-document). The guarded transitions '[SDP or PC]' and '[IRA]' model this 'split' in the STD. The production of a project management document is an iterative process of writing (call_pmd_write_(x)) and consulting.

When producing an SDP or PC the technical project manager consults with his section head, the customer and the account manager. His section head is informed on the progress of the writing and consulted on technical matters (call_ hprs_consult_(x)). The budget of the customer is checked with the account manager (call_am_inquire_cu_budget_(x)). The tpm also determines regularly if the customer agrees with the software development plan and the project contract (call_cu_agree_proj_man_doc_(x)). This applies both for the intermediate versions and the final version of the software development plan and the project contract.

When producing an IRA-document the technical project manager consults his section head, the heads of the production section and the quality assurance adviser. The technical project manager checks with the heads of all the support sections and with the quality assurance adviser to detemine if the needed internal resources are available within the time frame of the project (call_hss_inquire_resource_(x) and call_qaa_inquire_resource_(x)). Internal resources consist of : (software)engineers, computer equipment, support software, person-hours of the computer support section, work locations, office equipment, person-hours of the controller section and person-hours of the quality assurance adviser.

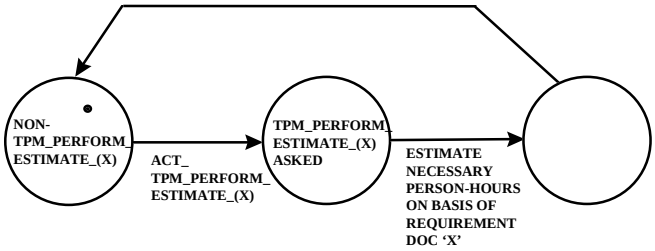


figure 5.89 int-tpm_perform_estimate_(x) : internal behavior STD

The operation 'tpm_perform_estimate_(x)' has one formal parameter 'x'. This is the requirement document on the basis of wich the estimate must be made. The multiplicity of concurrent executing STD instances is zero or more. This is indicated by the solid circle inside the first state.

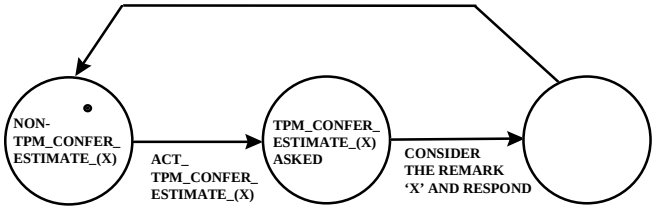


figure 5.90 int-tpm_confer_estimate_(x) : internal behavior STD

The operation 'tpm_confer_estimate_(x)' has one formal parameter 'x'. This models a question or a remark that one party in the dialogue is making. The second party (who executes this STD after it has been called by the first party), considers this question or remark and responds to it. The conference is thus modeled by a 'question-and-answer' session. The first party quizzes the second party repetitively until he is satisfied.

The multiplicity of concurrent executing STD instances is zero or more.

5.3.7.1.4 Technical_Project_Manager : manager-STD

The communication between the technical project manager's operations (callees) and their callers is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

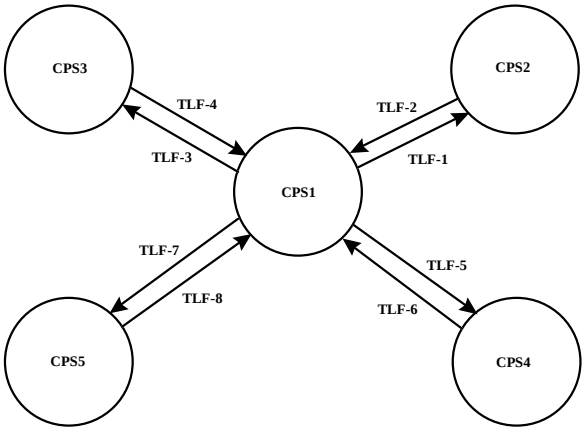


figure 5.91 technical_project_manager : manager STD

The notation CPSx in the STD stands for Consolidated Prescribed Subprocesses and is a set of CCx's. The notation CCx in the STD stands for 'Caller(s)-Callee' and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for 'Traps Logical Formula' and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

The caller-waits construct is modeled conform the second (2^{nd}) variant. This means that the TLF-1, TLF-3, TLF-5 and TLF-7 have to have some additional information for the manager to decide which transition to take. This information comes from the internal bookkeeping of the manager. If an operation is started by the manager on behalf of a caller, a caller-callee relation is initiated. In this way the manager can check whether a non-active employee still has some caller waiting to be allowed to proceed. If both the callee and the caller have terminated, the particular caller-callee relation is cancelled in the internal administration of the manager.

In the state 'neutral' the CPS and the TLFs for the transitions leaving the state are :

CPS1 = {CC1-1, CC2-1, CC3-1}
TLF-1 = T-1 and ((T-3 and not(caller-callee relation)) or (T-5 and not(caller-callee relation)))
TLF-3 = T-11 and (((T-13a and not(caller-callee relation)) or ((T-13b and not(caller-callee relation)) or
((T-13c and not(caller-callee relation)))
TLF-5 = (T-7 and T-9) and not(caller-callee relation)
TLF-7 = (T-1 and ((T-3 and (caller-callee relation)) or (T-5 and (caller-callee relation)))) or
((T-7 and T-9) and (caller-callee relation)) or
(T-11 and (((T-13a and (caller-callee relation)) or ((T-13b and (caller-callee relation)) or
((T-13c and (caller-callee relation)))) or
(T-11 and (T-15a or T-15b)

In the state 'disc_starting_tpm_perform_estimate_(x)' the CPS and the TLFs for the transitions leaving the state are :

CPS2 = {CC1-2, CC2-1, CC3-1}
TLF-2 = T-2

In the state 'starting_tpm_write_proj_man_doc_(x,y)' the CPS and the TLFs for the transitions leaving the state are :

CPS3 = {CC1-1, CC2-1, CC3-2}
TLF-4 = T-12

In the state 'starting_tpm_confer_estimate_(x)' the CPS and the TLFs for the transitions leaving the state are :

CPS4 = {CC1-1, CC2-2, CC3-1}
TLF-6 = T-8

The state 'disc_waiting_caller_proceed' is entered by the manager after it detects that some employee has returned a result. The manager decides in this state which caller is allowed to proceed. The CPS and the TLFs for the transitions leaving the state are :

CPS5 = {CC1-3, CC2-1, CC3-1} or
{CC1-1, CC2-3, CC3-1} or
{CC1-1, CC2-1, CC3-3}
TLF-8 = (T-4 or T-6) or T-10 or (T-14a or T-14b or T14c)

The caller-callee combinations for 'tpm_perform_estimate_(x)' and its two callers 'hprs_second_estimate_(x)' and 'tpm_write_proposal_(x)' are :

CC1-1 = {S1, S3, S5}
CC1-2 = {S2, S3, S5} (either caller has to wait)
CC1-3 = {S1, S4, S5} or (letting the caller 'hprs_second_estimate_(x)' proceed)
{S1, S3, S6} (letting the caller 'tpm_write_proposal_(x)' proceed)

The caller-callee combinations for 'tpm_confer_estimate_(x)' and its caller 'tpm_write_proposal_(x)' are :

CC2-1 = {S7, S9}
CC2-2 = {S8, S9} (caller has to wait)
CC2-3 = {S7, S10} (letting the caller proceed)

The caller-callee combinations for 'tpm_write_proj_man_doc_(x,y)' and its caller 'tpm_write_proposal_(x)' are :

CC3-1 = {S11, S13 or S15}
CC3-2 = {S12, S13}
CC3-3 = {S11, S14}

5.3.7.1.5 Technical_Project_Manager : employee-STDs

The manager STD has 5 employees relevant for this phase. These are the callee 'tpm_perform_estimate_(x)' and its two callers 'hprs_second_estimate_(x)' and 'tpm_write_proposal_(x)', the callee 'tpm_confer_estimate_(x)' and its caller 'tpm_write_proposal_(x)' and the callee 'tpm_write_proj_man_doc_(x,y)' and its caller 'tpm_write_proposal_(x)'.

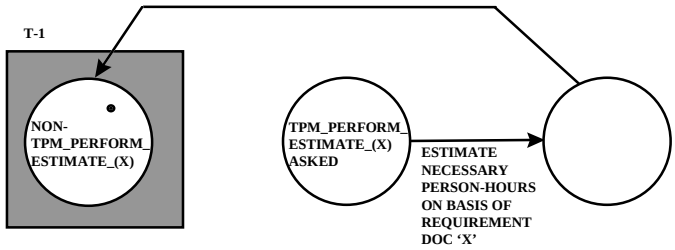


figure 5.92 employee int-tpm_perform_estimate_(x) : subprocess S1

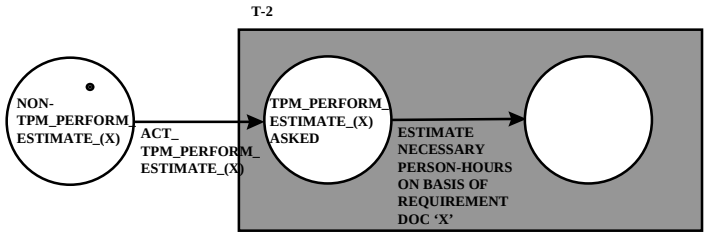


figure 5.93 employee int-tpm_perform_estimate_(x) : subprocess S2

The first employee is the own internal operation 'tpm_perform_estimate_(x)'. This employee has two subprocesses S1 and S1 and two traps T-1 and T-2 according to the caller_callee-construct. The caller has to wait for the result. This is modeled by the second variant of the caller-waits construct. That is to say, the manager detects that the employee 'tpm_perform_estimate_(x)' has entered trap T1 after its execution has finished. The manager will then transit to its state 'disc_waiting_ caller_proceed' and it will allow the caller to proceed. To be able to do this, the manager has an internal administration in which the caller-callee relations are recorded. A caller-callee relation connects a caller (i.e. the specific call of an instance of an operation of an object) with a callee (i.e. specific instance of the called operation) that has been

started by the manager on behalf of that caller. This way the manager can check if the operation ‘tpm_perform_estimate_(x)’ when it is in trap T1, still has a waiting caller associated with it.

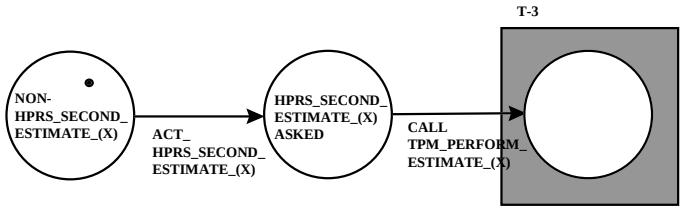


figure 5.94 employee int-hprs_second_estimate_(x) : subprocess S3

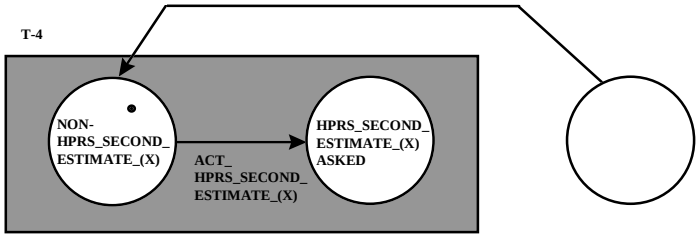


figure 5.95 employee int-hprs_second_estimate_(x) : subprocess S4

The second employee is the caller operation ‘hprs_second_estimate_(x)’. This employee has two subprocesses S3 and S4 and two traps T-3 and T-4 according to the caller_callee-construct. This caller of the operation ‘tpm_perform_ estimate_(x) has to wait for the the result produced by the callee. This is taken care of conform the second variant of the caller-callee construct.

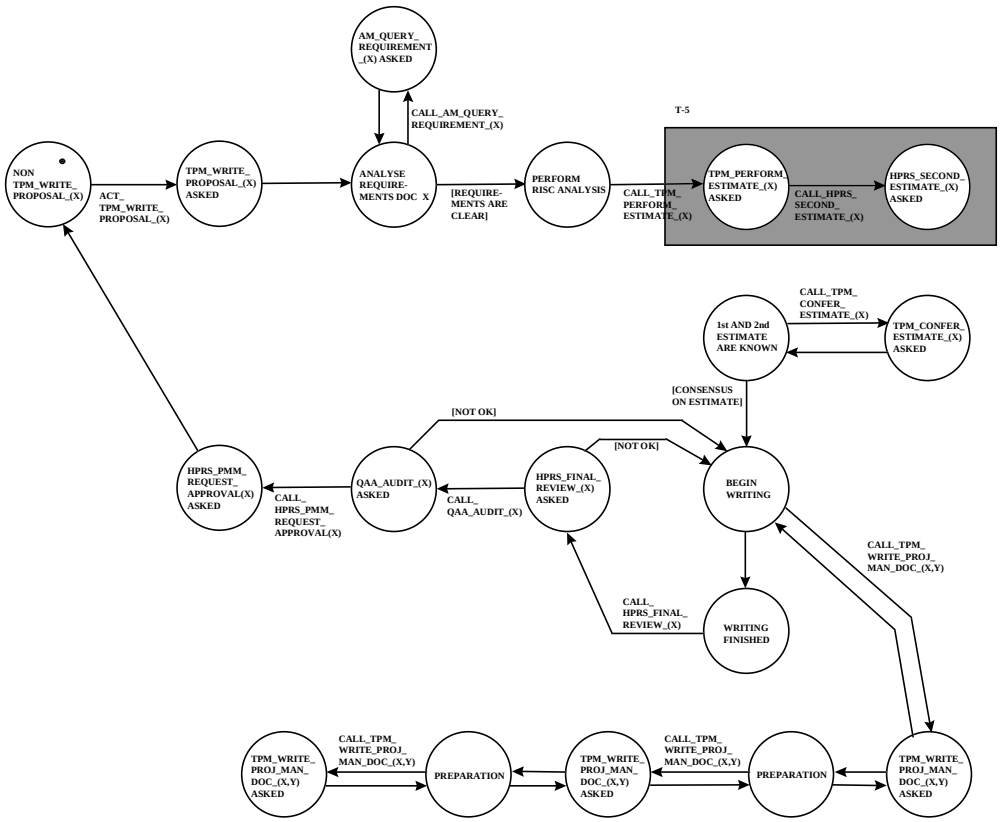


figure 5.96 employee int-tpm_write_proposal_(x) : subprocess S5

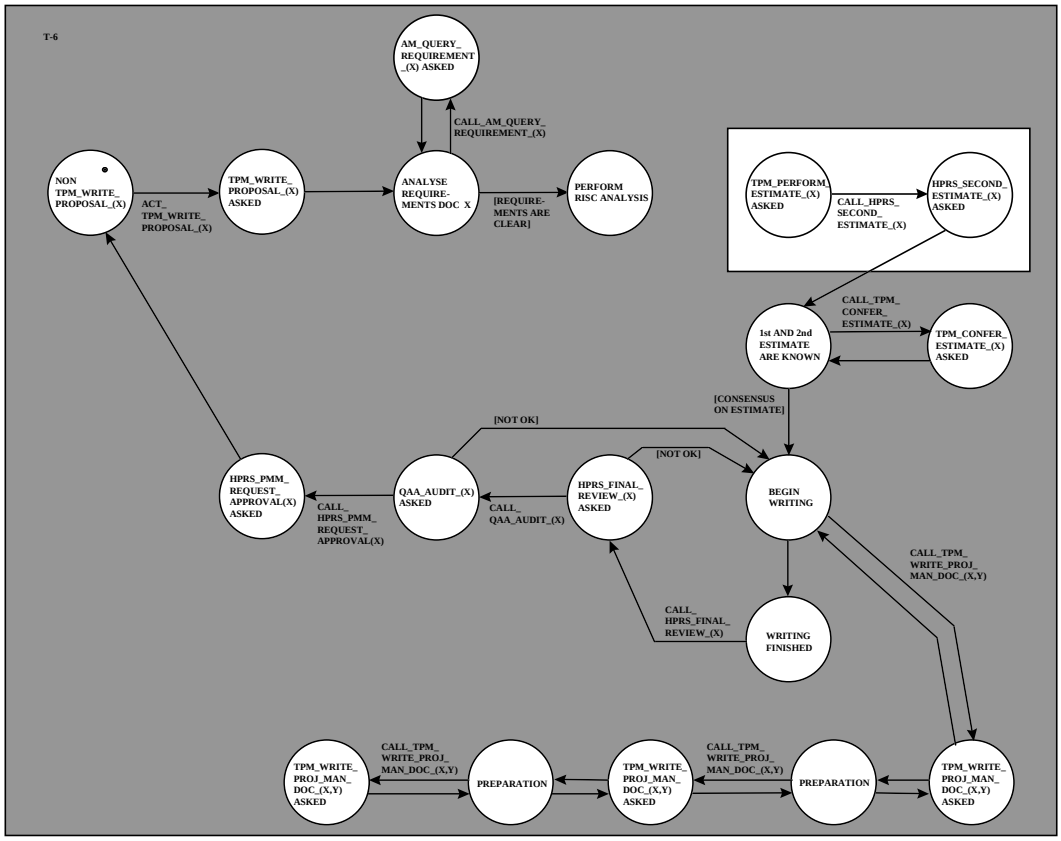


figure 5.97 employee int-tpm_write_proposal_(x) : subprocess S6

The third employee is the caller operation ‘tpm_write_proposal_(x)’. This operation calls ‘tpm_perform_estimate_(x)’ of the same object. After the call it may proceed another state (it calls ‘hprs_second_estimate_(x)’), but then it has to wait for the result of ‘tpm_perform_estimate(x)’. (At this point it also has to wait for the estimate returned by ‘hprs_ second_estimate_(x)’. So it may only proceed if both the first and second estimate are known.) The fact that it may proceed right after the call, but must wait for the result further on in the STD results in two subprocesses : S5 with the two state-trap T-5 allowing the STD to proceed one state after the call and S6 with the trap T-6.

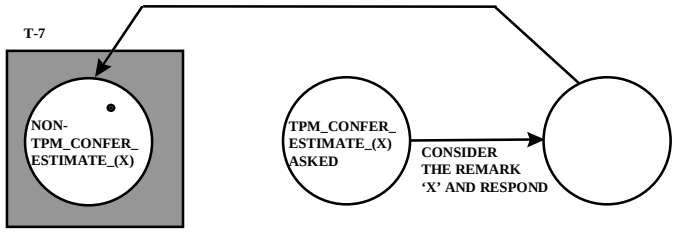


figure 5.98 employee int-tpm_confer_estimate_(x) : subprocess S7

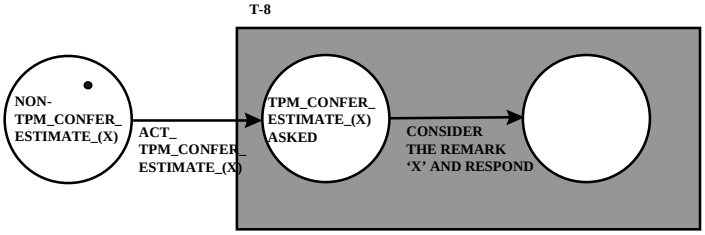


figure 5.99 employee int-tpm_confer_estimate_(x) : subprocess S8

The fourth employee is the own internal operation ‘tpm_confer_estimate_(x)’. This employee has two subprocesses S7 and S8 and two traps T-7 and T-8 according to the caller_callee-construct. The caller, which is the operation ‘tpm_

write_proposal_(x)' of another tpm-object, has to wait for the result. This is modeled conform the second variant of the caller-waits construct. The subprocesses and traps of the caller in this respect are S9 and S10 and T-9 and T-10 according to the caller-callee construct.

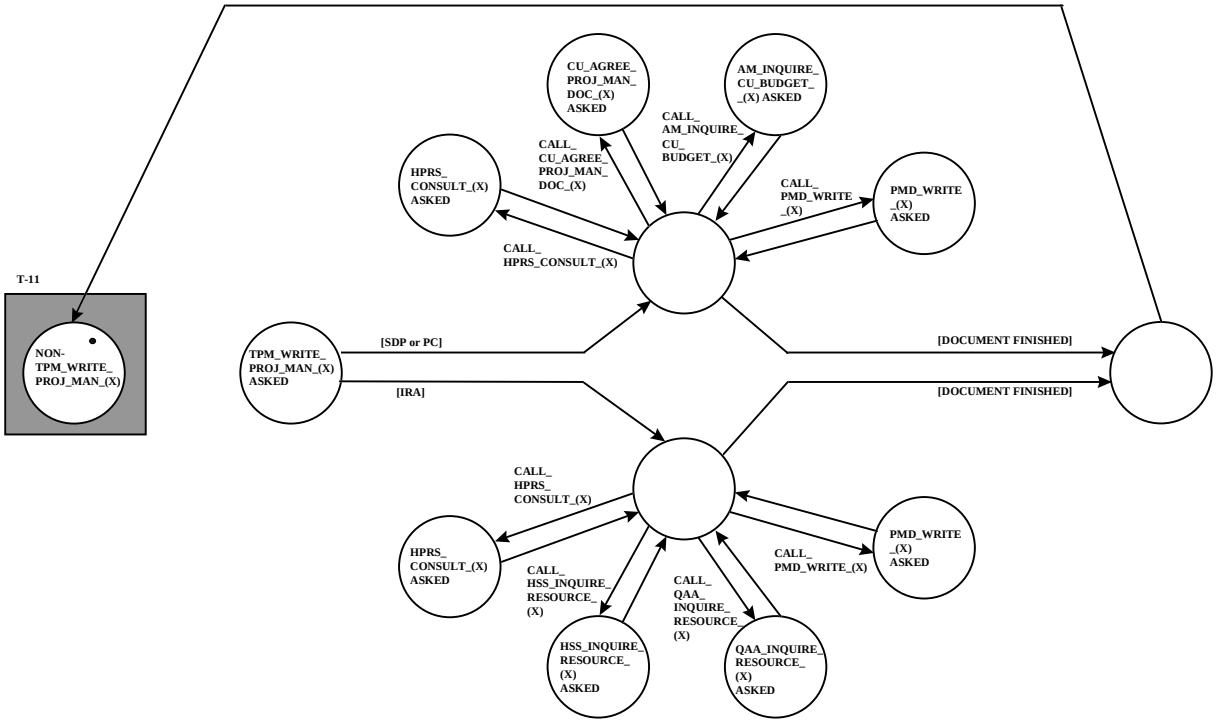


figure 5.100 employee int-tpm_write_proj_man_doc_(x,y) : subprocess S11

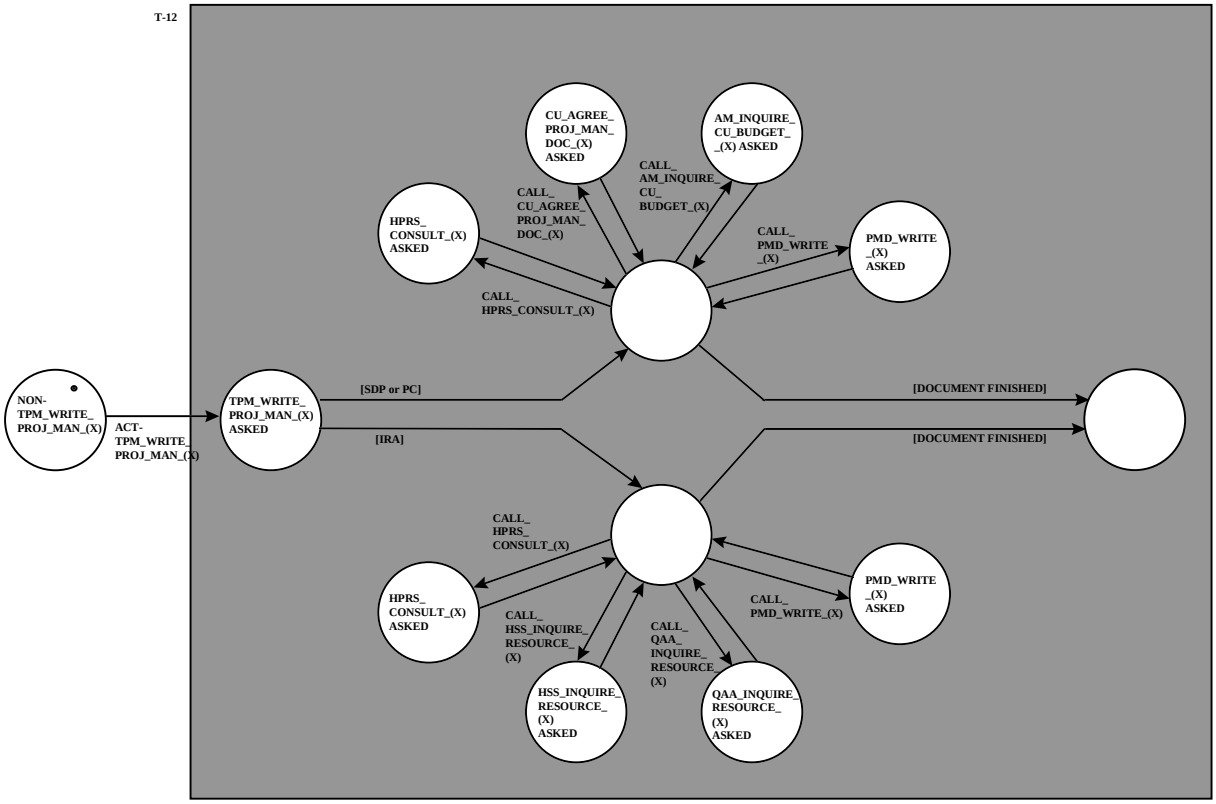


figure 5.101 employee int-tpm_write_proj_man_doc_(x,y) : subprocess S12

The fifth employee is the own internal operation ‘tpm_write_proj_man_doc_(x)’. This employee has two subprocesses S11 and S12 and two traps T-11 and T-12 according to the caller_callee-construct. The caller, which is the operation ‘tpm_write_proposal_(x)’ of the same tpm-object as to which the callee belongs, has to wait for a result to be returned. There can be up to three instances of ‘tpm_write_proj_man_doc_(x,y)’ running concurrently (each with a different value of y). Each instance is started by the manager on behalf of the same caller. Consequently the manager STD plays three roles and prescribes three subprocesses at any one time for the caller. These prescribed subprocesses can (but don’t have to) be different. The actual subprocess of the caller is the intersection of these three subprocesses. In this model the same subprocess is prescribed by the three manager-roles at any one time. So the intersection is equal to any one of these subprocesses.

The caller has to wait until all concurrent instances of the callee have turned in their result. This is achieved by breaking up the operation ‘tpm_write_proposal_(x)’ into three subprocesses and making use of the manager state ‘disc_waiting_ caller_proceed’. The subprocesses are S13 with the nested traps T-13a, T-13b and T-13c, S14 with the traps T-14a, T-14b and T-14c, and S15 with the traps T-15a and T-15b.

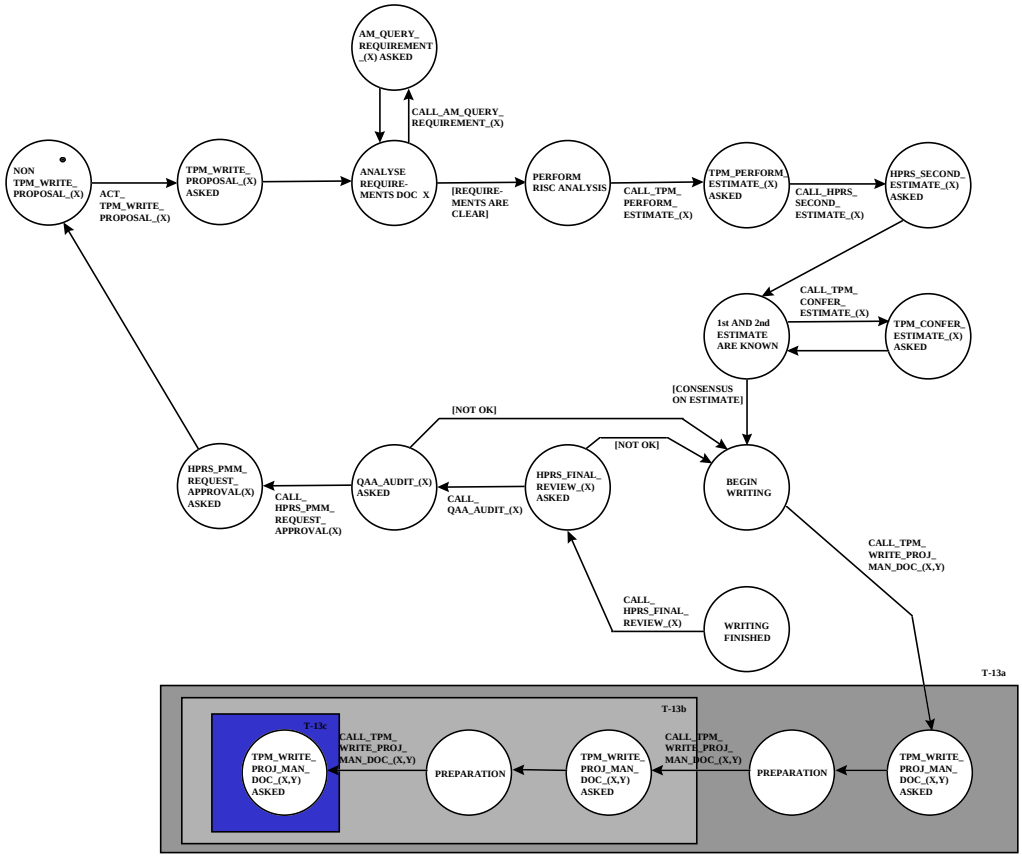


figure 5.102 employee int-tpm_write_proposal_(x) : subprocess S13

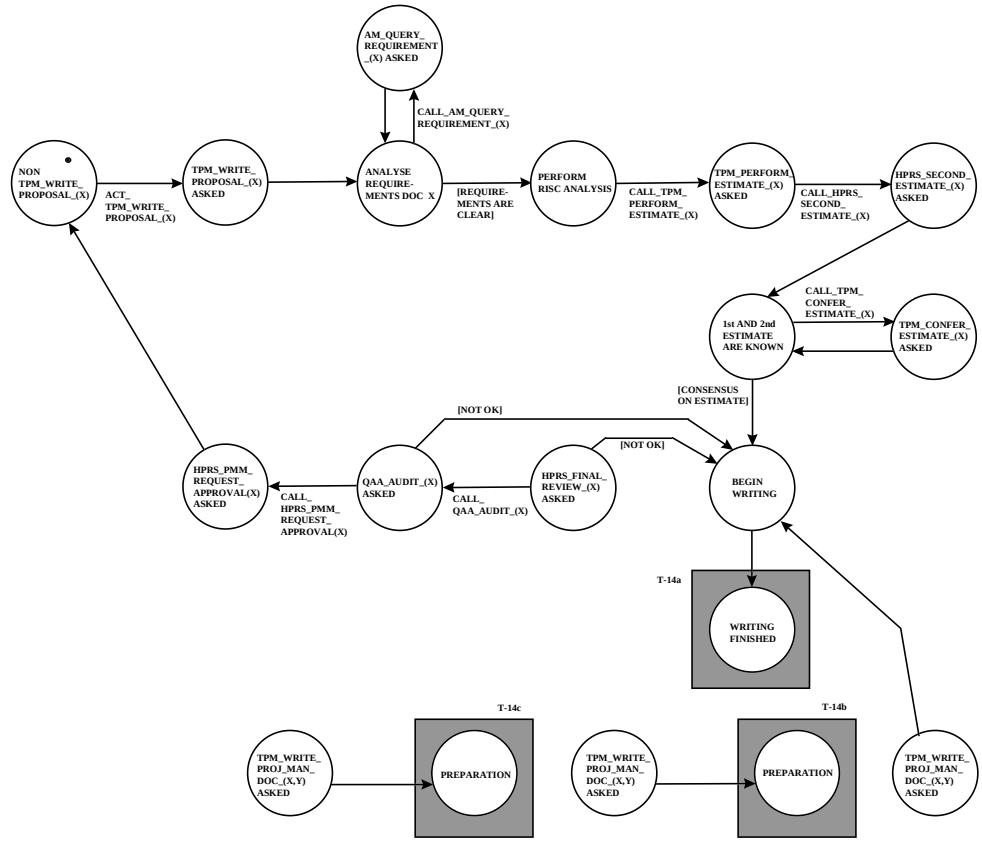


figure 5.103 employee int-tpm_write_proposal_(x) : subprocess S14

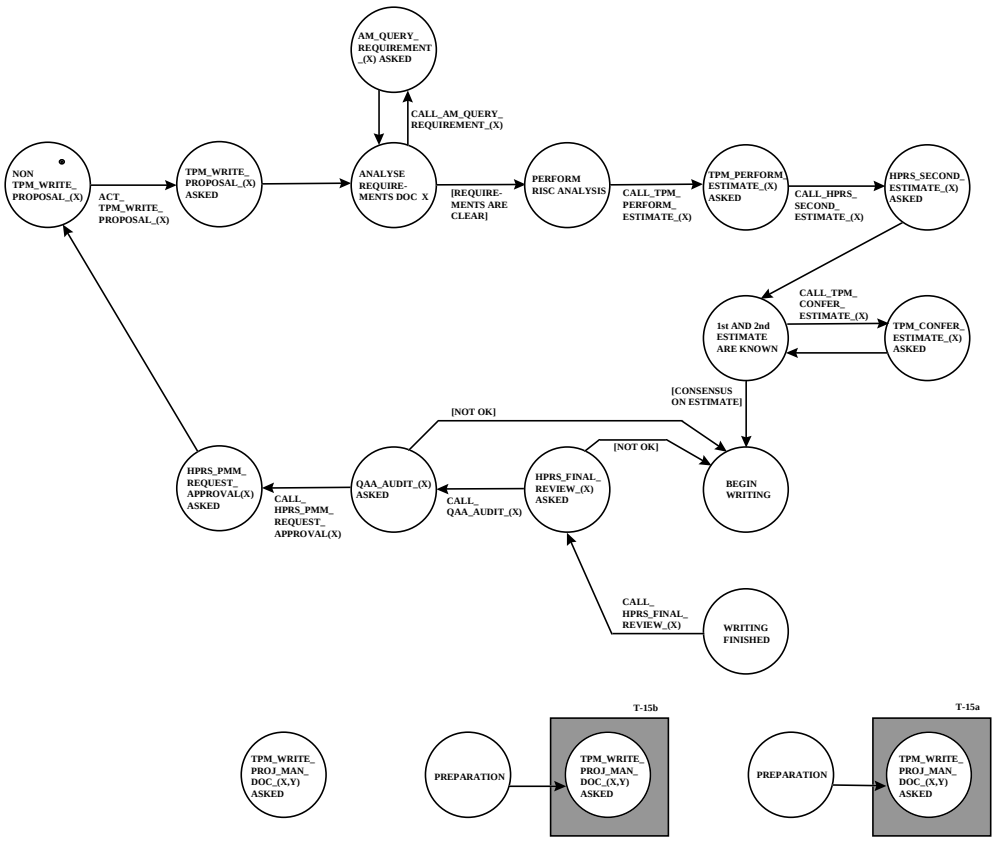


figure 5.104 employee int-tpm_write_proposal_(x) : subprocess S15

The assumption in the model is that the tpm calls the needed operations ‘tpm_write_proj_man_doc_(x,y)’ one after the other without delay. Initialy the tpm executes 3 calls when he starts from scratch. He has to produce the three project management documents ‘software development plan’, ‘internal resoures allocation-document’ and ‘project contract’. Later on, when he has to correct some (or all) of the documents, he executes 1,2 or 3 calls. It is further assumed that the point in time on which the tpm performs its last call precedes the point in time on which any one of the earlier called ‘tpm_write_proj_man_doc_(x,y)’ operations finishes its execution. After the last call the tpm stays in the corresponding state ‘tpm_write_proj_man_doc_(x,y) asked’ and will not proceed to the next ‘preparation’ state. The tpm can in this model no longer place a call to ‘tpm_write_proj_man_doc_(x,y)’ after one of the previously started operations has returned a result. These are reasonable assumptions considering the amount of time that is needed to write a projectmanagment document and the fact that all the needed calls are made within a short time frame.

At first the manager prescribes in its neutral state the subprocess S11 for the callee and S13 for the caller. The manager detects that the first call is placed when the caller enters it trap T13-a. The manager reacts (if the callee is in T-11) by going to it state ‘starting tpm_write_proj_man_doc_(x,y)’ and starts the first instance of the called operation. In this state ‘starting tpm_write_proj_man_doc_(x,y)’ the manager prescribes S12 for the callee instance and still S13 for the caller. If the callee enters trap T-12, the manager can (and will) transit back to neutral. The manager detects the second call (if there is any) when the caller enters the trap T-13b. The manager will then transit again to its state ‘starting tpm_write_ proj_man_doc_(x,y)’ and start a second instance of the callee (this second instance is per definition in its trap T-11). The manager precsribes in this state S12 for the second instance and still S11 for the first instance. For the caller it prescribes S13. (That is to say both manager roles prescribe S13 for the caller and the actual prescribed subprocess is thus S13). If the second callee instance enters its trap T-12, the manager transits back to neutral. The third call (is there is any) is handled by the manager in a similar way.

Now all the calls that the tpm is going to make, are made. The manager is in its neutral state. Then one of the ‘tpm_write_ proj_man_doc_(x,y)’ instances finishes its execution. The manager detects this when an instance, for which there exists a caller-callee relation, enters its trap T-11. The manager transits to its state ‘disc_waiting_caller_proceed’. Here it still prescribes S11 for the callee instance. For the caller the manager (in all it roles) prescribes the subprocess S14. The manager waits in the state ‘disc_waiting_caller_proceed’ until the caller has entered trap T-14a or T-14b or T14-c. I.e. the caller is allowed to ‘shift’ one state back.

Suppose the caller enters in its traps T-14c or T-14b (i.e. it has placed 2 or 3 calls). If it does so, the manager transits back to its neutral state where it now prescribes S15 for the caller. The manager can now handle the next finishing instance of ‘tpm_write_proj_man_doc_(x,y)’. When this next instance finishes, i.e has entered its T-11, and the caller is in its trap T-15a or T-15b (i.e. has shifted back another state), the manager transits again to the its state ‘disc_ waiting_caller_proceed’ where it prescribes again S14 for the caller.

Suppose the caller enters now its trap T-14a (i.e. there are no more instances executing on behalf of the caller). When the manager detects this, it transits back to its state ‘neutral’. Here it prescribes now S13 for the caller. This means that the caller is now allowed to proceed.

5.3.7.2 Customer

5.3.7.2.1 Customer : external behavior-STD, organizational view

The external behavior STD shows the possible sequences in which the exported operation(s) can be started. The class ‘customer’ has two operations relevant for the process fragment ‘writing project management documents’, phase2. The organizational view of the external STD does not show any communication details. Consequently the STD consists of one state ‘neutral’ in which the customer waits for a call to its exported operations ‘cu_query_requirement_(x)’ and ‘cu_agree_proj_man_doc_(x). If a call has taken place, the customer can make the transition labeled wih the called operation. The customer then comes again in the state ‘neutral’. The possible starting sequence specified by this STD is thus any combination of the operations ‘cu_query_requirement_(x)’ and ‘cu_agree_proj_man_doc_(x)’.

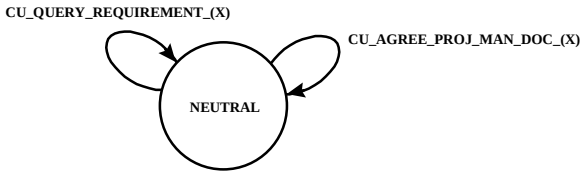


figure 5.105 customer : external behavior STD, organizational view

5.3.7.2.2 Customer : external behavior-STD, communicative view

The communicative view of the external behavior STD shows the communication details between the external STD and the called operation(s). It consists of a neutral state in which the customer waits for a call to its operation and two starting states in which the called operations are started. Typically the customer does not wait in this ‘starting’ states until the called operation is finished, but returns as soon as possible to its neutral state to allow handling a call of its other operations (or a call to the just started operation).

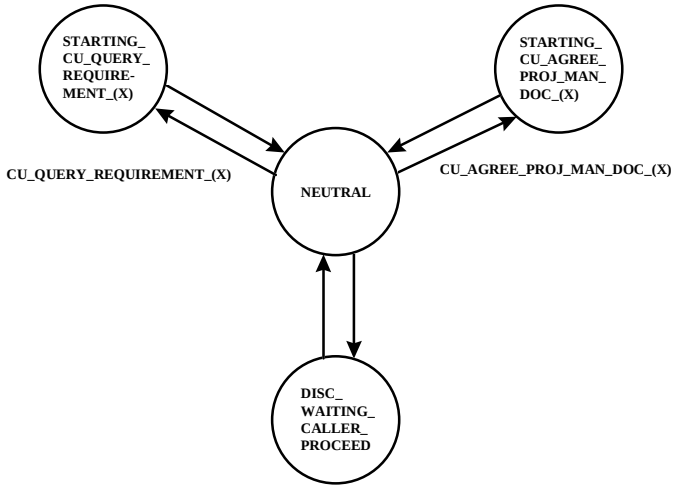


figure 5.106 customer : external behavior STD, communicative view

When ‘cu_query_requirement_(x)’ or ‘cu_agree_proj_man_doc_(x)’ are called, the callers must wait for the answer that is produced by the callees. When a callee has produced an answer, the manager transits to the state ‘disc_waiting_ caller_proceed’. This state‘disc_waiting_caller_proceed’ is an aggregate state in which the manager determines which waiting caller had called the callee that produced the result. This caller is then allowed to proceed.

5.3.7.2.3 Customer : internal behavior-STDs

The customer class has 2 internal operations, ‘cu_query_requirement_(x)’ and ‘cu_agree_proj_man_doc_(x)’, that are relevant for phase 2. These operations have the following internal behavior STDs.

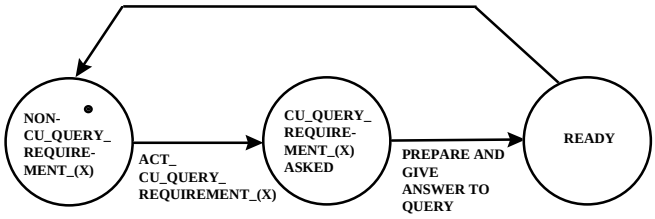


figure 5.107 int-cu_query_requirement_(x) : internal behavior STD

The operation ‘cu_query_requirement_(x)’ is called to ask the customer a question about a certain requirements document. The formal parameter ‘x’ specifies this requirements document. With this operation the customer answers the question.

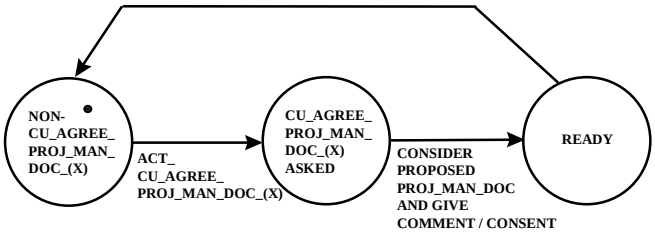


figure 5.108 int-cu_agree_proj_man_doc_(x) : internal behavior STD

The operation ‘cu_agree_proj_man_doc_(x)’ is called to ask the customer if he agrees with a certain (maybe final) version of a project management document. The formal parameter ‘x’ is this project management document in question. This can either be a software development plan or a project contract. With this operation the customer either gives his consent to the document or he does not agree and gives some comments. This is the operation that is called by the technical project manager when he writes a project management document. If the customer does not agree and gives some comments, the technical project manager updates the document accordingly. If the customer agrees with the document, then the technical project manager knows that the document in its present form will be eventually signed by the customer and it can therefore be given to his section head for final review.

5.3.7.2.4 Customer : manager-STD

The communication between the customer’s operations (callees) and its callers is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

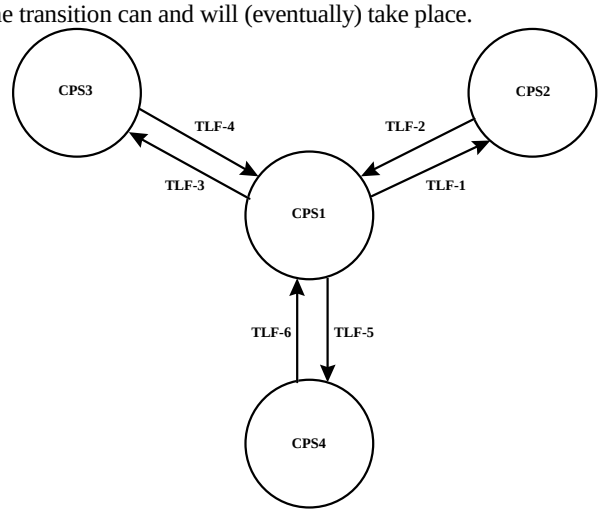


figure 5.109 customer : manager STD

The notation CPSx in the STD stands for Consolidated Prescribed Subprocesses and is a set of CCx's. The notation CCx in the STD stands for 'Caller(s)-Callee' and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for 'Traps Logical Formula' and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

The caller-waits construct is modeled conform the second variant. This means that the TLF-1, TLF-3,and TLF-5 have to have some additional information for the manager to decide which transition to take. This information comes from the internal bookkeeping of the manager. If an operation is started by the manager on behalf of a caller, a caller-callee relation is initiated. In this way the manager can check whether a non-active employee still has some caller waiting to be allowed to proceed. If both the callee and the caller have terminated, the particular caller-callee relation is cancelled in the internal administration of the manager.

In the state 'neutral' the CPS and the TLFs for the transitions leaving the state are :

CPS1 = {CC1-1, CC2-1}
TLF-1 = T-1 and (T-3 and not(caller-callee relation))
TLF-3 = T-5 and (T-7 and not(caller-callee relation))
TLF-5 = (T-1 and (T-3 and (caller-callee relation)) or
(T-5 and (T7 and (caller-callee relation))

In the state 'starting_cu_query_requirement_(x)' the CPS and the TLFs for the transitions leaving the state are :

CPS2 = {CC1-2, CC2-1}
TLF-2 = T-2

In the state 'starting_cu_agree_proj_man_doc_(x)' the CPS and the TLFs for the transitions leaving the state are :

CPS3 = {CC1-1, CC2-2}
TLF-4 = T-6

The state 'disc_waiting_caller_proceed' is entered by the manager after it detects that some employee has returned a result. The manager decides in this state which caller is allowed to proceed. The CPS and the TLFs for the transitions leaving the state are :

CPS4 = {CC1-3, CC2-1} or
{CC1-1, CC2-3}
TLF-6 = T-4 or T-8

The caller-callee combinations for 'cu_query_requirement_(x)' and its caller 'am_query_requirement_(x)' are :

CC1-1 = {S1, S3}
CC1-2 = {S2, S3} (the caller has to wait)
CC1-3 = {S1, S4} (the caller may proceed)

The caller-callee combinations for 'cu_agree_proj_man_doc_(x)' and its caller 'tpm_write_proj_man_doc_(x,y)' are :

CC2-1 = {S5, S7}
CC2-2 = {S6, S7} (the caller has to wait)
CC2-3 = {S5, S8} (the caller may proceed)

5.3.7.2.5 Customer : employee-STDs

The manager STD has 4 employees relevant for this phase. The first employee is the own internal operation 'cu_query_requirement_(x)'. This employee has two subprocesses S1 and S1 and two traps T-1 and T-2 according to the caller_callee-construct.

The second employee is the operation ‘am_query_requirement_(x)’ of the class account manager. This is the caller of ‘cu_query_requirement_(x)’. This second employee has the subprocesses S3 and S4 and traps T-3 and T-4 according to the caller-callee construct. The caller has to wait for the result. This is modeled conform the second variant of the caller-waits construct.

The third employee is the own internal operation ‘cu_agree_proj_man_doc_(x)’. This employee has two subprocesses S5 and S6 and two traps T-5 and T-6 according to the caller_callee-construct.

The fourth employee is the operation ‘tpm_write_proj_man_doc_(x,y)’ of the class technical project manager. This is the caller of ‘cu_agree_proj_man_doc_(x)’. This fourth employee has the subprocesses S7 and S8 and traps T-7 and T-8 according to the caller-callee construct. The caller has to wait for the result. This is modeled conform the second variant of the caller-waits construct.

5.3.7.3 Account manager

5.3.7.3.1 Account manager : external behavior-STD, organizational view

The external behavior STD shows the possible sequences in which the exported operation(s) can be started. The class ‘account manager’ has two operations relevant for the process fragment ‘writing project management documents’, phase2. The organizational view of the external STD does not show any communication details. Consequently the STD consists of one state ‘neutral’ in which the account manager waits for a call to its exported operations ‘am_query_requirement_(x)’ and ‘am_inquire_cu_budget_(x). If a call has taken place, the account manager can make the transition labeled wih the called operation. The account manager then comes again in the state ‘neutral’. The possible starting sequence specified by this STD is thus any combination of the operations ‘am_query_requirement_(x)’ and ‘am_inquire_cu_budget_(x)’.

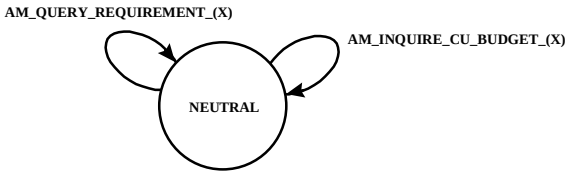


figure 5.110 account manager : external behavior STD, organizational view

5.3.7.3.2 Account manager : external behavior-STD, communicative view

The communicative view of the external behavior STD shows the communication details between the external STD and the called operation(s). It consists of a neutral state in which the customer waits for a call to its operation and two starting states in which the called operations are started. Typically the customer does not wait in this ‘starting’ states until the called operation is finished, but returns as soon as possible to its neutral state to allow handling another call to one of its operations.

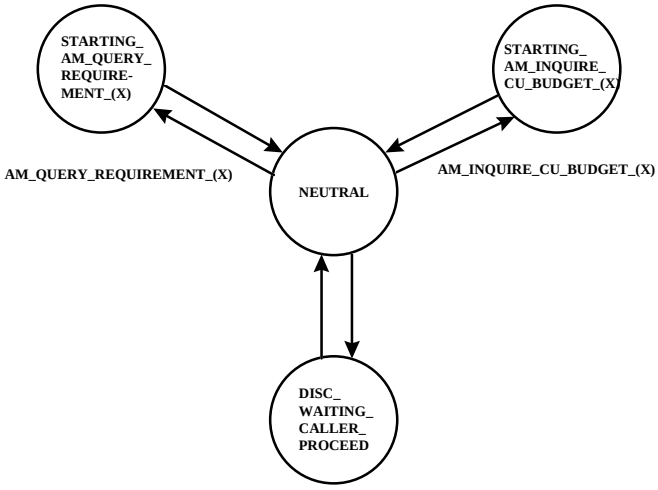


figure 5.111 account manager : external behavior STD, communicative view

When ‘am_query_requirement_(x)’ or ‘am_inquire_cu_budget_(x)’ are called, the callers must wait for the answer that is produced by the callees. When a callee has produced an answer, the manager transits to the state ‘disc_waiting_ caller_proceed’. This state‘disc_waiting_caller_proceed’ is an aggregate state in which the manager determines which waiting caller had called the callee that produced the result. This caller is then allowed to proceed.

5.3.7.3.3 Account manager : internal behavior-STDs

The account manager class has 2 internal operations, ‘am_query_requirement_(x)’ and ‘am_inquire_cu_budget_(x)’, that are relevant for phase 2. These operations have the following internal behavior STDs.

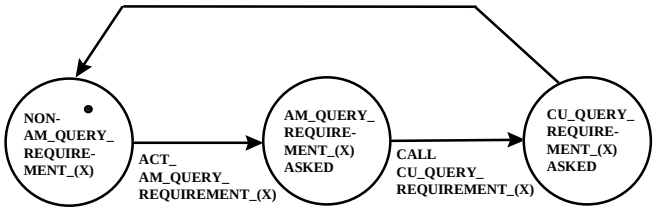


figure 5.112 int-am_query_requirement_(x) : internal behavior STD

The operation ‘am_query_requirement_(x)’ is called to ask the account manager a question about a certain requirements document. The formal parameter ‘x’ specifies this requirements document. The account manager relays the question to the customer (call cu_query_requirement_(x)). When the account manager has received an answer from the customer, this answer is relayed back to the original inquiring object.

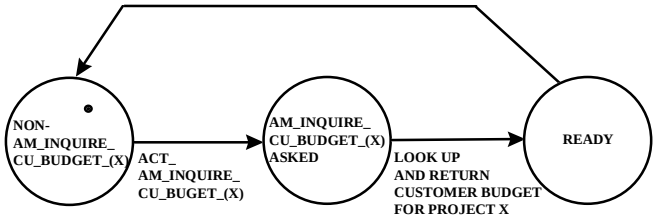


figure 5.113 int-am_inquire_cu_budget_(x) : internal behavior STD

The operation ‘am_inquire_cu_budget_(x)’ is called to ask the account manager if a customer has a sufficient budget for a certain project. The formal parameter ‘x’ is a compound datastructure which holds the customer and project in question. The account manager answers this question after consulting the relevant customer data.

5.3.7.3.4 Account manager : manager-STD

The communication between the account manager’s operations (callees) and its callers is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

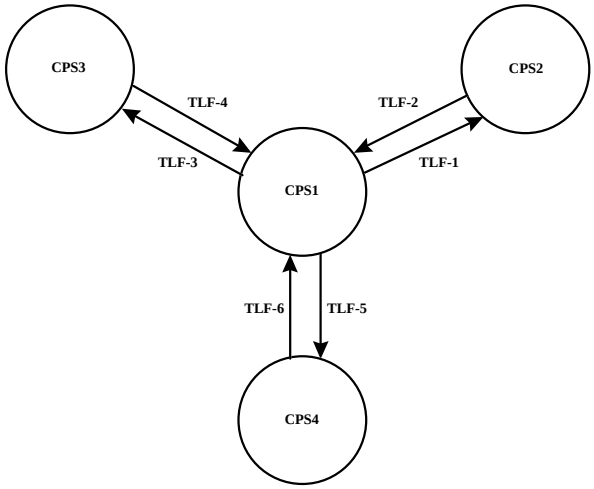


figure 5.114 account manager : manager STD

The notation CPSx in the STD stands for Consolidated Prescribed Subprocesses and is a set of CCx's. The notation CCx in the STD stands for 'Caller(s)-Callee' and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for 'Traps Logical Formula' and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

The caller-waits construct is modeled conform the second variant. This means that the TLF-1, TLF-3 and TLF-5 have to have some additional information for the manager to decide which transition to take. This information comes from the internal bookkeeping of the manager. If an operation is started by the manager on behalf of a caller, a caller-callee relation is initiated. In this way the manager can check whether a non-active employee still has some caller waiting to be allowed to proceed. If both the callee and the caller have terminated, the particular caller-callee relation is cancelled in the internal administration of the manager.

In the state 'neutral' the CPS and the TLFs for the transitions leaving the state are :

CPS1 = {CC1-1, CC2-1}
TLF-1 = T-1 and (T-3 and not(caller-callee relation))
TLF-3 = T-5 and (T-7 and not(caller-callee relation))
TLF-5 = (T-1 and (T-3 and (caller-callee relation)) or
(T-5 and (T-7 and (caller-callee relation))

In the state 'starting_am_query_requirement_(x)' the CPS and the TLFs for the transitions leaving the state are :

CPS2 = {CC1-2, CC2-1}
TLF-2 = T-2

In the state 'starting_am_inquire_cu_budget_(x)' the CPS and the TLFs for the transitions leaving the state are :

CPS3 = {CC1-1, CC2-2}
TLF-4 = T-6

The state 'disc_waiting_caller_proceed' is entered by the manager after it detects that some employee has returned a result. The manager decides in this state which caller is allowed to proceed. The CPS and the TLFs for the transitions leaving the state are :

CPS4 = {CC1-3, CC2-1} or
{CC1-1, CC2-3}
TLF-6 = T-4 or T-8

The caller-callee combinations for 'am_query_requirement_(x)' and its caller 'tpm_write_proposal_(x)' are :

CC1-1 = {S1, S3}
CC1-2 = {S2, S3} (the caller has to wait)
CC1-3 = {S1, S4} (the caller may proceed)

The caller-callee combinations for 'am_inquire_cu_budget_(x)' and its caller 'tpm_write_proj_man_doc_(x,y)' are :

CC2-1 = {S5, S7}
CC2-2 = {S6, S7} (the caller has to wait)
CC2-3 = {S5, S8} (the caller may proceed)

5.3.7.3.5 Account manager : employee-STDs

The manager STD has 4 employees relevant for this phase. The first employee is the own internal operation 'am_query_requirement_(x)'. This employee has two subprocesses S1 and S1 and two traps T-1 and T-2 according to the caller_callee-construct.

The second employee is the operation 'tpm_write_proposal_(x)' of the class technical project manager. This is the caller of 'am_query_requirement_(x)'. This second employee has the subprocesses S3 and S4 and traps T-3 and T-4 according to the caller-callee construct. The caller has to wait for the result. This is modeled by the second variant of the caller-waits construct.

The third employee is the own internal operation ‘am_inquire_cu_budget_(x)’. This employee has two subprocesses S5 and S6 and two traps T-5 and T-6 according to the caller_callee-construct.

The fourth employee is the operation ‘tpm_write_proj_man_doc_(x,y)’ of the class technical project manager. This is the caller of ‘am_inquire_cu_budget_(x)’. This fourth employee has the subprocesses S7 and S8 and traps T-7 and T-8 according to the caller-callee construct. The caller has to wait for the result. This is modeled conform the second variant of the caller-waits construct.

5.3.7.4 Quality assurance adviser

5.3.7.4.1 Quality assurance adviser : external behavior-STD, organizational view

The external behavior STD shows the possible sequences in which the exported operation(s) can be started. The class ‘quality assurance adviser’ has two operations relevant for the process fragment ‘writing project management documents’, phase2. The organizational view of the external STD does not show any communication details. Consequently the STD consists of one state ‘neutral’ in which the account manager waits for a call to its exported operations ‘qaa_inquire_resource_(x)’ and ‘qaa_audit_(x). If a call has taken place, the account manager can make the transition labeled wih the called operation. The account manager then comes again in the state ‘neutral’. The possible starting sequence specified by this STD is thus any combination of the operations ‘qaa_inquire_resource_(x)’ and ‘qaa_audit_(x)’.

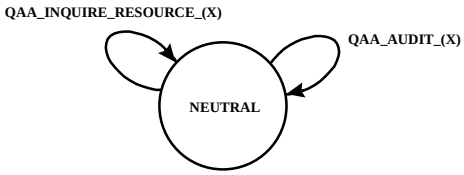


figure 5.115 quality asssurance adviser : external behavior STD, organizational view

5.3.7.4.2 Quality assurance adviser : external behavior-STD, communicative view

The communicative view of the external behavior STD shows the communication details between the external STD and the called operation(s). It consists of a neutral state in which the customer waits for a call to its operation and two starting states in which the called operations are started. Typically the customer does not wait in this ‘starting’ states until the called operation is finished, but returns as soon as possible to its neutral state to allow handling another call to one of its operations.

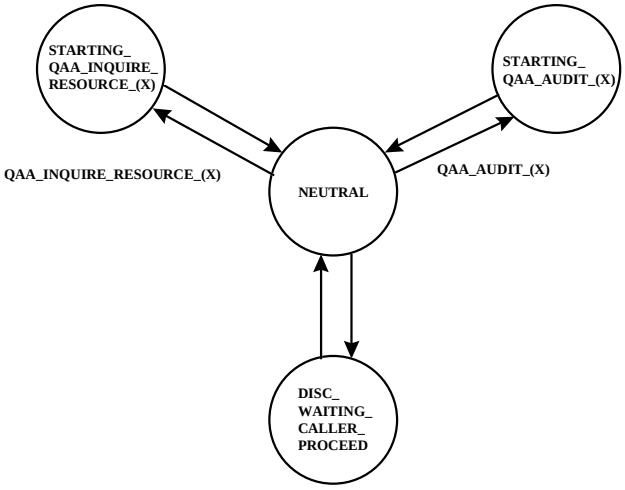


figure 5.116 quality assurance adviser : external behavior STD, communicative view

When ‘qaa_inquire_resource_(x)’ or ‘qaa_audit_(x)’ are called, the callers must wait for the answer that is produced by the callees. When a callee has produced an answer, the manager transits to the state ‘disc_waiting_ caller_proceed’. This state‘disc_waiting_caller_proceed’ is an aggregate state in which the manager determines which waiting caller had called the callee that produced the result. This caller is then allowed to proceed.

5.3.7.4.3 Quality assurance adviser : internal behavior-STDs

The quality assurance adviser class has 2 internal operations, ‘qaa_inquire_resource_(x)’ and ‘qaa_audit_(x)’, that are relevant for phase 2. These operations have the following internal behavior STDs.

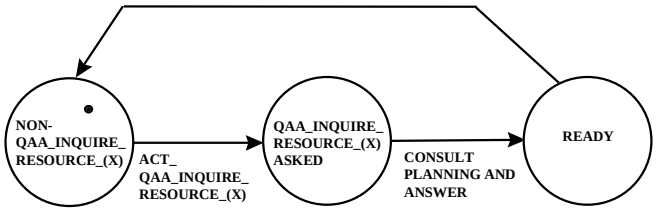


figure 5.117 int-qaa_inquire_resource_(x) : internal behavior STD

The operation ‘qaa_inquire_resource_(x)’ is called to ask the quality assurance adviser (qaa) if he can spend a certain amount of time (in a certain time frame) on a certain project. The formal parameter ‘x’ specifies the time frame during which the assistance of the quality assurance adviser is necessary. The quality assurance adviser inspects his planning (agenda) and indicates whether or not he is available for advise. The caller of this operation is the technical project manager (tpm) when he writes an ‘internal resources allocation’-document. If the qaa can accommodate the tpm, then the tpm will record it in the document. If the qaa can not accommodate the tpm, then the tpm has to determine a new time frame for the qaa. He will ask the qaa again if he is available in the new time frame. Etc. until a match has been found. If the schedule of the project changes significantly because of non-availability of the qaa, the tpm will inform his section head of the problem.

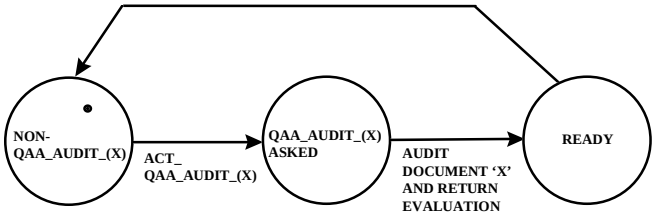


figure 5.118 int-qaa_audit_(x) : internal behavior STD

With the operation ‘qaa_audit_(x)’ the quality assurance adviser (qaa) audits a document. The document to be audited is passed as the parameter ‘x’. The operation returns the audit result to the caller. The caller of this operation is the technical project manager (tpm) after he has written a project management document. If the audit result is not OK, the tpm will update the document according to the comments of the qaa (see also operation ‘tpm_write_proj_man_doc_(x)’ of the tpm).

5.3.7.4.4 Quality assurance adviser : manager-STD

The communication between the quality assurance adviser’s operations (callees) and its callers is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

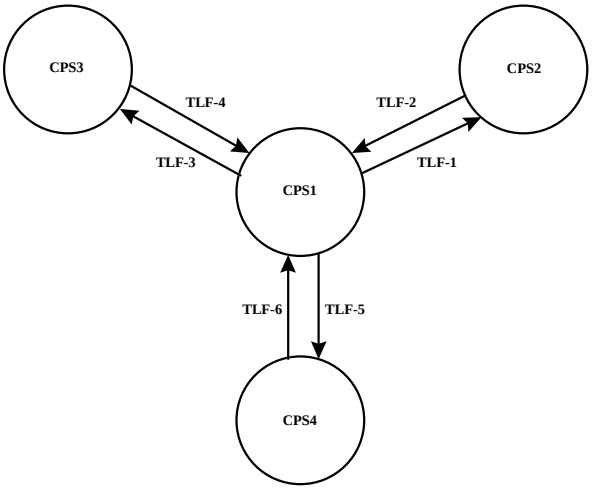


figure 5.119 quality assurance adviser : manager STD

The manager STD and the subprocesses and traps of its employees are conceptualy the same, and use the same names, as those used in the modeling of the account manager (in phase 2). For the definition of the CPSs, TLFs and CCs a reference is therefore made to those of the account manager (in phase 2). The account manager's employees must then be substituted by the quality assurance adviser's employees.

5.3.7.4.5 Quality assurance adviser : employee-STDs

The manager STD has 4 employees relevant for this phase. The first employee is the own internal operation 'qaa_inquire_resource_(x)'. This employee has two subprocesses S1 and S1 and two traps T-1 and T-2 according to the caller_callee-construct.

The second employee is the operation 'tpm_write_proj_man_doc_(x,y)' of the class technical project manager. This is the caller of 'qaa_inquire_resource_(x)'. This second employee has the subprocesses S3 and S4 and traps T-3 and T-4 according to the caller-callee construct. The caller has to wait for the result. This is modeled conform the second variant of the caller-waits construct.

The third employee is the own internal operation 'qaa_audit_(x)'. This employee has two subprocesses S5 and S6 and two traps T-5 and T-6 according to the caller-callee-construct.

The fourth employee is the operation 'tpm_write_proposal_(x)' of the class technical project manager. This is the caller of 'qaa_audit_(x)'. This fourth employee has the subprocesses S7 and S8 and traps T-7 and T-8 according to the caller-callee construct. The caller has to wait for the result. This is modeled conform the second variant of the caller-waits construct.

5.3.7.5 Head production section

5.3.7.5.1 Head production section : external behavior-STD, organizational view

The external behavior STD shows the possible sequences in which the exported operation(s) can be started. The class ‘head production section’ has four operations relevant for the process fragment ‘writing project management documents’, phase2. The organizational view of the external STD does not show any communication details. Consequently the STD consists of one state ‘neutral’ in which the customer waits for a call to one of its its exported operations ‘hprs_second_estimate_(x)’, ‘hprs_consult_(x)’, ‘hprs_final_review_(x)’ or ‘hprs_pmm_request_approval_(x)’. The possible starting sequence specified by this STD is any combination of the four exported operations.

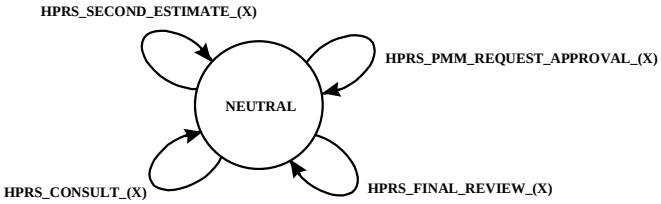


figure 5.120 head production section : external behavior STD, organizational view

5.3.7.5.2 Head production section : external behavior-STD, communicative view

The communicative view of the external behavior STD shows the communication details between the external STD and the called operation(s). It consists of a neutral state in which the customer waits for a call to its operation and four starting states in which the called operations are started. Typically the customer does not wait in this ‘starting’ states until the called operation is finished, but returns as soon as possible to its neutral state to allow handling of another call.

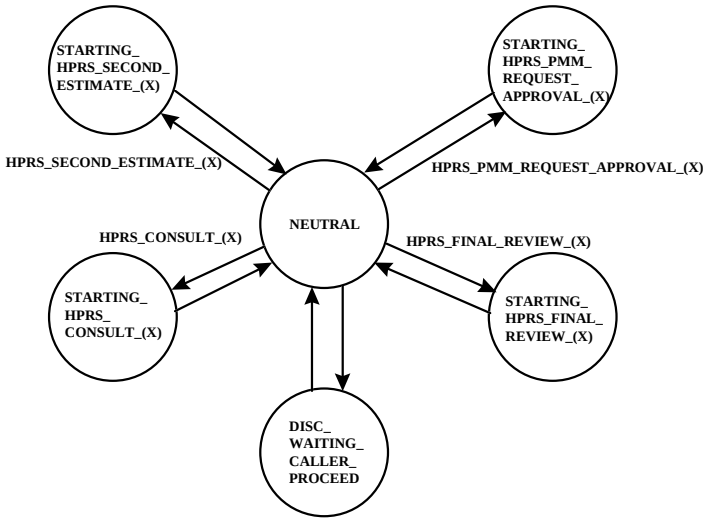


figure 5.121 head production section : external behavior STD, communicative view

When ‘hps_second_estimate_(x)’, ‘hprs_consult_(x)’ or ‘hprs_final_review_(x)’ are called, the callers must wait for the result that is produced by the callees. When a callee has produced an answer, the manager transits to the state ‘disc_waiting_ caller_proceed’. This state‘disc_waiting_caller_proceed’ is an aggregate state in which the manager determines which waiting caller had called the callee that produced the result. This caller is then allowed to proceed.

The caller of the operation ‘hprs_pmm_request_approval_(x)’ does not have to wait for a result but may continue immeditiately after its callee has been started by the manager.

The callers of the operations of this class can be found in the relevant import-export diagram. They are given in the ‘import_list’ attribute of the ‘uses associations’.

5.3.7.5.3 Head production section : internal behavior-STDs

The customer class has 4 internal operations relevant for this phase 2. These are ‘hprs_second_estimate_(x)’, ‘hprs_consult_(x)’, ‘hprs_final_review_(x)’ and ‘hprs_pmm_request_approval_(x)’. These operations have the following internal behavior STDs.

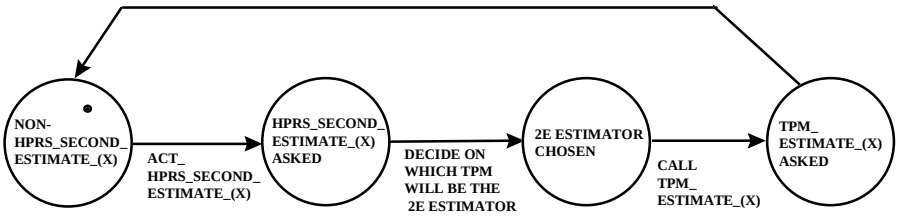


figure 5.122 int-hprs_second_estimate_(x) : internal behavior STD

When the ‘head production section’ is asked by a technical project manager for a second estimate on a project for which the tpm is writing a proposal, this operation is started. In it the ‘head production section’ chooses another tpm as second estimator and orders him to produce a (second) estimate (call tpm_estimate_(x)). When the second tpm returns his estimate to the ‘head production section’, the ‘head production section’ communicates this estimate to the first tpm. The parameter ‘x’ is the requirements document on which the estimate is to be based.

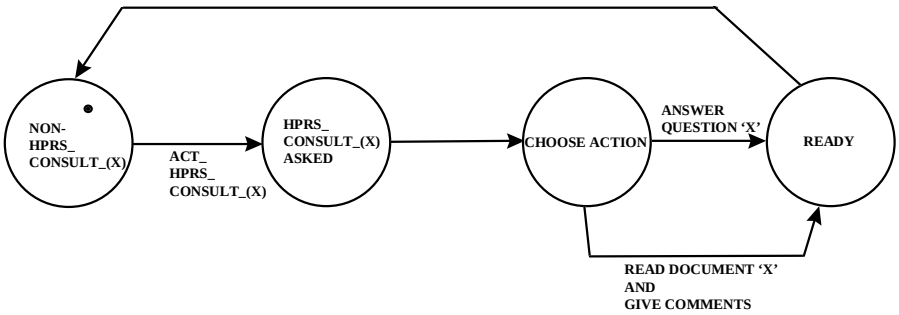


figure 5.123 int-hprs_consult_(x) : internal behavior STD

The operation ‘hprs_consult_(x)’ is called by a tpm who is busy writing a project management document. Its use is either to solicit comments of the ‘head production section’ on an intermediate version of the project management document (and at the same time informing the ‘head production section’ of the progress made so far in writing the document) or to ask some advice on a technical matter from the ‘head production section’. The ‘head production section’ reacts by reading the document and giving comments or by just answering the question. The parameter ‘x’ is either the project management document or the question asked.

The multiplicity of concurrent executing instances is zero or more. The ‘head production section’ has many technical project managers working for him who can all seek advice from him at the same time.

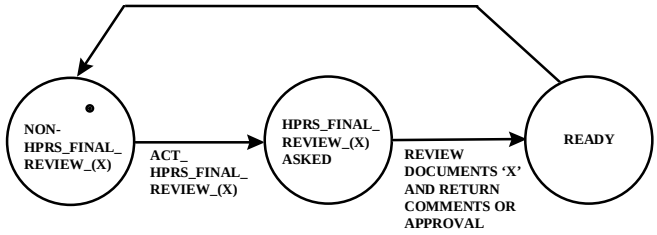


figure 5.124 int-hprs_final_review_(x) : internal behavior STD

With the operation ‘hprs_final_review_(x)’ the ‘head production section reviews the final versions of the three project management documents and gives his comments. If there are no comments, the documents are approved. The technical project manager presents the ‘head production section’ with all three project management documents (project contract,

software development plan and ‘internal resources allocation’-document) together at the same time. The parameter ‘x’ includes all three documents.

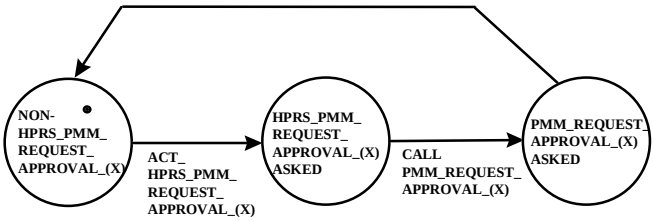


figure 5.125 int-hprs_pmm_request_approval_(x) : internal behavior STD

The technical project manager gives the three project management documents to his ‘head production secion’ for management approval. The ‘head production section’ thereupon enters the three project management documents in the next ‘project meeting minus’ for approval by the participants of that meeting (call pmm_request_approval_(x)). The formal parameter ‘x’ constitutes the three project management documents to be approved.

5.3.7.5.4 Head production section : manager-STD

The communication between the head production section’s operations (callees) and its callers is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

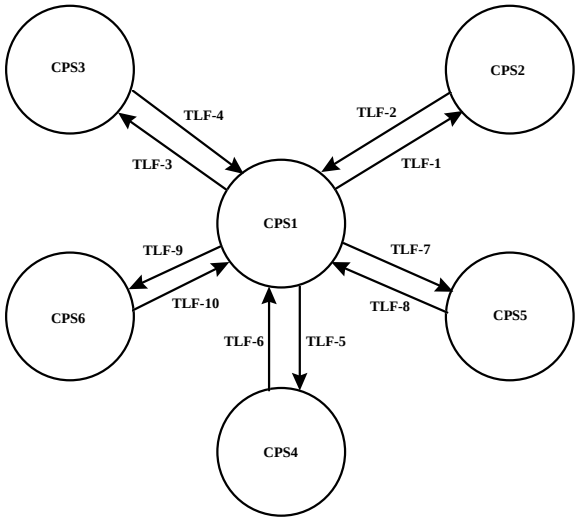


figure 5.126 head production section : manager STD

The notation CPSx in the STD stands for Consolidated Prescribed Subprocesses and is a set of CCx’s. The notation CCx in the STD stands for ‘Caller(s)-Callee’ and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for ‘Traps Logical Formula’ and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

The callers of the operations ‘hps_second_estimate_(x)’, ‘hprs_consult_(x)’ and ‘hprs_final_review_(x)’ have to wait for the return results of the callees. This is modeled conform the second variant of the caller-waits construct. This means that the TLF-3, TLF-5, TLF-7 and TLF-9 have to have some additional information for the manager to decide which transition to take. This information comes from the internal bookkeeping of the manager. If an operation is started by the manager on behalf of a caller, a caller-callee relation is initiated. In this way the manager can check whether a non-active employee still has some caller waiting to be allowed to proceed. If both the callee and the caller have terminated, the particular caller-callee relation is cancelled in the internal administration of the manager.

In the state ‘neutral’ the CPS and the TLFs for the transitions leaving the state are :

CPS1 = {CC1-1, CC2-1, CC3-1, CC4-1}
TLF-1 = T-1 and T-3
TLF-3 = T-5 and (T-7 and not(caller-callee relation))
TLF-5 = (T-5 and (T-7 and (caller-callee relation))) or
(T-9 and (T-11 and (caller-callee relation))) or
(T-13 and (T-15 and (caller-callee relation)))
TLF-7 = T-9 and (T-11 and not (caller-callee relation))
TLF-9 = T-13 and (T-15 and not(caller-callee relation))

In the state 'starting_hprs_pmm_request_approval_(x)' the CPS and the TLFs for the transitions leaving the state are :

CPS2 = {CC1-2, CC2-1, CC3-1, CC4-1}
TLF-2 = T-2 and T-4

In the state 'starting_hprs_second_estimate_(x)' the CPS and the TLFs for the transitions leaving the state are :

CPS3 = {CC1-1, CC2-2, CC3-1, CC4-1}
TLF-4 = T-6

In the state 'starting_hprs_final_review_(x)' the CPS and the TLFs for the transitions leaving the state are :

CPS5 = {CC1-1, CC2-1, CC3-2, CC4-1}
TLF-8 = T-10

In the state 'starting_hprs_consult_(x)' the CPS and the TLFs for the transitions leaving the state are :

CPS6 = {CC1-1, CC2-1, CC3-1, CC4-2}
TLF-10 = T-14

The state 'disc_waiting_caller_proceed' is entered by the manager after it detects that some employee has returned a result. The manager decides in this state which caller is allowed to proceed. The CPS and the TLFs for the transitions leaving the state are :

CPS4 = {CC1-1, CC2-3, CC3-1, CC4-1} or
{CC1-1, CC2-1, CC3-3, CC4-1} or
{CC1-1, CC2-1, CC3-1, CC4-3}
TLF-6 = T-8 or T-12 or T-16

The caller-callee combinations for 'hprs_pmm_request_approval_(x)' and its caller 'tpm_write_proposal_(x)' are :

CC1-1 = {S1, S3}
CC1-2 = {S2, S4}

The caller-callee combinations for 'hprs_second_estimate_(x)' and its caller 'tpm_write_proposal_(x)' are :

CC2-1 = {S5, S7}
CC2-2 = {S6, S7} (the caller has to wait)
CC2-3 = {S5, S8} (the caller may proceed)

The caller-callee combinations for 'hprs_final_review_(x)' and its caller 'tpm_write_proposal_(x)' are :

CC3-1 = {S9, S11}
CC3-2 = {S10, S11} (the caller has to wait)
CC3-3 = {S9, S12} (the caller may proceed)

The caller-callee combinations for 'hprs_consult_(x)' and its caller 'tpm_write_proj_man_doc_(x,y)' are :

CC4-1 = {S13, S15}
CC4-2 = {S14, S15} (the caller has to wait)
CC4-3 = {S13, S16} (the caller may proceed)

5.3.7.5.5 Head production section : employee-STDs

The manager STD has 6 employees relevant for this phase. These are the internal operations ‘hprs_pmm_request_approval_(x)’, ‘hprs_second_estimate_(x)’ and ‘hprs_final_review_(x)’ which are all called by the operation ‘tpm_write_proposal_(x)’ at some point during its execution. Plus the internal operation ‘hprs_consult_(x)’ and its caller ‘tpm_write_proj_man_doc_(x,y)’.

The first employee is ‘hprs_pmm_request_approval_(x)’. This employee has two subprocesses S1 and S1 and two traps T-1 and T-2 according to the caller-callee construct. The second employee is the operation ‘tpm_write_proposal_(x)’ in its role of caller of ‘hprs_pmm_request_approval_(x)’. This second employee has the subprocesses S3 and S4 and traps T-3 and T-4 according to the caller-callee construct.

The third employee is ‘hprs_second_estimate_(x)’. This employee has two subprocesses S5 and S6 and two traps T-5 and T-6 according to the caller-callee construct. Its caller is again the operation ‘tpm_write_proposal_(x)’. This operation has therefore in this respect the subprocesses S7 and S8 and the traps T-7 and T-8 according to the caller-callee construct.

The fourth employee is ‘hprs_final_review_(x)’. This employee has two subprocesses S9 and S10 and two traps T-9 and T-10 according to the caller-callee construct. Its caller is again the operation ‘tpm_write_proposal_(x)’. This operation has therefore in this respect the subprocesses S11 and S12 and the traps T-11 and T-12 according to the caller-callee construct.

The fifth employee is the operation ‘hprs_consult_(x)’. This employee has two subprocesses S13 and S14 and two traps T-13 and T-14 according to the caller-callee construct. Its caller is the sixth employee. It is the operation ‘tpm_write_proj_man_doc_(x,y)’ of the class technical project manager. This employee has the subprocesses S15 and S16 and the traps T-15 and T-16 according to the caller-callee construct.

5.3.7.6 Head support section

5.3.7.6.1 Head support section : external behavior-STD, organizational view

The external behavior STD shows the possible sequences in which the exported operation(s) can be started. The class 'head support section' has only one operation relevant for the process fragment 'writing project management documents', phase2. The organizational view of the external STD does not show any communication details. Consequently the STD consists of one state 'neutral' in which the 'head support section' waits for a call to its exported operation 'hss_inquire_resource_(x)'. If the call has taken place, the 'head support section' can make the transition labeled 'hss_inquire_resource_(x)'. The 'head support section' then comes again in the state 'neutral'. The possible starting sequence specified by this STD is thus 'hss_inquire_resource_(x)', 'hss_inquire_resource_(x)', etc.

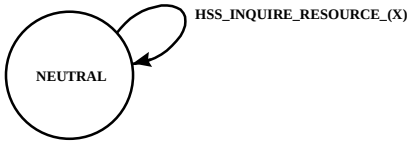


figure 5.127 head support section : external behavior STD, organizational view

5.3.7.6.2 Head support section : external behavior STD, communicative view

The communicative view of the external behavior STD shows the communication details between the external STD and the called operation(s).

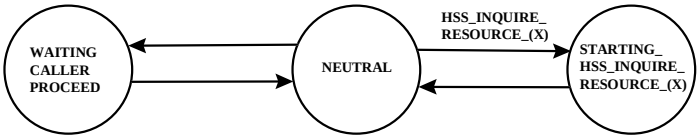


figure 5.128 head support section : external behavior STD, communicative view

It consists of a neutral state in which the 'head support section' waits for a call to its operation and a state in which he starts this operation. The 'head support section' does not wait in this 'starting' state for the called operation to finish, but returns as soon as possible to its neutral state. It can then handle another call to the operation before the current execution has finished.

The caller of the operation has to wait for the callee to return a result before it may proceed. The manager detects that the callee has produced a result, transits to the state 'waiting caller proceed' and there allows the caller to proceed.

The caller of the operation of this class can be found in the import-export diagram. It can be found in the the 'import_ list' attribute of the 'uses association'.

5.3.7.6.3 Head support section : internal behavior-STDs

The 1 operation 'hss_inquire_resource_(x)' of the 'head support section' has the following internal behavior STD.

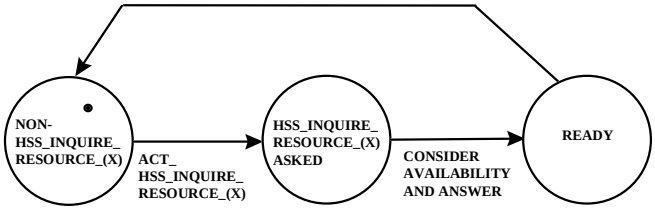


figure 5.129 int-hss_inquire_resource_(x) : internal behavior STD

With this operation the ‘head support section’ is asked if there is a certain resource ((software)engineers, computer equipment, support software, person-hours of the computer support section, work locations, office equipment or person-hours of the controller section) available during a certain time frame. This is reflected in the parameter ‘x’.

The ‘head suppport section’ considers this request and answers according to the availablity of the resource in question. The multiplicity of concurrent executing STD instances is zero or more. This means that the ‘head support section’ can handle inquiries made by several technical project managers at the same time.

5.3.7.6.4 Head support section : manager-STD

The communication between the head support section’s operation (callee) and its caller is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

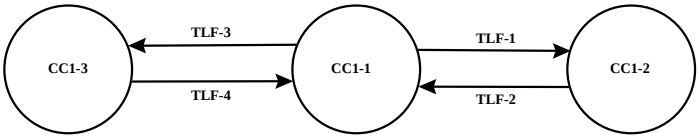


figure 5.130 head support section : manager STD

The notation CCx in the STD stands for ‘Caller(s)-Callee’ and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for ‘Traps Logical Formula’ and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

There is only one caller-callee combination, the callee ‘hss_inquire_resource_(x)’ and its caller ‘tpm_write_proj_ man_doc_(x,y)’ of the class ‘technical project manager’. Because the caller has to wait for the result produced by the callee, the call is modeled conform the second ‘caller waits’-variant of the caller-callee construct. This means that the TLF-1 and TLF-3 have to have some additional information for the manager to decide which transition to take. This information comes from the internal bookkeeping of the manager. If an operation is started by the manager on behalf of a caller, a caller-callee relation is initiated. In this way the manager can check whether a non-active employee still has some caller waiting to be allowed to proceed. If both the callee and the caller have terminated, the particular caller-callee relation is cancelled in the internal administration of the manager.

In the state ‘neutral’ the CC and the TLF for the transition leaving the state are :

CC1-1 = {S1, S3}
TLF-1 = T-1 and (T-3 and not(caller-callee relation))
TLF-3 = T-1 and (T-3 and (caller-callee relation))

In the state ‘starting_hss_inquire_resource_(x)’ the CC and the TLF for the transition leaving the state are :

CC1-2 = {S2, S3} (caller has to wait)
TLF-2 = T-2

In the state ‘waiting_caller_proceed’ the CC and the TLF for the transition leaving the state are :

CC1-3 = {S1, S4} (caller may proceed)
TLF-4 = T-4

5.3.7.6.5 Head support section : employee-STDs

The manager STD has the following 2 employee STDs. The first employee is ‘hss_inquire_resource_(x)’. This employee has two subprocesses S1 and S1 and two traps T-1 and T-2 according to the caller-callee construct. The second employee is the caller ‘tpm_write_proj_man_doc(x,y)’. This second employee has the subprocesses S3 and S4 and traps T-3 and T-4 according to the caller-callee construct.

5.3.7.7 Project management document

5.3.7.7.1 Project management document : external behavior-STD, organizational view

The external behavior STD shows the possible sequences in which the exported operation(s) can be started. The class ‘project management document’ has only one operation relevant for the process fragment ‘writing project management documents’, phase2. The organizational view of the external STD does not show any communication details. Consequently the STD consists of one state ‘neutral’ in which the ‘project management document’ waits for a call to its exported operation ‘pmd_write_(x)’. If the call has taken place, the ‘project management document’ can make the transition labeled ‘pmd_write_(x)’. It then comes again in the state ‘neutral’. The possible starting sequence specified by this STD is thus ‘pmd_write_(x)’, ‘pmd_write_(x)’, etc.

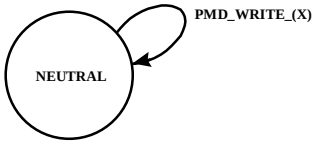


figure 5.131 project management document : external behavior STD, organizational view

5.3.7.7.2 Project management document : external behavior STD, communicative view

The communicative view of the external behavior STD shows the communication details between the external STD and the called operation(s).

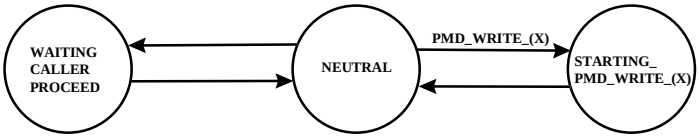


figure 5.132 project management document : external behavior STD, communicative view

It consists of a neutral state in which the ‘project management document’ waits for a call to its operation and a state in which he starts this operation. The ‘project management document’ does not wait in this ‘starting’ state for the called operation to finish, but returns as soon as possible to its neutral state. It can then handle a call to the one of its other operations (For this phase 2 it has no other operations, but it is possible that it has some operations relevant to another process fragment. These operations will not be modeled here.) The ‘project management document’ will not honor another call to ‘pmd_write_(x)’ while the operation is still executing. The operation ‘pmd_write_(x)’ has a multiplicity of concurrent executing STD instances of zero or 1. This means that the ‘write’-actions on the document only can take place one after the other (sequential) and not in parallel.

The caller of the operation has to wait for the callee to acknowledge that it has mutated the project management document before it may proceed. When the manager STD detects that the callee has indeed progressed far enough in its execution, it will transit to the state ‘waiting caller proceed’. In this state it will allow the caller to proceed.

5.3.7.7.3 Project management document : internal behavior-STDs

The 1 operation ‘pmd_write_(x)’ of the ‘project management document’ has the following internal behavior STD.

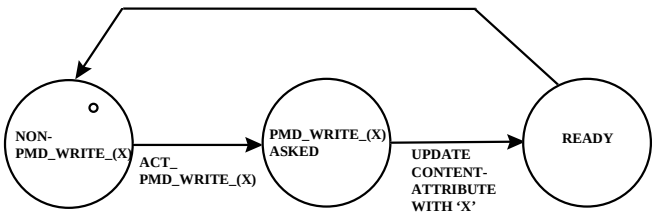


figure 5.133 int-pmd_write_(x) : internal behavior STD

The operation updates the content attribute of the project management document-object with the actual value of the parameter ‘x’.

The multiplicity of concurrent executing STD instances is zero or 1. This is indicated by the hollow circle inside the first state of the STD. This serializes the ‘writes’ in the project management document. I.e. the ‘writes’ can not happen in parallel, but only sequential.

5.3.7.7.4 Project management document : manager-STD

The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

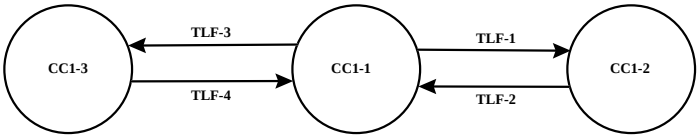


figure 5.134 project management document : manager STD

The notation CCx in the STD stands for ‘Caller(s)-Callee’ and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for ‘Traps Logical Formula’ and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

Because the caller has to wait until the callee has updated the project management document, the call is modeled conform the second ‘caller waits’-variant of the caller-callee construct. This means that the TLF-1 and TLF-3 have to have some additional information for the manager to decide which transition to take. This information comes from the internal bookkeeping of the manager. If an operation is started by the manager on behalf of a caller, a caller-callee relation is initiated. In this way the manager can check whether a non-active employee still has some caller waiting to be allowed to proceed. If both the callee and the caller have terminated, the particular caller-callee relation is cancelled in the internal administration of the manager.

In the state ‘neutral’ the CC and the TLF for the transition leaving the state are :

CC1-1 = {S1, S3}
TLF-1 = T-1 and (T-3 and not(caller-callee relation))
TLF-3 = T-1 and (T-3 and (caller-callee relation))

In the state ‘starting_pmd_write_(x)’ the CC and the TLF for the transition leaving the state are :

CC1-2 = {S2, S3} (caller has to wait)
TLF-2 = T-2

In the state ‘waiting_caller_proceed’ the CC and the TLF for the transition leaving the state are :

CC1-3 = {S1, S4} (caller may proceed)
TLF-4 = T-4

5.3.7.7.5 Project management document : employee-STDs

The manager STD has the following 2 employee STDs. The first employee is ‘pmd_write_(x)’. This employee has two subprocesses S1 and S1 and two traps T-1 and T-2 according to the caller-callee construct. The second employee is the caller ‘tpm_write_proj_man_doc(x,y)’. This second employee has the subprocesses S3 and S4 and traps T-3 and T-4 according to the caller-callee construct.

5.3.7.8 Project meeting minus

5.3.7.8.1 Project meeting minus : external behavior-STD, organizational view

The external behavior STD shows the possible sequences in which the exported operation(s) can be started. The class ‘project meeting minus’ has only one operation relevant for the process fragment ‘writing project management documents’, phase 2. The organizational view of the external STD does not show any communication details. Consequently the STD consists of one state ‘neutral’ in which the ‘project meeting minus’ waits for a call to its exported operation ‘pmm_request_approval_(x)’. If the call has taken place, the ‘project meeting minus’ can make the transition labeled the operation name. The ‘project meeting minus’ then comes back again in the state ‘neutral’. The possible starting sequence specified by this STD is thus ‘pmm_request_approval_(x)’, ‘pmm_request_approval_(x)’, etc.

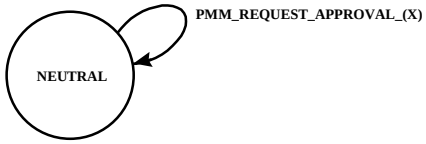


figure 5.135 project meeting minus : external behavior STD, organizational view

5.3.7.8.2 Project meeting minus : external behavior-STD, communicative view

The communicative view of the external behavior STD shows the communication details between the external STD and the called operation(s).

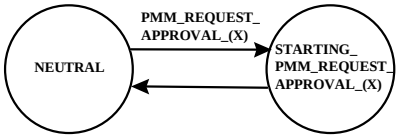


figure 5.136 project meeting minus : external behavior STD, communicative view

This external behavior STD has two states. In the state neutral the ‘project meeting minus’ waits until a call has been placed to its operation ‘pmm_request_approval_(x). If a call has been made, the manager STD can (and eventually will) transit to the state ‘starting_pmm_request_approval_(x)’. If the operation has been started the manager can transit back to the state neutral.

The caller of the operation does not have to wait for the result of the ‘project meeting minus’. This is because the projecmanagement documents have already ondergone extensive review by the heads of the support sections and by the customer. The approval is in fact only a formality.

The caller of the operation is ‘hprs_pmm_request_approval_(x)’. This is documented in the import_export diagram. It is presented there as a value of the ‘import_list’ attribute of the relevant ‘uses association’.

5.3.7.8.3 Project meeting minus : internal behavior-STDs

The ‘project meeting minus’ has 1 operation relevant for phase 2 of the process fragment ‘writing project management documents’ : ‘pmm_request_approval_(x)’. Only a ‘view’ of the internal STD of this operation is shown here. The full STD is described in the modeling of phase 3. A ‘view’ on an STD shows only the transitions and (aggregated) states that are important for a certain ‘user’ of that STD. (See also the explanation on ‘views’ in the SOCCA chapter). The view shown here is the view of the external STD on his internal STD. Only the one transition ‘act_pmm_request_approval _(x)’ is relevant to the external STD. And only two states are seen by the external STD. Either the internal STD is non-active or the internal STD is executing.

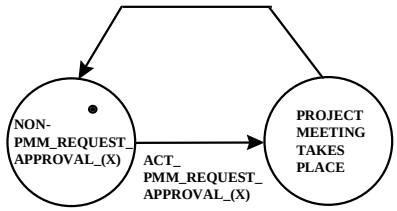


figure 5.137 int-pmm_request_approval_(x) : internal behavior STD, view

The operation ‘pmm_request_approval_(x)’ has one formal parameter ‘x’. This includes the three project management documents (software development plan, ‘internal resources allocation’-document and project contract) that the (participants of the) meeting have to approve. The functionality of the operation is described in the modeling of phase 3 of the process fragment ‘writing project management documents’. Here it is only shown as the aggregate state ‘project meeting takes place’.

5.3.7.8.4 Project meeting minus : manager-STD

The communication between the ‘project meeting minus’’s operations (callees) and their callers is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

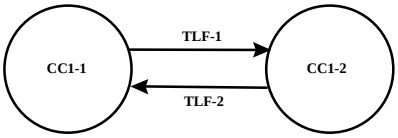


figure 5.138 project meeting minus : manager STD

The notation CCx in the STD stands for ‘Caller(s)-Callee’ and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for ‘Traps Logical Formula’ and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

There is only one caller-callee combination, the callee ‘pmm_request_approval_(x)’ and its caller ‘hprs_pmm_request_approval_(x)’ of the class ‘head production section’. The calling is modeled by the ‘caller does not wait’-variant of the caller-callee construct.

In the state ‘neutral’ the CC and the TLF for the transition leaving the state are :

CC1-1 = {S1, S3}
TLF-1 = T-1 and T-3 (corresponds with ‘pmm_request_approval_(x)’ transition in extern STD)

In the state ‘starting_pmm_request_approval_(x)’ the CC and the TLF for the transition leaving the state are :

CC1-2 = {S2, S4}
TLF-2 = T-2 and T-4

5.3.7.8.5 Project meeting minus : employee-STDs

The manager STD has 2 employee STDs. These are the callee ‘pmm_request_approval_(x)’ and its caller ‘hprs_pmm_request_approval_(x)’ of the class ‘head production section’.

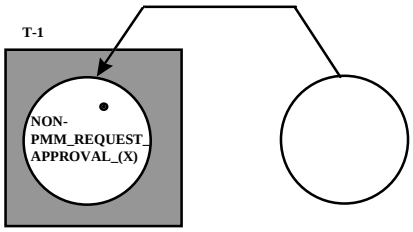


figure 5.139 employee int-pmm_request_approval_(x) : subprocess S1

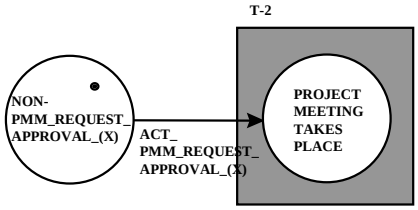


figure 5.140 employee int-pmm_request_approval_(x) : subprocess S2

The first employee, ‘pmm_request_approval_(x)’ has two subprocesses S1 and S1 and two traps T-1 and T-2 according to the caller_callee-construct.

The second employee, the caller ‘hprs_pmm_request_approval_(x)’, has two subprocesses S3 and S4 and two traps T-3 and T-4 according to the caller_callee-construct.

The manager prescribes initially subprocess S3 for the calling employee, it is waiting for the call. When the caller enters its trap T-3, i.e. excutes the call (and the callee is ready in its trap T-1) the manager can make the transition to the next state. Here it prescribes S4 for the caller, thereby allowing it to proceed in its next subprocess. This has the effect that the caller does not wait for the result of the called operation but proceeds right away after the manager has started the called operation.

When the callee has entered its trap T-2 and the caller has entered his trap T-4, the manager can transit back to the state ‘neutral’.

SOFTWARE PROCESS MODELING in SOCCA

student : H.G.Brugman
studentnr : 9105506
version : 0.10

5. Key Process Area ‘Software Project Planning’

5.3.8 State Transition Diagrams - Phase 3

Phase 3, ‘approval’, of the process fragment ‘writing project management documents’ (partly) models the behavior of the following classes :

- project meeting minus
- chief executive officer
- head support section
- quality assurance adviser
- head controller section
- account manager
- customer
- archive/documentation administrator
- technical project manager

5.3.8.1 Project meeting minus

5.3.8.1.1 Project meeting minus : external behavior-STD, organizational view

The class ‘project meeting minus’ has only one operation of interest for phase 3 of the process fragment ‘writing project management documents. This is ‘pmm_request_approval_(x)’. This operation is called in phase 2 of the process fragment. The external STD, organizational view, is already given during the modeling of phase 2. See the appropriate paragraph of the phase 2 modeling.

5.3.8.1.2 Project meeting minus : external behavior-STD, communicative view

The class ‘project meeting minus’ has only one operation of interest for phase 3 of the process fragment ‘writing project management documents. This is ‘pmm_request_approval_(x)’. This operation is called in phase 2 of the process fragment. The external STD, communicative view, is already given during the modeling of phase 2. See the appropriate paragraph of the phase 2 modeling.

5.3.8.1.3 Project meeting minus : internal behavior-STDs

The ‘project meeting minus’ has 1 operation relevant for phase 3 of the process fragment ‘writing project management documents’ : ‘pmm_request_approval’. This operation has the following internal behavior STDs.

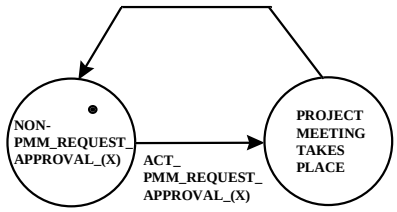


figure 5.141 int-pmm_request_approval_(x) : internal behavior STD, view

In the description of the operation ‘pmm_request_approval_(x)’ in phase 2 only a view of its internal STD was given (see figure above). Here the complete STD is given. To indicate how the view corresponds to the full STD, the aggregated state ‘project meeting takes place’ is shown in the full STD (see figure below).

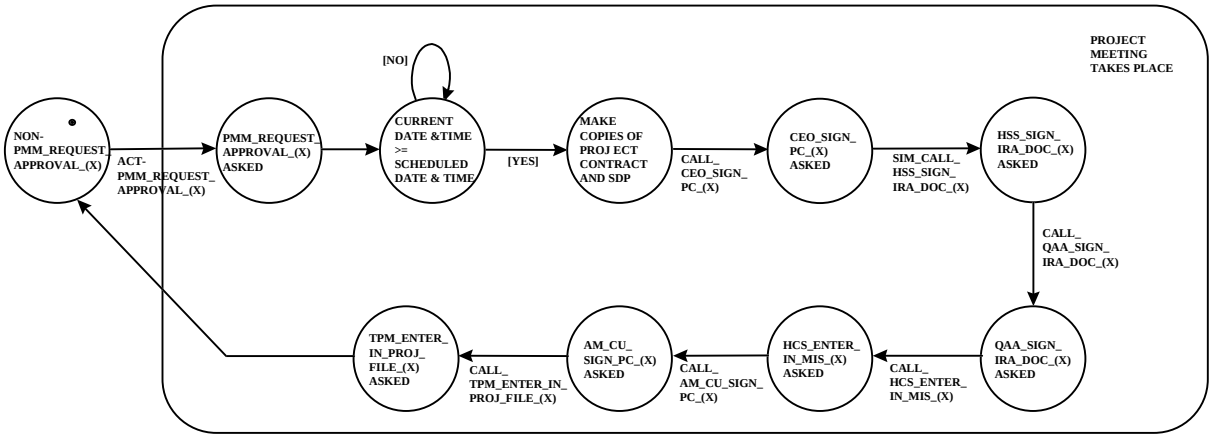


figure 5.142 int-pmm_request_approval_(x) : internal behavior STD

The operation ‘pmm_request_approval_(x)’ has one formal parameter ‘x’. This signifies the project management documents which the meeting has to approve. The project management documents encompass the software development plan (sdp), the ‘internal resources allocation’-document (ira-doc) and the project contract (pc). The multiplicity of

concurrent executing STD instances is zero or more. This means that the ‘project meeting minus’ can handle management documents of more than one project at the same time. This is usually the case.

When an instance of the operation ‘pmm_request_approval_(x)’ is started, it goes to the its state ‘current date & time >= scheduled time & date’. There it compares the current time and date with the time and date the meeting is scheduled to start. This way the ‘project meeting minus’ starts on it scheduled time. If there are more instances executing (i.e. more projects want their documents approved at the meeting), they all wait for the scheduled time and are thus synchronized. The meeting can then handle the documents of more then one project a the same time.

It is presumed that the current date and time is available for inspection by the ‘project meeting minus’-object. This is not modeled in detail.

The project management documents that are handled by the ‘project meeting minus’ are a project contract, a software development plan and a ‘internal resources allocation’-document (ira-document). The ira-document contains the project form as an appendix.

When the meeting has started, a copy is made of the project contract and the software development plan. For this copying the ‘project meeting minus’ uses the method ‘pmd_copy’ of the class ‘project management document’. This is not further detailed, but shown here as an aggregate (higher level) state.

The project contract and its copy are then given to the chief excutive officer for his signature (call_ceo_sign_(x)).

Then each of the support section heads (head controller section, head personnel section, head computer support section and head infrastructure section) are given that part of the ‘internal resources allocation’-document which concerns his section. They all sign their respective parts if the ira-document. They will do this in parallel.This is achieved by placing a simultaneous call (sim_call) to all four heads (sim_call_hss_sign_(x)).

Next the quality assurance part of the ira-document will be given to the quality assurance adviser. He will sign this part of the document.

Then the updated project form is given to the head of the controller section. The project form is an appendix to te ‘ira’-document and has been updated by the technical project manager during the writing of the ‘ira’- document. The updated data in the project form is entered by the (personnel of the) controller section into the Management Information System (MIS) of the Waco Business Unit (WBU).

At this point in its execution, the ‘project meeting minus’ has to wait until all documents are signed. After the signed documents become available, the account manager is given the two copies of the project contract (with the software development plan as appendix). The account manager will approach the customer for his signature on the project contract.

The ira-documents (containing the signed parts) is given back to the technical project manager of the project. He will file the document in the appropriate ‘project file’.

5.3.8.1.4 Project meeting minus : manager-STD

The class ‘project meeting minus’ has only one operation of interest for phase 3 of the process fragment ‘writing project management documents. This is ‘pmm_request_approval_(x)’. This operation is called in phase 2 of the process fragment. The manager STD is already given during the modeling of phase 2. See the appropriate paragraph of the phase 2 modeling.

5.3.8.1.5 Project meeting minus : employee-STDs

The manager STD has 2 employee STDs. These are the callee ‘pmm_request_approval_(x)’ and its caller ‘hprs_ pmm_request_approval_(x)’ of the class ‘head production section’.

The employees are already given during the modeling of phase 2. See the appropriate paragraph of the phase 2 modeling.

5.3.8.2 Chief executive officer

5.3.8.2.1 Chief executive officer : external behavior-STD, organizational view

The class ‘chief executive officer’ has one operation relevant to this phase 3. This is ‘ceo_sign_pc_(x)’. The possible starting sequence of this operation is shown in the external STD. Because there is only one operation the starting sequence is simply the succession ‘ceo_sign_pc_(x)’, ‘ceo_sign_pc_(x)’, etc.

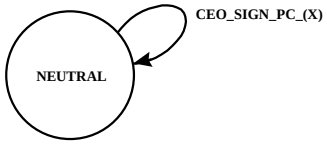


figure 5.143 chief executive officer : external behavior STD, organizational view

5.3.8.2.2 Chief executive officer : external behavior STD, communicative view

In addition to the possible starting sequence of the internal operations, the communicative view of the external behavior STD shows also communication details between external STD and the called operation(s).

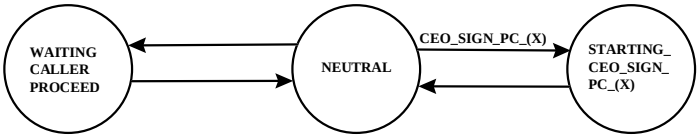


figure 5.144 chief executive officer : external behavior STD, communicative view

The STD consists of a neutral state. When in this state the object is ready to handle a call to its operation. When the call has been made, the object can go to the ‘starting’-state. Here the internal operation is started. The caller of the operation has to wait for the callee to return some result (in this case the signed project contract). When the STD notices that the callee indeed has returned its result it can transit to the state ‘waiting caller proceed’. In this state the caller is allowed to proceed with its execution.

5.3.8.2.3 Chief executive officer : internal behavior-STDs

The operation ‘ceo_sign_pc_(x)’ has the following internal behavior STD.

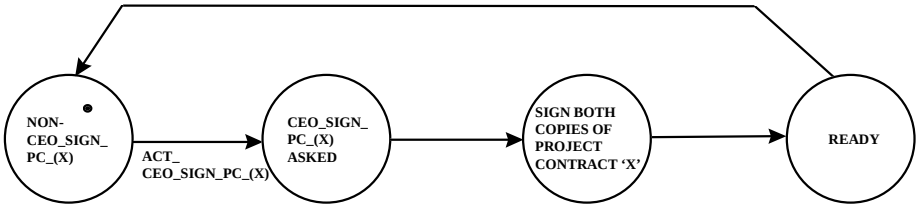


figure 5.145 int-ceo_sign_pc_(x) : internal behavior STD

With this operation the ‘chief executive officer’ (ceo) signs both copies of the project contract. For the actual signing the ceo uses the operation ‘pmd_sign’ of the class ‘project management document’. This is not modeled in detail, but shown here as an aggregate (high level) state.

The parameter ‘x’ contains the object id-s of both the project contracts to be signed. Because the ceo can sign the project contracts of more then one project in parallel, the multiplicity of concurrent executing STDs is zero or more.

5.3.8.2.4 Chief executive officer : manager-STD

The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

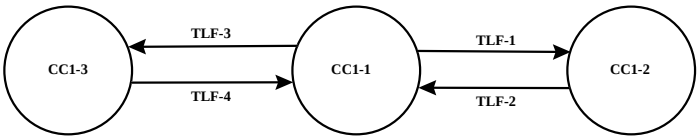


figure 5.146 chief executive officer : manager STD

The notation CCx in the STD stands for 'Caller(s)-Callee' and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for 'Traps Logical Formula' and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

There is only one caller-callee combination, the callee 'ceo_sign_pc_(x)' and its caller 'pmm_request_approval_(x)' of the class 'project meeting minus'. Because the caller may only proceed for four more states after the call and then has to wait for the result produced by the callee, the call is modeled conform the second 'caller waits'-variant of the caller-callee construct. The trap of the subprocess S3 of the caller contains four states. This reflects the fact that the caller may proceed four states after placing the call.

The manager keeps track of which callee is started for which caller. This infomation is called the 'caller-callee' relation. In the second variant of the caller waits-construct the manager can not decide by the trap-information alone if it has to make the transition TLF-1 or TLF-3 (the trap-information is the same for both transitions). The manager uses the caller-callee relation to distinguish between both transitions.

In the state 'neutral' the CC and the TLF for the transition leaving the state are :

CC1-1 = {S1, S3}
TLF-1 = T-1 and (T-3 and not(caller-callee relation))
TLF-3 = T-1 and (T-3 and (caller-callee relation))

In the state 'starting_ceo_sign_pc_(x)' the CC and the TLF for the transition leaving the state are :

CC1-2 = {S2, S3} (caller may proceed four states and then has to wait)
TLF-2 = T-2

In the state 'waiting_caller_proceed' the CC and the TLF for the transition leaving the state are :

CC1-3 = {S1, S4} (caller may proceed)
TLF-4 = T-4

5.3.8.2.5 Chief executive officer : employee-STDs

The manager STD has the following 2 employee STDs. The first employee is 'ceo_sign_pc_(x)'. This employee has two subprocesses S1 and S1 and two traps T-1 and T-2 according to the caller-callee construct.

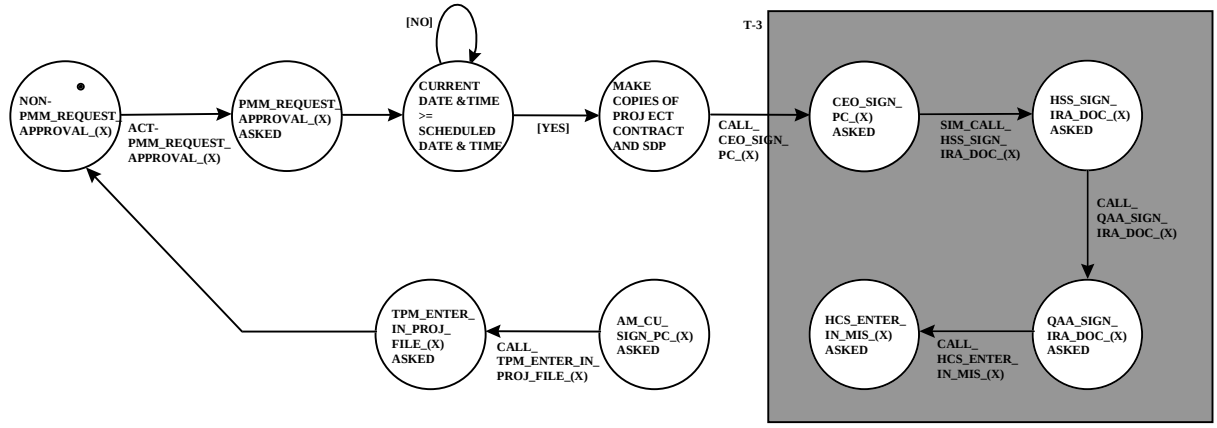


figure 5.147 employee int-pmm_request_approval_(x) : subprocess S3

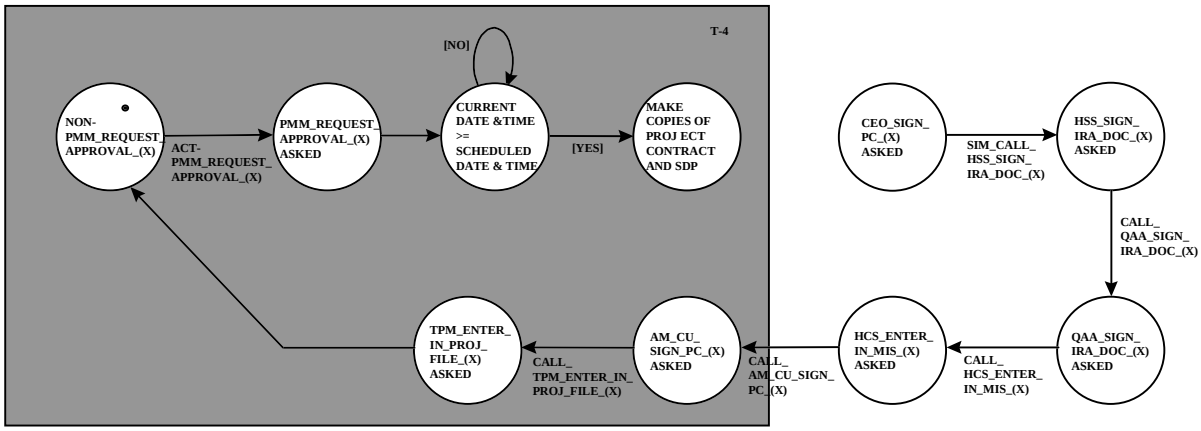


figure 5.148 employee int-pmm_request_approval_(x) : subprocess S4

The second employee is the caller ‘pmm_request_approval_(x)’. This second employee has the subprocesses S3 and S4 and traps T-3 and T-4 according to the caller-callee construct. Because the caller may proceed for four states after the call, the trap T-3 contains four states. Accordingly the trap T-4 contains the remaining states.

Initially the operation ‘ceo_sign_pc_(x)’ is not executing. The manager prescribes S3 for the caller (the manager is waiting for the call). It also prescribes S1 for the callee (the manager is waiting for the callee to be ready to be started). When the caller enters its trap T-3, i.e. excutes the call (and the callee is ready in its trap T-1) the manager can make the transition to the state ‘starting_ceo_sign_pc_(x)’. Here it prescribes S2 for the callee, thereby starting it. It also still prescribes S3 for the caller. This allows the caller to proceed four states after the call. Then it can proceed no further.

When the manager transits to the starting state it will record the caller-callee relation. That is to say, it will remember on behalf of which caller the callee is starting execution.

When in the state ‘starting_ceo_sign_(x)’ the callee enters the trap T-2 (signaling that it has started execution), the manager can (and eventualy will) transit back to neutral.

Here it prescribes S1 for the callee. Which means that the callee just continues excuting. The manager prescribes S3 for the caller, thereby allowing it to proceed for a maximum of the next four states..

When the callee finishes execution (indicated by its entering the trap T-1), the manager can transit to the state ‘waiting caller proceed’. Here it prescribes S4 for the caller. This means that the caller can start executing again if it already had reached the last of the four states allowed in S3. If the caller had not reached the fourth state yet, it means that it can just continue after it has reached that fourth state.

When the caller enters now trap T-4 in its prescribed behavior restriction S4, the manager can and will transit back to the neutral state. On this transition the manager deletes the caller-callee relation and is ready for the next cycle.

5.3.8.3 Head support section

5.3.8.3.1 Head support section : external behavior-STD, organizational view

The class ‘head support section’ has one operation relevant to this phase 3. This is ‘hss_sign_ira_doc_(x)’. The possible starting sequence of this operation is shown in the external STD. Because there is only one operation the starting sequence is simply a succession of ‘ceo_sign_pc_(x)’-operations.

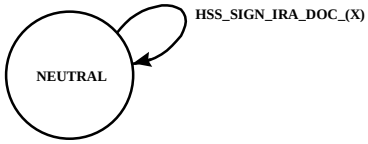


figure 5.149 head support section : external behavior STD, organizational view

5.3.8.3.2 Head support section : external behavior STD, communicative view

In addition to the possible starting sequence of the internal operations, the communicative view of the external behavior STD shows also communication details between external STD and the called operation(s).

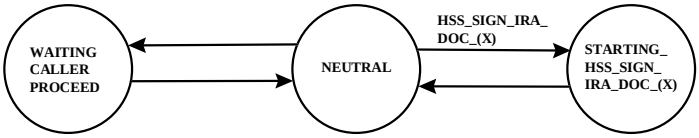


figure 5.150 head support section : external behavior STD, communicative view

The STD consists of a neutral state. When in this state the object is ready to handle a call to its operation. When the call has been made, the object can go to the ‘starting’-state. Here the internal operation is started. The caller of the operation has to wait for the callee to return some result (in this case the signed part of the ‘internal resources allocation’-document). When the STD notices that the callee indeed has returned its result it can transit to the state ‘waiting caller proceed’. In this state the caller is allowed to proceed with its execution.

5.3.8.3.3 Head support section : internal behavior-STDs

The operation ‘hss_sign_ira_doc_(x)’ has the following internal behavior STD.

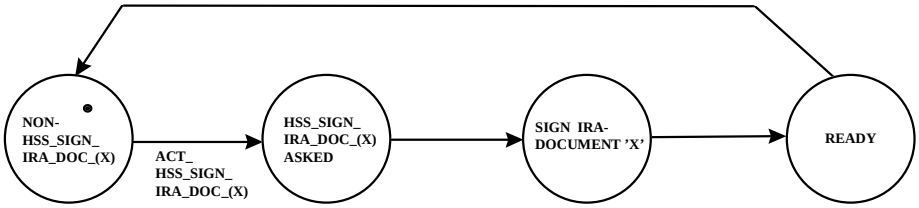


figure 5.151 int-hss_sign_ira_doc_(x) : internal behavior STD

With this operation the ‘head support section’ (hss) signs his part of the ‘internal resources allocation’-document (ira-document). For the actual signing the hss uses the operation ‘pmd_sign’ of the class ‘project management document’. This is not modeled in detail, but shown here as an aggregate (high level) state.

The parameter ‘x’ contains the object id of the ira -document that has to be signed. Because the hss can sign ira-documents of more then one project in parallel, the multiplicity of concurrent executing STDs is zero or more.

5.3.8.3.4 Head support section : manager-STD

The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

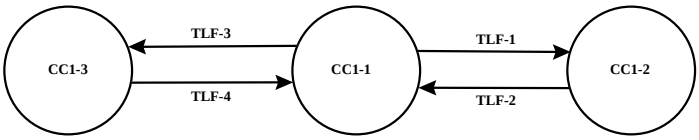


figure 5.152 head support section : manager STD

The manager STD and the subprocesses and traps of its employees are conceptualy the same, and use the same names, as those used in the modeling of the chief executive officer (in phase 3). For the definition of the CPSs, TLFs and CCs a reference is therefore made to those of the chief executive officer (in phase 3). The chief executive officer's employees must then be substituted by the head support section's employees.

There is only one caller-callee combination, the callee 'hss_sign_ira_doc_(x)' and its caller 'pmm_request_ approval_(x)' of the class 'project meeting minus'. Because the caller may proceed for three more states after the call and then has to wait for the result produced by the callee, the call is modeled conform the second 'caller waits'-variant of the caller-callee construct. The trap of the subprocess S3 of the caller contains three states. This reflects the fact that the caller may proceed three states after placing the call.

All the heads of a support section (head controller section, head personnel section, head computer support section and head infrastructure section) are asked to sign at the same time. This is modeled by a 'simultaneous' call (sim_call). The working of the sim_call is explained in the SOCCA chapter. Because the caller may proceed for three states after the call, these three states are also part of the simultaneous call-construct. When the internal STD would be given in more detail, these states are repeated at the end of every branch leaving the 'selector' state.

5.3.8.3.5 Head support section : employee-STDs

The manager STD has the following 2 employee STDs. The first employee is 'hss_sign_ira_doc_(x)'. This employee has two subprocesses S1 and S1 and two traps T-1 and T-2 according to the caller-callee construct.

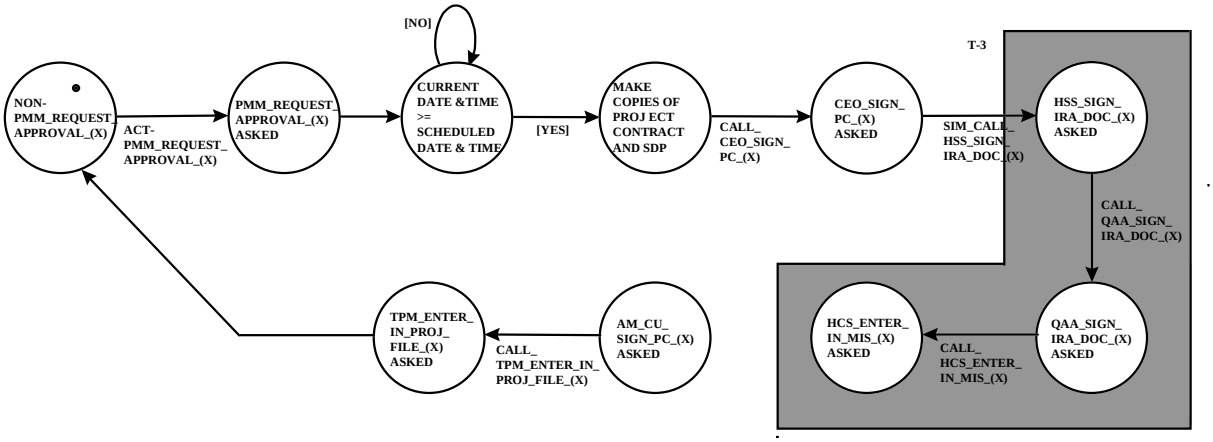


figure 5.153 employee int-pmm_request_approval_(x) : subprocess S3

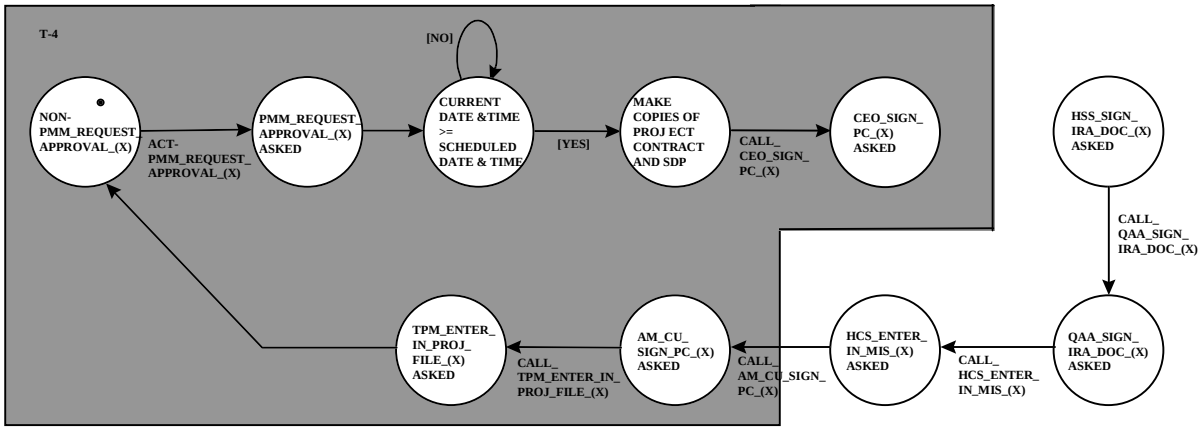


figure 5.154 employee int-pmm_request_approval_(x) : subprocess S4

The second employee is the caller ‘pmm_request_approval_(x)’. This second employee has the subprocesses S3 and S4 and traps T-3 and T-4 according to the caller-callee construct. Because the caller may proceed for three states after the call, the trap T-3 contains three states. Accordingly the trap T-4 contains the remaining states.

The manager STD and the subprocesses and traps of its employees are conceptualy the same, and use the same names, as those used in the modeling of the chief executive officer (in phase 3). For the working of the manager STD with respect to its employees a reference is therefore made to the chief executive officer (in phase 3). The chief executive officer’s employees must then be substituted by the head support section’s employees.

The working of the manager STD in the ‘simultaneous’ call construct assures that the caller STD can only proceed after all four callees (head controller section, head personnel section, head computer support section and head infrastructure section) have returned their result.

5.3.8.4 Quality assurance adviser

5.3.8.4.1 Quality assurance adviser : external behavior-STD, organizational view

The class ‘quality assurance adviser’ has one operation relevant to this phase 3. This is ‘qaa_sign_ira_doc_(x)’. The possible starting sequence of this operation is shown in the external STD. Because there is only one operation the starting sequence is simply a succession of ‘qaa_sign_ira_doc_(x)’ operations.

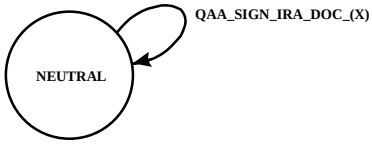


figure 5.155 quality assurance adviser : external behavior STD, organizational view

5.3.8.4.2 Quality assurance adviser : external behavior STD, communicative view

In addition to the possible starting sequence of the internal operations, the communicative view of the external behavior STD shows also communication details between external STD and the called operation(s).

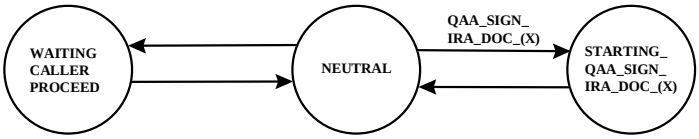


figure 5.156 quality assurance adviser : external behavior STD, communicative view

The STD consists of a neutral state. When in this state the object is ready to handle a call to its operation. When the call has been made, the object can go to the ‘starting’-state. Here the internal operation is started. The caller of the operation has to wait for the callee to return some result (in this case the signed quality-part of the ‘internal resources allocation’-document). When the STD notices that the callee indeed has returned its result it can transit to the state ‘waiting caller proceed’. In this state the caller is allowed to proceed with its execution.

5.3.8.4.3 Quality assurance adviser : internal behavior-STDs

The operation ‘qaa_sign_ira_doc_(x)’ has the following internal behavior STD.

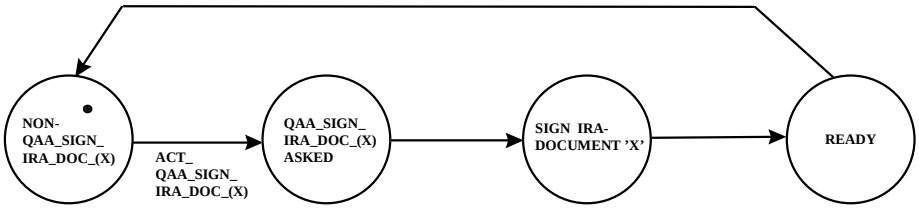


figure 5.157 int-qaa_sign_ira_doc_(x) : internal behavior STD

With this operation the ‘quality assurance adviser’ (qaa) signs his part of the ‘internal resources allocation’-document (ira-document). For the actual signing the qaa uses the operation ‘pmd_sign’ of the class ‘project management document’. This is not modeled in detail, but shown here as an aggregate (high level) state.

The parameter ‘x’ contains the object id of the ira-document that has to be signed. Because the qaa can sign ira-documents of more then one project in parallel, the multiplicity of concurrent executing STDs is zero or more.

5.3.8.4.4 Quality assurance adviser : manager-STD

The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

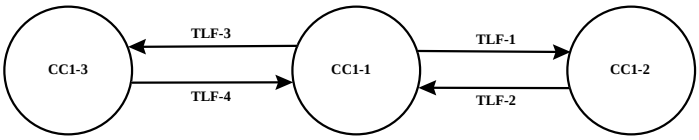


figure 5.158 chief executive officer : manager STD

There is only one caller-callee combination, the callee ‘qaa_sign_ira_doc_(x)’ and its caller ‘pmm_request_ approval_(x)’ of the class ‘project meeting minus’. Because the caller may proceed for two more states after the call and then has to wait for the result produced by the callee, the call is modeled conform the second ‘caller waits’-variant of the caller-callee construct. The trap of the subprocess S3 of the caller contains two states. This reflects the fact that the caller may proceed two states after placing the call.

The manager STD and the subprocesses and traps of its employees are conceptualy the same, and use the same names, as those used in the modeling of the chief executive officer (in phase 3). For the definition of the CCs and TLFs a reference is therefore made to those of the chief executive officer (in phase 3). The chief executive officer’s employees must then be substituted by the quality assurance adviser’s employees.

5.3.8.4.5 Quality assurance adviser : employee-STDs

The manager STD has the following 2 employee STDs. The first employee is ‘qaa_sign_ira_doc_(x)’. This employee has two subprocesses S1 and S1 and two traps T-1 and T-2 according to the caller-callee construct.

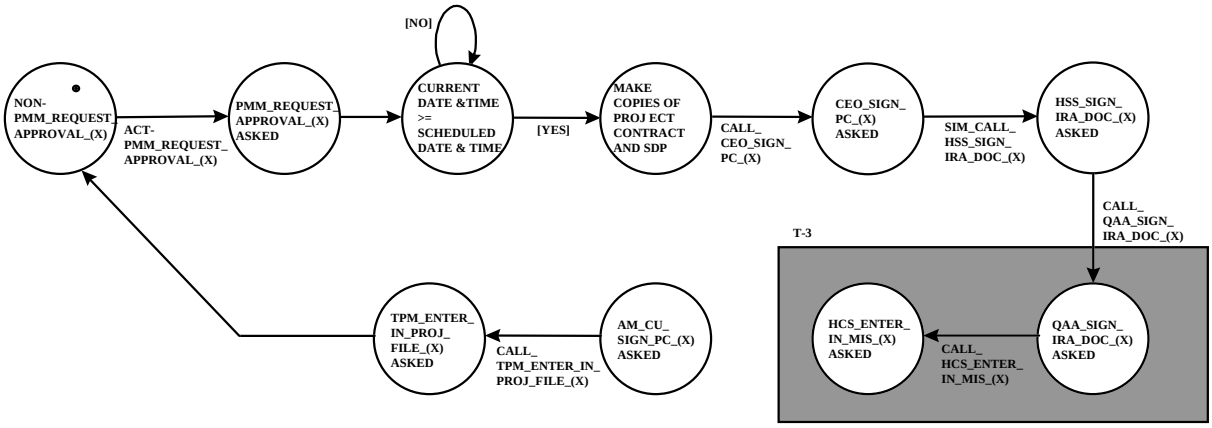


figure 5.159 employee int-pmm_request_approval_(x) : subprocess S3

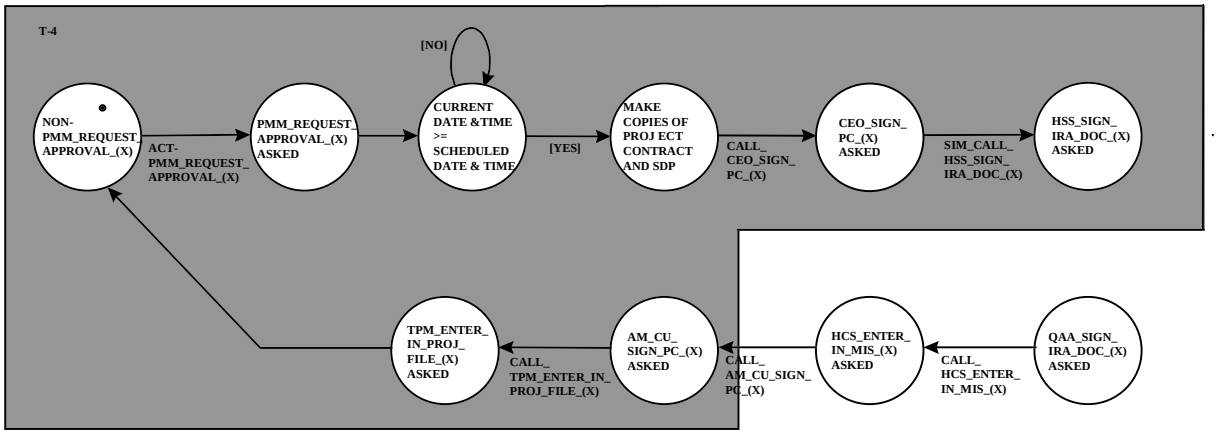


figure 5.160 employee int-pmm_request_approval_(x) : subprocess S4

The second employee is the caller ‘pmm_request_approval_(x)’. This second employee has the subprocesses S3 and S4 and traps T-3 and T-4 according to the caller-callee construct. Because the caller may proceed for two states after the call, the trap T-3 contains two states. Accordingly the trap T-4 contains the remaining states.

The manager STD and the subprocesses and traps of its employees are conceptualy the same, and use the same names, as those used in the modeling of the chief executive officer (in phase 3). For the working of the manager STD with respect to its employees a reference is therefore made to the chief executive officer (in phase 3). The chief executive officer’s employees must then be sustituted by the quality assurance adviser’s employees.

5.3.8.5 Head controller section

5.3.8.5.1 Head controller section : external behavior-STD, organizational view

The class ‘head controller section’ has one operation relevant to this phase 3. This is ‘hcs_enter_in_mis_(x)’. This operation is also used in phase 1. The only difference is in the caller of the operation. In phase 1 the caller is ‘hps_initiate_project_form_(x)’. Here, in phase 3, the caller is ‘pmm_request_approval_(x)’. The possible starting sequence of this operation is shown in the external STD. Because there is only one operation the starting sequence is simply a succession of ‘hcs_enter_in_mis_(x)’ operations.

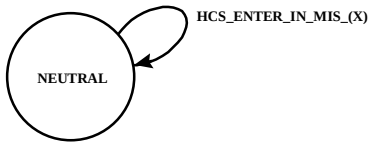


figure 5.161 head controller section : external behavior STD, organizational view

5.3.8.5.2 Head controller section : external behavior STD, communicative view

In addition to the possible starting sequence of the internal operations, the communicative view of the external behavior STD shows also communication details between external STD and the called operation(s).

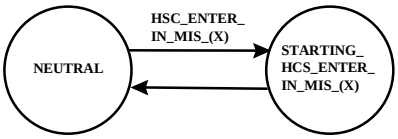


figure 5.162 head controller section : external behavior STD, communicative view

The STD consists of a neutral state. When in this state the object is ready to handle a call to its operation. When the call has been made, the object can go to the ‘starting’-state. Here the internal operation is started. The caller of the operation does not have to wait for the callee to return some result. As soon as the internal operation is started, the manager can transit back to neutral.

5.3.8.5.3 Head controller section : internal behavior-STDs

For the internal STD of the operation ‘hcs_enter_in_mis_(x)’ a reference is made to the description of this STD during the modeling of phase 1.

5.3.8.5.4 Head controller section : manager-STD

For the manager STD of the class ‘head controller section’ a reference is made to the description of this STD during the modeling of phase 1. The only difference is that the operation ‘pmm_request_approval_(x)’ is the caller in this phase 3. The names of the subprocesses and traps of this caller are the same as those used for the caller in phase 1.

5.3.8.5.5 Head controller section : employee-STDs

The manager STD has 2 employee STDs. The first employee is ‘hcs_enter_in_mis_(x)’. This employee has two subprocesses S1 and S1 and two traps T-1 and T-2 according to the caller-callee construct. The second employee is the caller ‘pmm_request_approval_(x)’. This second employee has the subprocesses S3 and S4 and traps T-3 and T-4 according to the caller-callee construct.

5.3.8.6 Account manager

5.3.8.6.1 Account manager : external behavior-STD, organizational view

The class ‘account manager’ has one operation relevant to this phase 3. This is ‘am_cu_sign_pc_(x)’. The possible starting sequence of this operation is shown in the external STD. Because there is only one operation the starting sequence is simply a succession of ‘am_cu_sign_pc_(x)’ operations.

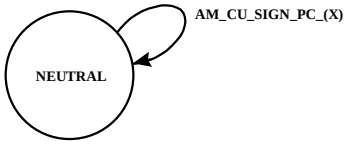


figure 5.163 account manager : external behavior STD, organizational view

5.3.8.6.2 Account manager : external behavior-STD, communicative view

The communicative view of the external behavior STD shows the communication details between the external STD and the called operation(s).

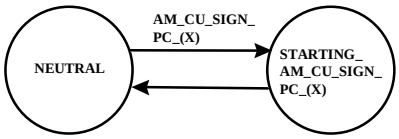


figure 5.164 head controller section : external behavior STD, communicative view

This external behavior STD has two states. In the state neutral the account manager waits until a call has been placed to its operation ‘am_cu_sign_pc_(x). If a call has been made, the manager STD can (and eventually will) transit to the state ‘starting_am_cu_sign_pc_(x)’. If the operation has been started the manager can transit back to the state neutral. The caller does not have to wait for a result from the callee, but can proceed right after the call is acknowledged by the manager.

The caller of the operation of this class can be found in the relevant import-export diagram. It is given in the ‘import_list’ attribute of the ‘uses association’.

5.3.8.6.3 Account manager : internal behavior-STDs

The account manager has 1 operation relevant for phase 3 : ‘am_cu_sign_pc_(x)’. This operation has the following internal behavior STD.

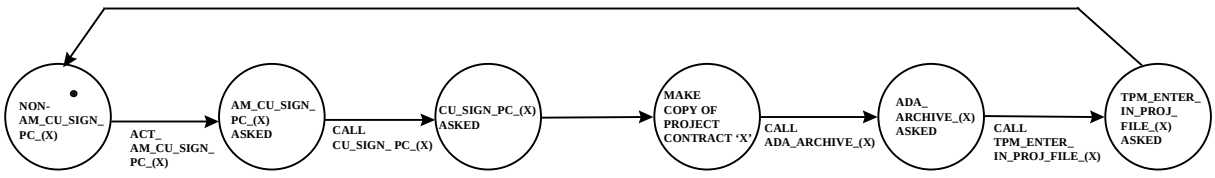


figure 5.165 int-am_cu_sign_pc_(x) : internal behavior STD

The formal parameter of the operation is the project contract (in duplicate) that the customer has to sign.

The account manager presents the two copies of the project contract (plus the appended software development plan) to the customer for his signature (call_cu_sign_pc_(x)). The customer signs both copies. He keeps one copy for himself and returns the other.
The returned (signed) copy is now duplicated. For this copying the ‘account manager’ uses the method ‘pmd_copy’ of the class ‘project management document’. This is not further detailed, but shown here as an aggregate (higher level) state.

The original is filed by ‘archive/documentation administrator’ in the central archive (call_ada_archive_(x)). The duplicate is filed by the technical project manager in the ‘project file’ of the project (call_tpm_enter_in_proj_file_(x)).

5.3.8.6.4 Account manager : manager-STD

The communication between the head controller’s operations (callees) and their callers is managed by a manager STD. The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

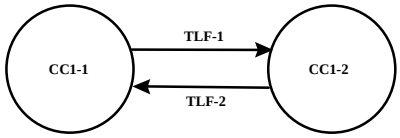


figure 5.166 account manager : manager STD

The notation CCx in the STD stands for ‘Caller(s)-Callee’ and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for ‘Traps Logical Formula’ and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

There is only one caller-callee combination, the callee ‘am_cu_sign_pc_(x)’ and its caller ‘pmm_request_approval_(x)’ of the class ‘project meeting minus’. The calling is modeled by the ‘caller does not wait’-variant of the caller-callee construct.

In the state ‘neutral’ CC and the TLF for the transition leaving the state are :

CC1-1 = {S1, S3}
TLF-1 = T-1 and T-3

In the state ‘starting_am_cu_sign_pc_(x)’ the CC and the TLF for the transition leaving the state are :

CC1-2 = {S2, S4}
TLF-2 = T-2 and T-4

5.3.8.6.5 Account manager : employee-STDs

The manager STD has the following 2 employee STDs. The first employee is ‘am_cu_sign_pc_(x)’. This employee has two subprocesses S1 and S1 and two traps T-1 and T-2 according to the caller-callee construct.

The second employee is the caller ‘pmm_request_approval_(x)’. This second employee has the subprocesses S3 and S4 and traps T-3 and T-4 according to the caller-callee construct.

5.3.8.7 Customer

5.3.8.7.1 Customer : external behavior-STD, organizational view

The class customer has one operation relevant to this phase 3. This is ‘cu_sign_pc_(x)’. The possible starting sequence of this operation is shown in the external STD. Because there is only one operation the starting sequence is simply a succession of ‘cu_sign_pc_(x)’ operations.

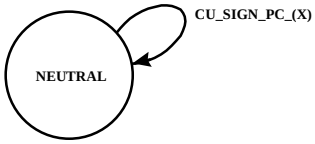


figure 5.167 customer : external behavior STD, organizational view

5.3.8.7.2 Customer : external behavior STD, communicative view

In addition to the possible starting sequence of the internal operations, the communicative view of the external behavior STD shows also communication details between external STD and the called operation(s).

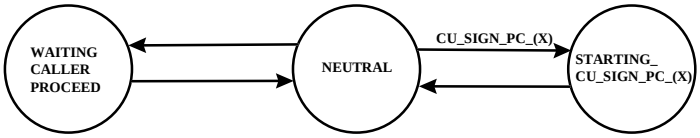


figure 5.168 customer : external behavior STD, communicative view

The STD consists of a neutral state. When in this state the object is ready to handle a call to its operation. When the call has been made, the object can go to the ‘starting’-state. Here the internal operation is started. The caller of the operation has to wait for the callee to return some result (in this case one signed copy of the project contract). When the STD notices that the callee indeed has returned its result it can transit to the state ‘waiting caller proceed’. In this state the caller is allowed to proceed with its execution.

5.3.8.7.3 Customer : internal behavior-STDs

The operation ‘cu_sign_pc_(x)’ has the following internal behavior STD.

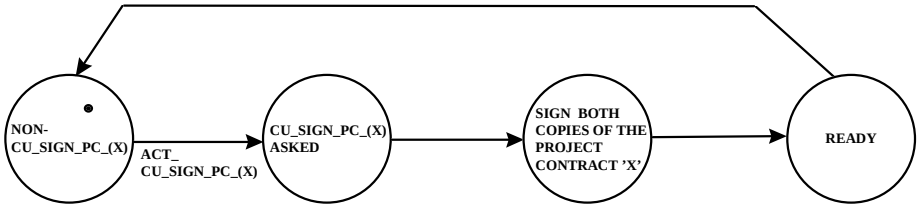


figure 5.169 int-cu_sign_pc_(x) : internal behavior STD

With this operation the customer signs both copies of the project contract. For the actual signing the cutomer uses the operation ‘pmd_sign’ of the class ‘project management document’. This is not modeled in detail, but shown here as an aggregate (high level) state.

The parameter ‘x’ contains the object id-s of both copies the project contract that have to be signed. Because the customer can sign project contracts of more then one project in parallel, the multiplicity of concurrent executing STDs is zero or more.

5.3.8.7.4 Customer : manager-STD

The manager STD prescribes in its states the subprocesses for its employees. The transitions of the manager STD are labeled with a combination of traps. The entering of the trap(s) by the relevant employee(s) is a condition for the transition. I.e. the transition can not take place if the relevant employee(s) has/have not entered the trap(s). If the relevant employee(s) has/have entered the trap(s), the transition can and will (eventually) take place.

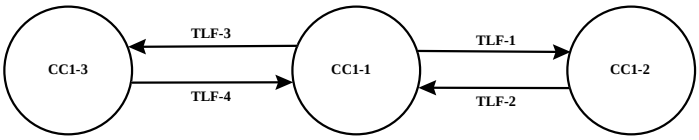


figure 5.170 customer : manager STD

There is only one caller-callee combination, the callee ‘cu_sign_pc_(x)’ and its caller ‘am_cu_sign_pc_(x)’ of the class ‘account manager’. Because the caller has to wait for the result produced by the callee, the call is modeled conform the second ‘caller waits’-variant of the caller-callee construct.

The manager keeps track of which callee is started for which caller. This infomation is called the ‘caller-callee’ relation. In the second variant of the caller waits-construct the manager can not decide by the trap-information alone if it has to make the transition TLF-1 or TLF-3 (the trap-information is the same for both transitions). The manager uses the caller-callee relation to distinguish between both transitions.

In the state ‘neutral’ the CC and the TLF for the transition leaving the state are :

CC1-1 = {S1, S3}
TLF-1 = T-1 and (T-3 and not(caller-callee relation))
TLF-3 = T-1 and (T-3 and (caller-callee relation))

In the state ‘starting_cu_sign_pc_(x)’ the CC and the TLF for the transition leaving the state are :

CC1-2 = {S2, S3} (caller has to wait)
TLF-2 = T-2

In the state ‘waiting_caller_proceed’ the CC and the TLF for the transition leaving the state are :

CC1-3 = {S1, S4} (caller may proceed)
TLF-4 = T-4

5.3.8.7.5 Customer : employee-STDs

The manager STD has the following 2 employee STDs. The first employee is ‘cu_sign_pc_(x)’. This employee has two subprocesses S1 and S1 and two traps T-1 and T-2 according to the caller-callee construct.

The second employee is the caller ‘am_cu_sign_pc_(x)’. This second employee has the subprocesses S3 and S4 and traps T-3 and T-4 according to the caller-callee construct.

5.3.8.8 Archive/documentation administrator

5.3.8.8.1 Archive/documentation administrator : external behavior-STD, organizational view

The class ‘account manager adviser’ has one operation relevant to this phase 3. This is ‘ada_archive_(x)’. The possible starting sequence of this operation is shown in the external STD. Because there is only one operation the starting sequence is simply a succession of ‘ada_archive_(x)’ operations.

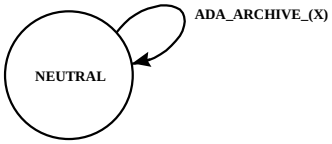


figure 5.171 archive/documentation administrator : external behavior STD, organizational view

5.3.8.8.2 Archive/documentation administrator : external behavior-STD, communicative view

The communicative view of the external behavior STD shows the communication details between the external STD and the called operation(s).

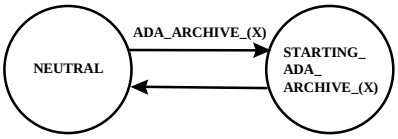


figure 5.172 archive/documentation administrator : external behavior STD, communicative view

This external behavior STD has two states. In the state neutral the administrator waits until a call has been placed to its operation. If a call has been made, the manager STD can (and eventually will) transit to the starting state. If the operation has been started the manager can transit back to the state neutral. The caller does not have to wait for a result from the callee, but can proceed right after the call is acknowledged by the manager.

The caller of the operation of this class can be found in the relevant import-export diagram. It is given in the ‘import_list’ attribute of the ‘uses association’.

5.3.8.8.3 Archive/documentation administrator : internal behavior-STDs

The archive/documentation administrator has 1 operation relevant for phase 3 : ‘ada_archive_(x)’. This operation has the following internal behavior STD.

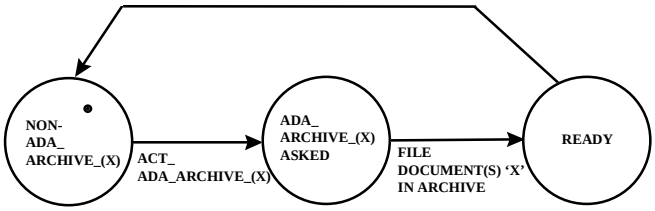


figure 5.173 int-ada_archive_(x) : internal behavior STD

With this operation the archive/documentation administrator files the document(s) represented by the parameter ‘x’ in the archive of the Waco Business Unit (WBU). The administator can file more documents in parallel. So the multiplicity of concurrent STDs is zero or more.

5.3.8.8.4 Archive/documentation administrator : manager-STD

The states and transitions of the manager STD corresponds with the states and transitions of the external STD, communicative view.

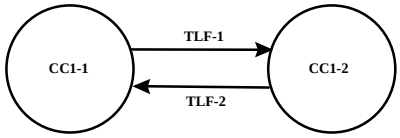


figure 5.174 archive/documentation administrator : manager STD

There is only one caller-callee combination, the callee ‘ada_archive_(x)’ and its caller ‘am_cu_sign_pc_(x)’ of the class ‘account manager’. The calling is modeled by the ‘caller does not wait’-variant of the caller-callee construct.

In the state ‘neutral’ CC (caller-callee combination) and the TLF (transition logical formula) for the transition leaving the state are :

CC1-1 = {S1, S3}
TLF-1 = T-1 and T-3

In the state ‘starting_ada_archive_(x)’ the CC and the TLF for the transition leaving the state are :

CC1-2 = {S2, S4}
TLF-2 = T-2 and T-4

5.3.8.8.5 Archive/documentation administrator : employee-STDs

The manager STD has the following 2 employee STDs. The first employee is ‘ada_archive_(x)’. This employee has two subprocesses S1 and S1 and two traps T-1 and T-2 according to the caller-callee construct.

The second employee is the caller ‘am_cu_sign_pc_(x)’. This second employee has the subprocesses S3 and S4 and traps T-3 and T-4 according to the caller-callee construct.

5.3.8.9 Technical_Project_Manager

5.3.8.9.1 Technical_Project_Manager : external behavior-STD, organizational view

The class ‘technical project manager’ has one operation relevant to this phase 3. This is ‘tpm_enter_in_proj_file_(x)’. The possible starting sequence of this operation is shown in the external STD. Because there is only one operation the starting sequence is simply a succession of ‘tpm_enter_in_proj_file_(x)’ operations.

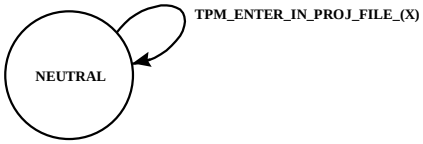


figure 5.175 technical project manager : external behavior STD, organizational view

5.3.8.9.2 Technical_Project_Manager : external behavior-STD, communicative view

The communicative view of the external behavior STD shows the communication details between the external STD and the called operation(s).

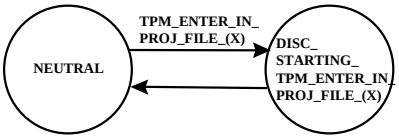


figure 5.176 technical project manager : external behavior STD, communicative view

This external behavior STD has two states. In the state neutral the technical project manager waits until a call has been placed to its operation. If a call has been made, the manager STD can (and eventually will) transit to the starting state. The starting state is a so-called ‘discriminator’-state. Here the manager decides which caller has made the call. The two possible callers are ‘pmm_request_approval_(x)’ and ‘am_cu_sign_pc_(x)’. After the operation has been started the manager can transit back to the state neutral. Neither of the two possible callers have to wait for a result from the callee, but can proceed right after the call is acknowledged by the manager.

5.3.8.9.3 Technical_Project_Manager : internal behavior-STDs

The technical project manager has 1 operation relevant for phase 3 : ‘tpm_enter_in_proj_file_(x)’. This operation has the following internal behavior STD.

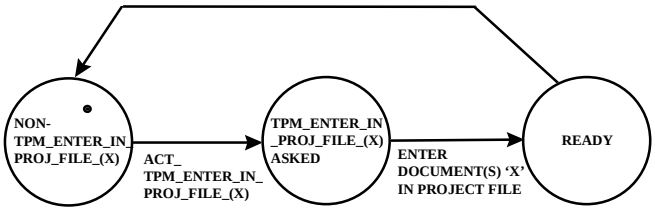


figure 5.177 int-tpm_enter_in_proj_file_(x) : internal behavior STD

With this operation the technical project manager (tpm) enters the document(s) represented by the parameter ‘x’ in the appropriate project file. The tpm can file more documents at the same time. So the multiplicity of concurrent STDs is zero or more.

5.3.8.9.4 Technical_Project_Manager : manager-STD

The states and transitions of the manager STD corresponds with the states and transitions of the external STD, communicative view.

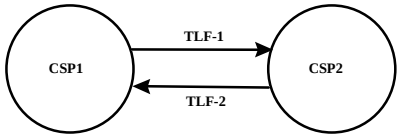


figure 5.178 technical project manager : manager STD

There is only one caller-callee combination : the callee ‘tpm_enter_in_proj_file_(x)’ and its two callers ‘pmm_request_ approval_(x)’ and ‘am_cu_sign_pc_(x)’. The calling is modeled by the ‘caller does not wait’-variant of the caller-callee construct.

The notation CCx in the STD stands for ‘Caller(s)-Callee’ and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for ‘Traps Logical Formula’ and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

The ‘or’-conditions in the CCs and TLFs are caused by the discriminator-state. The transition to the discriminator state takes place when either T-3 or T-5 has been entered. In the state is decided which caller has placed the call. I.e. it is decided if T-3 or T-5 was entered. Accordingly either {S2, S4, S5} or {S2, S3, S6} is prescribed in the discriminator state.

In the state ‘neutral’ CC and the TLF for the transition leaving the state are :

CC1-1 = {S1, S3, S5}
TLF-1 = T-1 and (T-3 or T-5)

In the state ‘disc_starting_tpm_enter_in_proj_file_(x)’ the CC and the TLF for the transition leaving the state are :

CC1-2 = {S2, S4, S5} or
{S2, S3, S6}
TLF-2 = T-2 and (T-4 or T-6)

5.3.8.9.5 Technical_Project_Manager : employee-STDs

The manager STD has 3 employee STDs. The first employee is ‘tpm_enter_in_proj_file_(x)’. This employee has two subprocesses S1 and S1 and two traps T-1 and T-2 according to the caller-callee construct.

The second employee is the caller ‘pmm_request_ approval_(x)’. This second employee has the subprocesses S3 and S4 and traps T-3 and T-4 according to the caller-callee construct.

The third employee is the caller ‘am_cu_sign_pc_(x)’. This third employee has the subprocesses S5 and S6 and traps T-5 and T-6 according to the caller-callee construct.

SOFTWARE PROCESS MODELING in SOCCA

student : H.G.Brugman
studentnr : 9105506
version : 0.10

5. Key Process Area ‘Software Project Planning’

figure 5.1figure 5.2figure 5.3figure 5.4figure 5.5figure 5.6figure 5.7figure 5.8figure 5.9figure 5.10
figure 5.11figure 5.12figure 5.13figure 5.14figure 5.15figure 5.16figure 5.17figure 5.18figure 5.19figure 5.20
figure 5.21figure 5.22figure 5.23figure 5.24figure 5.25figure 5.26figure 5.27figure 5.28figure 5.29figure 5.30
figure 5.31figure 5.32figure 5.33figure 5.34figure 5.35figure 5.36figure 5.37figure 5.38figure 5.39figure 5.40
figure 5.41figure 5.42figure 5.43figure 5.44figure 5.45figure 5.46figure 5.47figure 5.48figure 5.49figure 5.50
figure 5.51figure 5.52figure 5.53figure 5.54figure 5.55figure 5.56figure 5.57figure 5.58figure 5.59figure 5.60
figure 5.61figure 5.62figure 5.63figure 5.64figure 5.65figure 5.66figure 5.67figure 5.68figure 5.69figure 5.70
figure 5.71figure 5.72figure 5.73figure 5.74figure 5.75figure 5.76figure 5.77figure 5.78figure 5.79figure 5.80
figure 5.81figure 5.82figure 5.83figure 5.84figure 5.85figure 5.86figure 5.87figure 5.88figure 5.89figure 5.90
figure 5.91figure 5.92figure 5.93figure 5.94figure 5.95figure 5.96figure 5.97figure 5.98figure 5.99figure 5.100
figure 5.101figure 5.102figure 5.103figure 5.104figure 5.105figure 5.106figure 5.107figure 5.108figure 5.109figure 5.110
figure 5.111figure 5.112figure 5.113figure 5.114figure 5.115figure 5.116figure 5.117figure 5.118figure 5.119figure 5.120
figure 5.121figure 5.122figure 5.123figure 5.124figure 5.125figure 5.126figure 5.127figure 5.128figure 5.129figure 5.130
figure 5.131figure 5.132figure 5.133figure 5.134figure 5.135figure 5.136figure 5.137figure 5.138figure 5.139figure 5.140
figure 5.141figure 5.142figure 5.143figure 5.144figure 5.145figure 5.146figure 5.147figure 5.148figure 5.149figure 5.150
figure 5.151figure 5.152figure 5.153figure 5.154figure 5.155figure 5.156figure 5.157figure 5.158figure 5.159figure 5.160
figure 5.161figure 5.162figure 5.163figure 5.164figure 5.165figure 5.166figure 5.167figure 5.168figure 5.169figure 5.170
figure 5.171figure 5.172figure 5.173figure 5.174figure 5.175figure 5.176figure 5.177figure 5.178

5.3.9 State Transition Diagrams - Phase 4

Phase 4, ‘resource allocation’, of the process fragment ‘writing project management documents’ (partly) models the behavior of the following classes :

- head personnel section
- head computer support section
- terms of reference document
- project form
- internal memorandum
- technical project manager
- head production section
- engineer
- head controller section

5.3.9.1 Head personnel section

5.3.9.1.1 Head personnel section : external behavior-STD, organizational view

The class ‘head personnel section’ has one operation relevant for this phase 4. This is the autonomous operation ‘hps_allocate_resource’. This operation is not called by any outside caller, but is autonomously started by an object of the class. The possible starting sequence of this operation is shown in the external STD. Because there is only one operation the starting sequence is simply a succession of ‘hps_allocate_resource’ operations.

figure 5.179 head personnel section : external behavior STD, organizational view

5.3.9.1.2 Head personnel section : external behavior STD, communicative view

In addition to the possible starting sequence of the internal operations, the communicative view of the external behavior STD shows also communication details between external STD and the called operation(s).

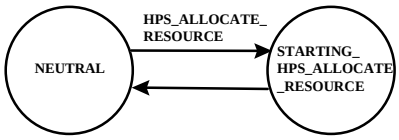


figure 5.180 head personnel section : external behavior STD, communicative view

The head of the personnel section exihibits autonomous behavior by starting the operation without it being called from another object. It can do this in its state ‘neutral’. When the object decides to execute the internal operation, it transits to the ‘starting’-state. Here the internal operation is started. The object then transits back to the neutral state. It is then ready to again start (another instance of) the internal operation.

5.3.9.1.3 Head personnel section : internal behavior-STDs

The operation ‘hps_allocate_resource’ has the following internal behavior STD.

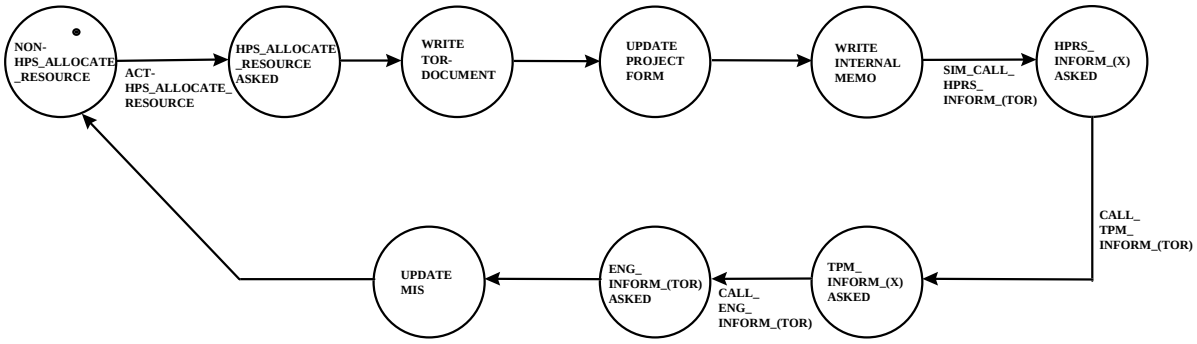


figure 5.181 int-hps_allocate_resource : internal behavior STD

First the ‘head of the personnel section’ (hps) writes a ‘terms of reference’-document (tor-document). For the actual writing the hps uses the operation ‘tor_write_(x)’ of the class ‘terms of reference’-document. This is not modeled in detail, but shown here as an aggregate (high level) state.

Then the hps updates the project form with the information on allocated personnel. For the actual updating the hps uses the operation ‘pf_update_(x)’ of the class ‘project form’. This is not modeled in detail, but shown here as an aggregate (high level) state.

Then the hps writes an memorandum as a covering letter for the tor-document. For the actual writing the hps uses the operation ‘im_write_(x)’ of the class ‘internal memorandum’. This is not modeled in detail, but shown here as an aggregate (high level) state.

Next the hps informs the technical project manager (call_tpm_inform_(tor)), his head production section, the engineer involved (call_eng_inform_(tor)), and his head production section, of the allocation of the engineer to the project. Both heads production section are informed by a simultaneous call (sim_call_hprs_inform(tor)). If the engineer and the technical project manager have the same section head, the simultaneous call becomes a single call.

Next the updated project form is given to the head of the controller section (hcs). The controller section will enter the data on the form in the Management Information System (MIS) of the Waco Business Unit (WBU). The hps uses the operation ‘hcs_enter_in_mis_(x)’ of the class ‘head controller section’ to present the updated project form to the hcs. This is not modeled in detail, but shown here as an aggregate (high level) state.

5.3.9.1.4 Head personnel section : manager-STD

The states and transitions of the manager STD corresponds with the states and transitions of the external STD, communicative view.

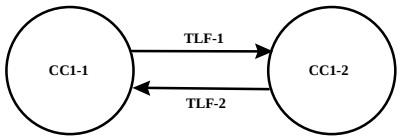


figure 5.182 head personnel section : manager STD

The notation CCx in the STD stands for ‘Caller(s)-Callee’ and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for ‘Traps Logical Formula’ and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

The caller-callee combination for ‘hps_allocate_resource’ consist only of the prescribed subprocesses of the callee ‘hps_ allocate_resource’. This is because ‘hps_allocate_resource’ is an autonomous operation.

In the state ‘neutral’ the CC and the TLF for the transition leaving the state are :

CC1-1 = {S1}
TLF-1 = T-1 (corresponds with ‘hps_allocate_resource’ transition in extern STD)

In the state ‘starting_hps_allocate_resource’ the CC and the TLF for the transition leaving the state are :

CC1-2 = {S2}
TLF-2 = T-2

5.3.9.1.5 Head personnel section : employee-STDs

The manager STD has 1 employee STD. This is the autonomous operation ‘hps_allocate_resource’. This employee has two subprocesses S1 and S2 and two traps T-1 and T-2 according to the caller-callee construct.

5.3.9.2 Head computer support section

5.3.9.2.1 Head computer support section : external behavior-STD, organizational view

The class ‘head computer support section’ has one operation relevant for this phase 4. This is the autonomous operation ‘hcss_allocate_resource’. This operation is not called by any outside caller, but autonomously started by an object of the class. The possible starting sequence of this operation is shown in the external STD. Because there is only one operation the starting sequence is simply a succession of ‘hcss_allocate_resource’ operations.

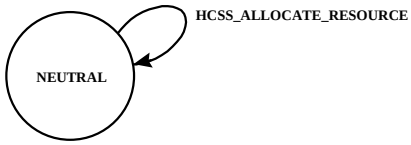


figure 5.183 head computer support section : external behavior STD, organizational view

5.3.9.2.2 Head computer support section : external behavior STD, communicative view

In addition to the possible starting sequence of the internal operations, the communicative view of the external behavior STD shows also communication details between external STD and the called operation(s).

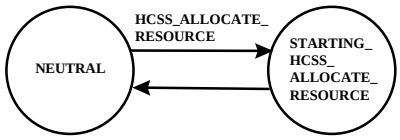


figure 5.184 head computer support section : external behavior STD, communicative view

The head of the computer support section exihibits autonomous behavior by starting the operation without it being called from another object. It can do this in its state ‘neutral’. When the object decides to execute the internal operation, it transits to the ‘starting’-state. Here the internal operation is started. The object then transits back to the neutral state. It is then ready to again start (another instance of) the internal operation.

5.3.9.2.3 Head computer support section : internal behavior-STDs

The operation ‘hcss_allocate_resource’ has the following internal behavior STD.

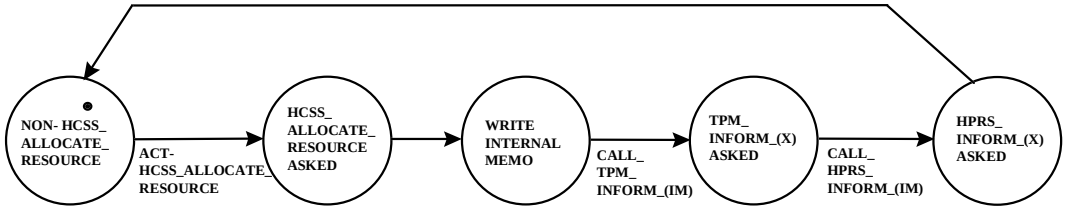


figure 5.185 int-hcss_allocate_resource : internal behavior STD

First the ‘head of the computer support section’ (hcss) writes an internal memorandum, describing the allocation of computer section-personnel to a project. For the actual writing the hcss uses the operation ‘im_write_(x)’ of the class ‘internal memorandum’. This is not modeled in detail, but shown here as an aggregate (high level) state.

Next the hcss informs the technical project manager (call_tpm_inform_(im)) and his head production section (call_hprs_inform(im)) of this allocation by communicating the internal memo to them.

5.3.9.2.4 Head computer support section : manager-STD

The states and transitions of the manager STD corresponds with the states and transitions of the external STD, communicative view.

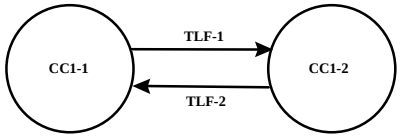


figure 5.186 head computer section section : manager STD

The caller-callee combination for ‘hcss_allocate_resource’ consist only of the prescribed subprocesses of the callee ‘hcss_allocate_resource’. This is because ‘hcss_allocate_resource’ is an autonomous operation.

In the state ‘neutral’ the CC (caller-callee combination) and the TLF (traps logical formula) for the transition leaving the state are :

CC1-1 = {S1}
TLF-1 = T-1 (corresponds with ‘hcss_allocate_resource’ transition in extern STD)

In the state ‘starting_hcss_allocate_resource’ the CC and the TLF for the transition leaving the state are :

CC1-2 = {S2}
TLF-2 = T-2

5.3.9.2.5 Head computer support section : employee-STDs

The manager STD has 1 employee STD. This is the autonomous operation ‘hcss_allocate_resource’. This employee has two subprocesses S1 and S2 and two traps T-1 and T-2 according to the caller-callee construct.

5.3.9.3 Terms of Reference-document

5.3.9.3.1 Terms of Reference-document : external behavior-STD, organizational view

The class ‘terms of reference-document’ has one operation relevant for this phase 4. This is the operation ‘tor_write_(x)’. The possible starting sequence of this operation is shown in the external STD. Because there is only one operation the starting sequence is simply a succession of ‘tor_write_(x)’ operations.

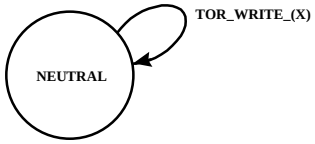


figure 5.187 terms of reference-document : external behavior STD, organizational view

5.3.9.3.2 Terms of Reference-document : external behavior STD, communicative view

Because the calling of this operation is not modeled in detail in the caller STD, hps_allocate_resource, the organizational view of the external STD is also not detailed into the communicative view.

5.3.9.3.3 Terms of Reference-document : internal behavior-STDs

The operation ‘tor_write_(x)’ has the following internal behavior STD.

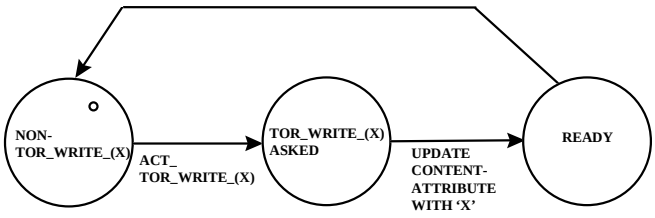


figure 5.188 int-tor_write_(x) : internal behavior STD

The operation updates the content attribute of the ‘terms of reference’-document object with the actual value of the parameter ‘x’.

The multiplicity of concurrent executing STD instances is zero or 1. This is indicated by the hollow circle inside the first state of the STD. This serializes the ‘writes’ in the tor-document. I.e. the ‘writes’ can not happen in parallel, but only sequential.

5.3.9.3.4 Terms of Reference-document : manager-STD

Because the calling of this operation is not modeled in detail in the caller STD, hps_allocate_resource, the manager STD controlling this caller-callee combination is also not modeled.

5.3.9.3.5 Terms of Reference-document : employee-STDs

Because the calling of this operation is not modeled in detail in the caller STD, hps_allocate_resource, the employees (caller-callee combination) and the prescribed traps and subprocesses are also not modeled.

5.3.9.4 Project form

5.3.9.4.1 Project form : external behavior-STD, organizational view

The class ‘project form’ has one operation relevant for this phase 4. This is the operation ‘pf_update_(x)’. The possible starting sequence of this operation is shown in the external STD. Because there is only one operation the starting sequence is simply a succession of ‘pf_update_(x)’ operations.

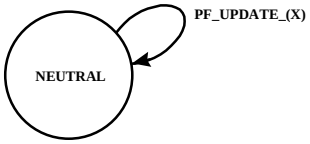


figure 5.189 project form : external behavior STD, organizational view

5.3.9.4.2 Project form : external behavior STD, communicative view

Because the calling of this operation is not modeled in detail in the caller STD, hps_allocate_resource, the organizational view of the external STD is also not detailed into the communicative view.

5.3.9.4.3 Project form : internal behavior-STDs

The operation ‘pf_update_(x)’ has the following internal behavior STD.

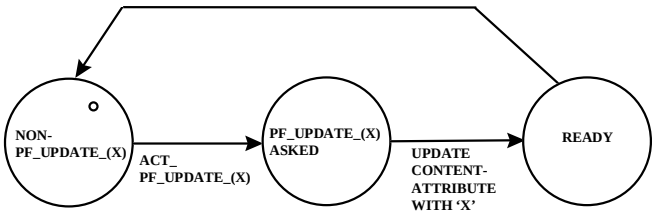


figure 5.190 int-pf_update_(x) : internal behavior STD

The operation updates the content attribute of the ‘project form’ object with the actual value of the parameter ‘x’.

The multiplicity of concurrent executing STD instances is zero or 1. This is indicated by the hollow circle inside the first state of the STD. This serializes the ‘updates’ in the project form. I.e. the ‘updates’ can not happen in parallel, but only sequential.

5.3.9.4.4 Project form : manager-STD

Because the calling of this operation is not modeled in detail in the caller STD, hps_allocate_resource, the manager STD controlling this caller-callee combination is also not modeled.

5.3.9.4.5 Project form : employee-STDs

Because the calling of this operation is not modeled in detail in the caller STD, hps_allocate_resource, the employees (caller-callee combination) and the prescribed traps and subprocesses are also not modeled.

5.3.9.5 Internal memorandum

5.3.9.5.1 Internal memorandum : external behavior-STD, organizational view

The class ‘internal memorandum’ has one operation relevant for this phase 4. This is the operation ‘im_write_(x)’. The possible starting sequence of this operation is shown in the external STD. Because there is only one operation the starting sequence is simply a succession of ‘im_write_(x)’ operations.

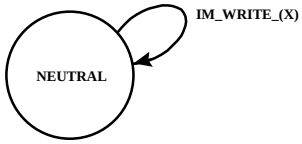


figure 5.191 internal memorandum : external behavior STD, organizational view

5.3.9.5.2 Internal memorandum : external behavior STD, communicative view

Because the calling of this operation is not modeled in detail in the STD of the callers, hps_allocate_resource and hcss_ allocate_resource, the organizational view of the external STD is also not detailed into the communicative view.

5.3.9.5.3 Internal memorandum : internal behavior-STDs

The operation ‘im_write_(x)’ has the following internal behavior STD.

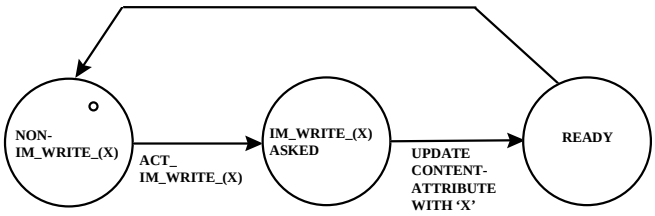


figure 5.192 int-im_write_(x) : internal behavior STD

The operation updates the content attribute of the ‘internal memorandum’ object with the actual value of the parameter ‘x’.

The multiplicity of concurrent executing STD instances is zero or 1. This is indicated by the hollow circle inside the first state of the STD. This serializes the ‘writes’ in the internal memo. I.e. the ‘writes’ can not happen in parallel, but only sequential.

5.3.9.5.4 Internal memorandum : manager-STD

Because the calling of this operation is not modeled in detail in the caller STDs, hps_allocate_resource and hcss _allocate_resource, the manager STD controlling this caller-callee combination is also not modeled. N.B. the starting state of this manager STD is a ‘discriminator’ state. In this way the STD can distinguish between the two callers.

5.3.9.5.5 Internal memorandum : employee-STDs

Because the calling of this operation is not modeled in detail in the caller STDs, the employees (caller-callee combinations) and the prescribed traps and subprocesses are also not modeled.

5.3.9.6 Technical_Project_Manager

5.3.9.6.1 Technical_Project_Manager : external behavior-STD, organizational view

The class ‘technical project manager’ has one operation relevant for this phase 4. This is ‘tpm_inform_(x)’. The possible starting sequence of this operation is shown in the external STD. Because there is only one operation the starting sequence is simply a succession of ‘tpm_inform_(x)’ operations.

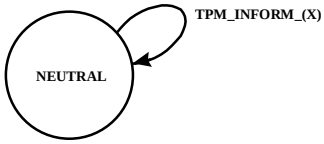


figure 5.193 technical project manager : external behavior STD, organizational view

5.3.9.6.2 Technical_Project_Manager : external behavior-STD, communicative view

The communicative view of the external behavior STD shows the communication details between the external STD and the called operation(s).

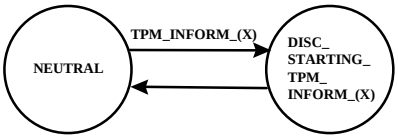


figure 5.194 technical project manager : external behavior STD, communicative view

This external behavior STD has two states. In the state neutral the technical project manager waits until a call has been placed to its operation. If a call has been made, the manager STD can (and eventually will) transit to the starting state. The starting state is a so-called ‘discriminator’-state. Here the manager decides which caller has made the call. The two possible callers are ‘hps_allocate_resource’ and ‘hcss_allocate_resource’. After the operation has been started the manager can transit back to the neutral state. Neither of the two possible callers have to wait for a result from the callee, but can proceed right after the call is acknowledged by the manager.

5.3.9.6.3 Technical_Project_Manager : internal behavior-STDs

The technical project manager has 1 operation relevant for phase 4 : ‘tpm_inform_(x)’. This operation has the following internal behavior STD.

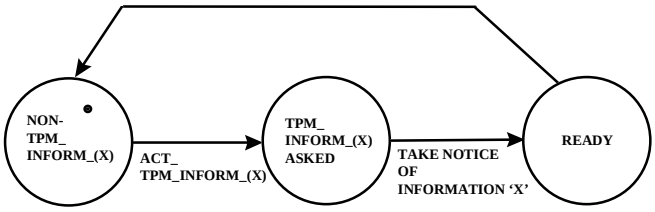


figure 5.195 int-tpm_inform_(x) : internal behavior STD

With this operation the technical project manager (tpm) is informed of the information represented by the parameter ‘x’. The tpm can absorb more pieces of information at the same time. So the multiplicity of concurrent STDs is zero or more.

5.3.9.6.4 Technical_Project_Manager : manager-STD

The states and transitions of the manager STD corresponds with the states and transitions of the external STD, communicative view.

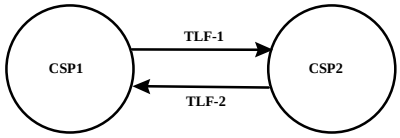


figure 5.196 technical project manager : manager STD

There is only one caller-callee combination : the callee ‘tpm_inform_(x)’ and its two callers ‘hps_allocate_resource’ and ‘hcss_allocate_resource’. The calling is modeled by the ‘caller does not wait’-variant of the caller-callee construct.

The notation CCx in the STD stands for ‘Caller(s)-Callee’ and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for ‘Traps Logical Formula’ and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

The ‘or’-conditions in the CCs and TLFs are caused by the discriminator-state. The transition to the discriminator state takes place when either T-3 or T-5 has been entered. In the state is decided which caller has placed the call. I.e. it is decided if T-3 or T-5 was entered. Accordingly either {S2, S4, S5} or {S2, S3, S6} is prescribed in the discriminator state.

In the state ‘neutral’ CC and the TLF for the transition leaving the state are :

CC1-1 = {S1, S3, S5}
TLF-1 = T-1 and (T-3 or T-5)

In the state ‘disc_starting_tpm_inform_(x)’ the CC and the TLF for the transition leaving the state are :

CC1-2 = {S2, S4, S5} or
{S2, S3, S6}
TLF-2 = T-2 and (T-4 or T-6)

5.3.9.6.5 Technical_Project_Manager : employee-STDs

The manager STD has 3 employee STDs. The first employee is ‘tpm_inform_(x)’. This employee has two subprocesses S1 and S2 and two traps T-1 and T-2 according to the caller-callee construct.

The second employee is the caller ‘hps_allocate_resource’. This second employee has the subprocesses S3 and S4 and traps T-3 and T-4 according to the caller-callee construct.

The third employee is the caller ‘hcss_allocate_resource. This third employee has the subprocesses S5 and S6 and traps T-5 and T-6 according to the caller-callee construct.

5.3.9.7 Head production section

5.3.9.7.1 Head production section : external behavior-STD, organizational view

The class ‘head production section’ has one operation relevant for this phase 4. This is ‘hprs_inform_(x)’. The possible starting sequence of this operation is shown in the external STD. Because there is only one operation the starting sequence is simply a succession of ‘hprs_inform_(x)’ operations.

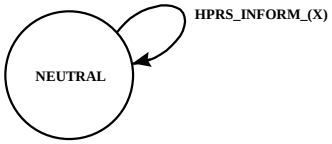


figure 5.197 head production section : external behavior STD, organizational view

5.3.9.7.2 Head production section : external behavior-STD, communicative view

The communicative view of the external behavior STD shows the communication details between the external STD and the called operation(s).

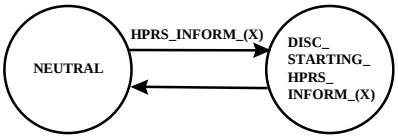


figure 5.198 head production section : external behavior STD, communicative view

This external behavior STD has two states. In the state neutral the head of the production section waits until a call has been placed to its operation. If a call has been made, the manager STD can (and eventually will) transit to the starting state. The starting state is a so-called ‘discriminator’-state. Here the manager decides which caller has made the call. The two possible callers are ‘hps_allocate_resource’ and ‘hcss_allocate_resource’. After the operation has been started the manager can transit back to the neutral state. Neither of the two possible callers have to wait for a result from the callee, but can proceed right after the call is acknowledged by the manager.

5.3.9.7.3 Head production section : internal behavior-STDs

The head production section has 1 operation relevant for phase 4 : ‘hprs_inform_(x)’. This operation has the following internal behavior STD.

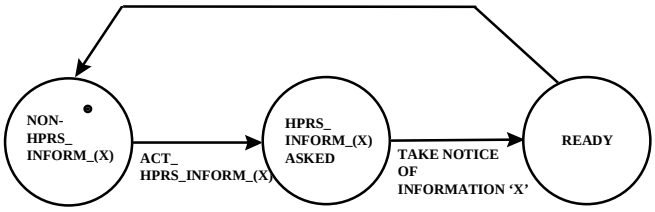


figure 5.199 int-hprs_inform_(x) : internal behavior STD

With this operation the head production section (hprs) is given the information represented by the parameter ‘x’. The hprs can absorb more pieces of information at the same time. So the multiplicity of concurrent STDs is zero or more.

5.3.9.7.4 Head production section : manager-STD

The states and transitions of the manager STD corresponds with the states and transitions of the external STD, communicative view.

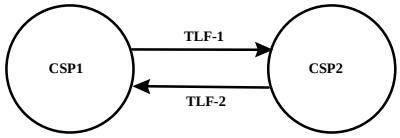


figure 5.200 head production section : manager STD

There is only one caller-callee combination : the callee ‘hprs_inform_(x)’ and its two callers ‘hps_allocate_resource’ and ‘hcss_allocate_resource’. The calling is modeled by the ‘caller does not wait’-variant of the caller-callee construct.

The notation CCx in the STD stands for ‘Caller(s)-Callee’ and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for ‘Traps Logical Formula’ and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

The ‘or’-conditions in the CCs and TLFs are caused by the discriminator-state. The transition to the discriminator state takes place when either T-3 or T-5 has been entered. In the state is decided which caller has placed the call. I.e. it is decided if T-3 or T-5 was entered. Accordingly either {S2, S4, S5} or {S2, S3, S6} is prescribed in the discriminator state.

In the state ‘neutral’ CC and the TLF for the transition leaving the state are :

CC1-1 = {S1, S3, S5}
TLF-1 = T-1 and (T-3 or T-5)

In the state ‘disc_starting_hprs_inform_(x)’ the CC and the TLF for the transition leaving the state are :

CC1-2 = {S2, S4, S5} or
{S2, S3, S6}
TLF-2 = T-2 and (T-4 or T-6)

5.3.9.7.5 Head production section : employee-STDs

The manager STD has 3 employee STDs. The first employee is ‘hprs_inform_(x)’. This employee has two subprocesses S1 and S2 and two traps T-1 and T-2 according to the caller-callee construct.

The second employee is the caller ‘hps_allocate_resource’. This second employee has the subprocesses S3 and S4 and traps T-3 and T-4 according to the caller-callee construct.

The third employee is the caller ‘hcss_allocate_resource. This third employee has the subprocesses S5 and S6 and traps T-5 and T-6 according to the caller-callee construct.

5.3.9.8 Engineer

5.3.9.8.1 Engineer : external behavior-STD, organizational view

The class ‘engineer’ has one operation relevant for this phase 4. This is ‘eng_inform_(x)’. The possible starting sequence of this operation is shown in the external STD. Because there is only one operation the starting sequence is simply a succession of ‘eng_inform_(x)’ operations.

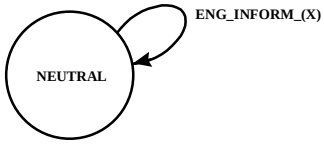


figure 5.201 engineer : external behavior STD, organizational view

5.3.9.8.2 Engineer : external behavior-STD, communicative view

The communicative view of the external behavior STD shows the communication details between the external STD and the called operation(s).

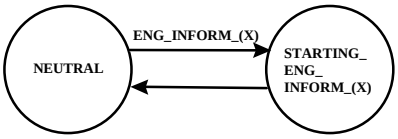


figure 5.202 engineer : external behavior STD, communicative view

This external behavior STD has two states. In the state neutral the engineer waits until a call has been placed to its operation. If a call has been made, the manager STD can (and eventually will) transit to the starting state. After the operation has been started the manager can transit back to the neutral state. The caller does not have to wait for a result from the callee, but can proceed right after the call is acknowledged by the manager.

5.3.9.8.3 Engineer : internal behavior-STDs

The engineer has 1 operation relevant for phase 4 : ‘eng_inform_(x)’. This operation has the following internal behavior STD.

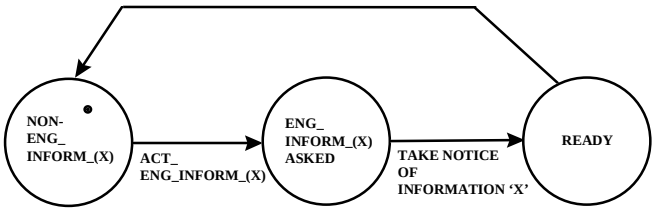


figure 5.203 int-eng_inform_(x) : internal behavior STD

With this operation the engineer (eng) is given the information represented by the parameter ‘x’. The engineer can absorb more pieces of information at the same time. So the multiplicity of concurrent STDs is zero or more.

5.3.9.8.4 Engineer : manager-STD

The states and transitions of the manager STD corresponds with the states and transitions of the external STD, communicative view.

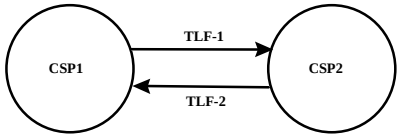


figure 5.204 engineer : manager STD

There is only one caller-callee combination : the callee ‘eng_inform_(x)’ and its caller ‘hps_allocate_resource’. The calling is modeled by the ‘caller does not wait’-variant of the caller-callee construct.

The notation CCx in the STD stands for ‘Caller(s)-Callee’ and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for ‘Traps Logical Formula’ and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

In the state ‘neutral’ CC and the TLF for the transition leaving the state are :

CC1-1 = {S1, S3}
TLF-1 = T-1 and T-3

In the state ‘starting_eng_inform_(x)’ the CC and the TLF for the transition leaving the state are :

CC1-2 = {S2, S4}
TLF-2 = T-2 and T-4

5.3.9.8.5 Engineer : employee-STDs

The manager STD has 2 employee STDs. The first employee is ‘eng_inform_(x)’. This employee has two subprocesses S1 and S2 and two traps T-1 and T-2 according to the caller-callee construct.

The second employee is the caller ‘hps_allocate_resource’. This second employee has the subprocesses S3 and S4 and traps T-3 and T-4 according to the caller-callee construct.

5.3.9.9 Head controller section

5.3.9.9.1 Head controller section : external behavior-STD, organizational view

The class ‘head controller section’ has one operation relevant for this phase 4. This is ‘hcs_enter_in_mis_(x)’. This operation is also used in phase 1. The only difference is in the caller of the operation. In phase 1 the caller is ‘hps_initiate_project_form_(x)’. Here, in phase 4, the caller is ‘hps_allocate_resource’. The possible starting sequence of this operation is shown in the external STD. Because there is only one operation the starting sequence is simply a succession of ‘hcs_enter_in_mis_(x)’ operations.

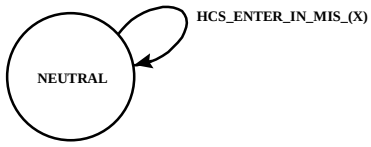


figure 5.205 head controller section : external behavior STD, organizational view

5.3.9.9.2 Head controller section : external behavior STD, communicative view

In addition to the possible starting sequence of the internal operations, the communicative view of the external behavior STD shows also communication details between external STD and the called operation(s).

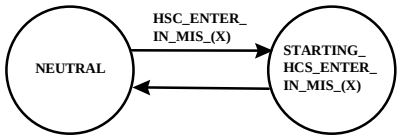


figure 5.206 head controller section : external behavior STD, communicative view

The STD consists of a neutral state. When in this state the object is ready to handle a call to its operation. When the call has been made, the object can go to the ‘starting’-state. Here the internal operation is started. The caller of the operation does not have to wait for the callee to return some result. As soon as the internal operation is started, the manager can transit back to neutral.

5.3.9.9.3 Head controller section : internal behavior-STDs

For the internal STD of the operation ‘hcs_enter_in_mis_(x)’ a reference is made to the description of this STD during the modeling of phase 1.

5.3.9.9.4 Head controller section : manager-STD

For the manager STD of the class ‘head controller section’ a reference is made to the description of this STD during the modeling of phase 1. The only difference is that the operation ‘hps_allocate_resource’ is the caller in this phase 4. The names of the subprocesses and traps of this caller are the same as those used for the caller in phase 1.

5.3.9.9.5 Head controller section : employee-STDs

The manager STD has 2 employee STDs. The first employee is ‘hcs_enter_in_mis_(x)’. This employee has two subprocesses S1 and S2 and two traps T-1 and T-2 according to the caller-callee construct. The second employee is the caller ‘hps_allocate_resource’. This second employee has the subprocesses S3 and S4 and traps T-3 and T-4 according to the caller-callee construct.

SOFTWARE PROCESS MODELING in SOCCA

student : H.G.Brugman
studentnr : 9105506
version : 0.10

6. Integration of process fragment 'writing project management documents'

6.1 Introduction

To look into the question of the 'scaleability' of SOCCA (i.e can a larger SOCCA model be constructed from separate smaller sub-models), the modeling of the process fragment 'writing project management documents' is done by splitting the process fragment into four smaller process fragments (phases). These four phases are modeled indepently of each other. The four phases are : 'phase 1, recognizing customer requirements', 'phase 2, writing and consultation', 'phase 3, approval' and 'phase 4, resource allocation'. The sub-models of these four phases are described in chapter 5.

In this chapter 6 the integration of the four sub-models of 'phase 1', 'phase 2', 'phase 3' and 'phase 4' into one SOCCA model of the process fragment 'writing project management documents' is performed.

In paragraph 6.2 the general principles involved in the integration will be discussed, as they apply to the integration of the process fragment 'writing project management documents'.

The SOCCA model of the integrated process fragment 'writing project management documents' is described in paragraph 6.3. Paragraph 6.3.1 describes the 'class diagrams' of the model, as they apply to the integration. Paragraph 6.3.2 describes the 'state transition diagrams' of the model, as they apply to the integration.

Paragraph 6.3 also contains an explanation of how the integrated process fragment 'writing project management documents' executes. Also is explained in paragraph 6.3 how multiple instances of the process fragment can execute concurrently.

6.2 General principles

In this paragraph the general principles involved in the integration will be discussed, as they apply to the integration of the process fragment 'writing project management documents'.

The intention of the integration process is not to influence the models of the four phases. That is to say to make the modeling of the constituent smaller process fragments independent of their integration. This allows for a separate modeling of the sub-models (by separate engineers) in a big project. It also facilitates the update of the total model if there are any changes required during the life time of the model. This is according to the accepted software engineering principle of low coupling between software modules. Still some changes to the sub-models may be necessary. The possible reasons for these changes are described in the next subparagraph.

The integration of SOCCA behavior sub-models into one bigger model, takes place in 4 steps (analogous to the 4 steps that are used in database view (schema) integration [ELM]) :

- identifying correspondences and conflicts between the sub-models
- modifying the sub-models to conform with each other (resolving the conflicts)
- merging the sub-models. This includes the modeling of the sequential dependencies between the sub-models.
- restructuring of the resulting model (optimizing)

<u>conflicts between sub-models</u>

Two kinds of conflicts are possible. These are 'type'-conflicts and 'name'-conflicts.

a. type conflict

With this type of conflict some concept is modeled in one sub-model by a class. This results in behavior STDs for this class. In another sub-model this same concept may be modeled by an attribute (of some other class). When this is the case, one of the sub-models has to be re-modeled before any integration can take place.

In the sub-models of the four phases of the process fragment 'writing project managment documents' no 'type'-conflicts exist.

b. name conflict

When different sub-models use different names for two identical internal STDs (synonyms), one of these internal STD's must be renamed. When different sub-models use the same name for two different internal STDs (homonyms), one of these internal STDs must be renamed. Subprocess names (e.g. S1) and trap names (e.g. T-1) used in the sub-models are abbreviations. The full name includes a qualifier as to the manager STD which prescribes them (as well as the class name and operation-name in which the subprocesses and traps appear using the 'dot'-notation). E.g. the full name of subprocess S3 of the operation 'cu_request_proposal' of the class 'customer' as modeled in phase 1, paragraph 'requirements document : employee STDs', is 'requirements_document.cu_request_proposal.S3_with_respect_ to_requirements_document'. So these names are actually unique and don't require renaming. The sub-model uses only the abbreviated names because the manager STD can be readily deduced.

In the sub-models of the four phases of the process fragment 'writing project management documents' no naming conflicts exist.

modifying sub-models

The modification necessary for the resolving of the type-conflicts and the naming conflicts is already discussed in the above subparagraph.

There may be yet another modification necessary. This is when two sub-models model an identical functionality with a (slightly) different internal STD. Here one (or maybe two) of the STDs has to be remodeled so that they become identical. Care has to be taken as that this remodeled STD does not to affect the remaining part of sub-model.

In the sub-models of the four phases of the process fragment 'writing project management documents' this type of modification is not necessary.

However the behavior of the class customer in phase 1 has to be changed. In the original sub-model it was assumed that the customer could start phase 1 (and therefore the total process fragment 'writing project management documents') repetitively by sequential calls to its autonomous operation 'cu_request_proposal'. In this way more than one request for proposal could be handled in parallel. In the integration this parallel functionality is handled in another way, by parallelizing a new operation instead of 'cu_request_proposal'. The customer now calls its autonomous operation 'cu_project_life_cycle' repetitively to start parallel executions of the process fragment. During each separate execution the customer can only once call its operation 'cu_request_proposal'.

The changes in the class 'customer' will be shown separately in this paragraph instead of directly in the sub-model. This is to clearly document the work required during the integration.

merging the sub-models

- internal STDs

All the internal STDs of the sub-models are part of the integrated model. This includes the prescribed subprocesses and traps of these internal STDs. In case of identical internal STDs exactly one will be part of the total model. On the other hand, all different/various sets of subprocesses and traps of the identical STDs together belong to the integrated model.

- external STDs

For each class in the process fragment a total external STD will be constructed. The construction algorithm is as follows

- extend each phase-external STD of the class with a start state end a final state. The start state has a transition leaving it and entering the phase-external STD. The final state has a transition coming into it from the phase-external STD.

- connect the so extended phase-external STD with each other in such a way that the final state of one phase-external STD coincides with the start state of the next phase-external STD. This 'coinciding' state between two phase-external STDs is called an 'intermediate' state. This models the sequential dependency between the phases.

- the transitions leaving the start states get the label 'change_to_phase_x'. These represent the 'phase-changing'-operations. The transitions entering the final states get the label 'phase_ended. These represent the 'phase-ended' operation. When a 'phase_ended'-operation is called the total external STD can and will make the transition to the next intermediate state. When a 'phase_changing'-operation is called the total external STD can and will make the transition to the next phase-external STD. In this way the total external STD to transit from one phase-external STD to the next via an

intermediate state. The intermediate state in conjunction with the 'phase_ended'-operation is important in the model to prevent that the classes 'race' through their total external STD without the phases being properly terminated.

If a class does not participates in some phase it will not have an external STD for that phase. In this case a dummy state is inserted in the total external STD. This is done for reasons of symmetry. It will make the integration process easier to perform.

The 'phase-changing'-operations of a total external STD are called by a special kind of object. This is a so-called control object.

- control object

A control object models the (higher level) control flow of a process (fragment). It does this by manipulating the external STDs of the classes involved in the constituent (smaller) process fragments. For this purpose the control object has an internal operation (control operation) which calls the 'phase-changing'-operations of a (total) external STD in sequence.

This concept of differents parts of an external STD being active at different times models the foreseen 'evolution' of the process (fragment) [FIN]. The behavior of the process (fragment) evolves over time. It is a predictable evolution in that the future behavior is known beforehand.

So the internal STD of the control operation of the control object manages (models) the control flow of the total process fragment. It 'switches' the total external STDs from phase 1 to phase 2 to phase 3 to phase 4.

There are 19 classes participating in the process fragment 'writing project management documents'. The switching from phase to phase is done for all 19 total external STDs at the same time. To do so the control operation places a 'simultaneous call (sim_call)' to the 'phase changing'-operations of all 19 classes.

As control object is chosen an object of the class 'project'. This is a natural choice because all the work in the process fragment is related to a certain project. It is also in compliance with the definition in [FIN] of a project : 'a project is an instantiation of a process to produce a specific product in a given organization, with specific objectives and constraints.'

6.3 SOCCA model (integration)

This paragraph starts with an explanation of how the integrated process fragment 'writing project management documents' executes. Then is explained how multiple instances of the process fragment can execute concurrently.

Then follows the description of the SOCCA model of the integrated process fragment 'writing project management documents'. Paragraph 6.3.1 describes the 'class diagrams' of the model, as they apply to the integration. Paragraph 6.3.2 describes the 'state transition diagrams' of the model, as they apply to the integration.

The next table shows which classes are taking part in which phase of the process fragment.

class	phase 1	phase 2	phase 3	phase 4
customer	yes	yes	yes	-
requirements document	yes	-	-	-
account manager	yes	yes	yes	-
make or buy meeting	yes	-	-	-
chief executive officer	yes	-	yes	-
head personnel section	yes	-	-	yes
project form	yes	-	-	yes
head controller section	yes	-	yes	yes
technical project manager	-	yes	yes	yes
quality assurance adviser	-	yes	yes	-
head production section	-	yes	-	yes
head support section	-	yes	yes	-
project management document	-	yes	-	-
project meeting minus	-	-	yes	-
archive/documentation administrator	-	-	yes	-

head computer support section	-	-	-	yes
terms of reference document	-	-	-	yes
internal memorandum	-	-	-	yes
engineer	-	-	-	yes

table 6.1 participating classes versus phases

Executing process fragment

The process fragment is ‘started’ by the customer. The started and executing process fragment is called the ‘enacting’ model in the terminology of the Software Process Modeling-community [FIN]. To start the process fragment the customer invokes its own autonomous operation ‘cu_project_life_cycle’. In this operation the operation ‘pr_ project_life_cycle’ of a certain project object is called. The operation ‘pr_project_life_cycle’ then switches all the (19) total external STDs from their start state to their phase 1-state. It does this by the calling the operation ‘change_to_phase_1’ of all (19) participating classes. It performs all the (19) calls simultaneously by placing a sim_call. All (19) external STD will (eventually) enter their phase 1-state. So phase 1 of the process fragment ‘writing project management documents’ can now take place.

Although the ‘phase-changing’-operation of all (19) classes are called simultenuously, there is no guarantee that the total external STDs are all in their phase 1-state at the same time. Some of them may already be active in their phase 1 while others still are ‘on their way’ to their state ‘phase 1’. The situation can arise where some internal STD of a class participating in phase 1 places a call to another participating class of which the external STD is not yet in its ‘phase 1’-state. In this case the calling STD just waits until the external STD in question has arrived in its phase 1-state and can accept its call. This is the same situation as when a ‘normal’ external STD is not (yet) in a state where it can accept some call.

At the end of phase 1 the last operation of the phase 1 calls (as the last action it performs) the operation ‘pr_phase_ended’ of the control object ‘project’. In this way it signals to the control object that the phase has ended. The control object then sim_calls all the participating classes to exit from their current phase-external STD. The last operation of phase 1 is the operation ‘mb_request_decision_(x)’ of the class ‘make or buy meeting’.

The reason for the design decision to let the last operation call the control object instead of the last operation (sim_)calling the ‘phase_ended’/ ‘phase_changing’ operations of the other classes directly is twofold. Firstly, in this way the control flow is clearly visible in the model of the control operation of the control object. More importantly, the responsiblity for the flow in the process fragment now lies at a level above the internal operations of the participating classes. This is were it should lie. Secondly, it provides for a lower ‘coupling’ between the modeled internal operations of the process fragment. The last operation does not have to know all the other operations in the process fragment. Nor does it have to know which phase comes next in the process fragment. It only has to deal with the control object of the process fragment.

At the’ sim_call_phase_ended’ from the control operation ‘pr_phase_ended’, all the (19) classes will (eventually) leave their current phase_external STD and enter the next intermediate state.

Now the ’sim_call_change_to_phase_2’ of the control operation ‘pr_project_life_cyle’ (which is still executing and may or may not have placed this call already) can be handled by the classes of the process fragment. They are now in the correct intermediate state to handle the call. All participating classes will (eventually) transit from the intermediate state to the next state, phase 2. The process fragment comes in its next phase. Phase 2 of the process fragment ‘writing project management documents’ can now take place.

At the end of phase 2 the last operation of the phase 2 calls (as the last action it performs) the operation ‘pr_phase_ended’ of the control object ‘project’. In this way it signals to the control object that the phase has ended. The control object then ‘sim_calls’ all the participating classes to exit from their current phase-external STD. The last operation of phase 2 is the operation ‘hprs_pmm_ request_approval_(x)’ of the class ‘head production section’.

At the’ sim_call_phase_ended’ from the control operation ‘pr_phase_ended’, all the (19) classes will (eventually) leave their current phase_external STD and enter the next intermediate state.

Now the ’sim_call_change_to_phase_3’ of the control operation ‘pr_project_life_cyle’ (which is still executing and may or may not have placed this call already) can be handled by the classes of the process fragment. They are now in the correct intermediate state to handle the call. All participating classes will (eventually) transit from the intermediate state to the next state, phase 3. The process fragment comes in its next phase. Phase 3 of the process fragment ‘writing project management documents’ can now take place.

At the end of phase 3 the last operation of the phase 3 calls (as the last action it performs) the operation ‘pr_phase_ended’ of the control object ‘project’. In this way it signals to the control object that the phase has ended. The control object then ‘sim_calls’ all the participating classes to exit from their current phase-external STD. The last operation of phase 3 is the operation ‘pmm_request_approval_(x)’ of the class ‘project meeting minus’.

At the’ sim_call_phase_ended’ from the control operation ‘pr_phase_ended’, all the (19) classes will (eventually) leave their current phase_external STD and enter the next intermediate state.

Now the ’sim_call_change_to_phase_4’ of the control operation ‘pr_project_life_cyle’ (which is still executing and may or may not have placed this call already) can be handled by the classes of the process fragment. They are now in the correct intermediate state to handle the call. All participating classes will (eventually) transit from the intermediate state to the next state, phase 4. The process fragment comes in its next phase. Phase 4 of the process fragment ‘writing project management documents’ can now take place.

The detection of the end of phase 4 is somewhat more complicated. The operation ‘hcss_allocate_resource’ of the class ‘head computer support section’ and the operation ‘hps_allocate_resource’ of the class ‘head personnel section’ execute indepently of each other. Both operations have to be finished for phase 4 to be completed. This is handled by the operation ‘pr_count_two_phase_ended’ of the control object ‘project’ in conjunction with two ‘counting’ states in the manager STD of the control object ‘project’.

Both operations ‘hccs_allocate_resources’ and ‘hps_allocate_resources’ call (as their last call before they finish executing) the operation ‘pr_count_two_phase_ended’. The number of calls is counted via transitions by the manager STD of control object to the (next) counting state. When two calls have been placed (and accepted by the manager STD) the manager STD calls autonomously its own operation ‘pr_phase_ended’. The sim_call in this operation to the participating classes finishes phase 4.

All 19 total external STDs of the participating classes will (eventually) transit from their state ‘phase 4’ to their next state (the execution of the process fragment has ended). When the ended process fragment is the last fragment in the project life cycle, this next state is the final state. When the ended process fragment has a successor process fragment in the project life cycle, the next state is the start state of this successor process fragment.

Parallel executing process fragments

The process fragment is ‘started’ by the customer. The customer will invoke the internal operation ‘pr_project_life _cycle’ of a certain project object.

The customer uses it own autonomous operation ‘cu_ project_life_cycle’ to call ‘pr_ project_life_cycle’. The autonomous operation ‘cu_ project_life_cycle’ stays available to the customer in all the phases of the process fragment ‘writing project management documents’ (see the updated external STD of the class customer)

Normaly many projects are in progress at the same time in the WBU (Waco Business Unit). This is modeled as follows. If the customer wants to start another project, he will call again its operation ‘cu_ project_life_cycle’. This operation will now call the operation ‘pr_ project_life_cycle’ of another project object. This other project will use another set of ocurrences of the 19 total external STDs of the participating classes. In this other set it switches all total external STDs from phase to phase, starting with the transition from the start state to the ‘phase 1’-state. All concurrently running instances of the external STDs are valid for that class. This is known as ‘multiplicity of concurrent external STDs’. It is equivalent to the notion of ‘multiplicity of concurrent internal STDs’. The notation convention of the multiplicity of concurrent executing external STDs is just like that used with internal STDs. A solid little circle in the start state indicates a multiplicity of concurrent instances of zero or more. A hollow little circle inside the start state indicates a multiplicity of zero or one.
The function of the operation on the transition leaving the start state of an total external STD is twofold. Firstly it starts a new instance of the external STD. This is analogous to the act-transition of an internal STD. Secondly it starts the called operation. This is the normal function. So if one instance of an external STD is for example in phase 4 and another project object (representing a second project that has been started by the customer) place a call to the change_to_phase_1 operation of a class participating in the project fragment, then a next instance of the external STD of that class is started and the internal operation ‘change_to_phase_1’ of the second instance of that class is started.

So, the general rule for concurrently executing external STDs can be formulated. If no instance of the external STD is currently executing, and the operation labeling the (a) transition leaving the start state is called, the external STD starts

executing (and it starts the called operation). If there is currently excuting an instance of the external STD and the operation labeling the (a) transition leaving the start state is called, two possiblities exist. If the multiplicity of the external STD is zero or one, nothing will happen. If the multiplicity of the external STD is zero or more, another instance of the external STD starts executing (and it starts the called operation).

- multiple instances of process fragment

Now suppose that there are two projects started, then each participating class will have two (identical instances of) total external STDs that run in parallel. Both total external STDs are in use at the same time.

On a higher level one can say that the process fragment has now two instances executing. One for each project. The projects may be in the same phase or in a different phase. This is in compliance with the definition in [FIN]. There a project is defined as an instantiation of a process. Two projects are thus two instantiations of the (same) process that run concurrently. The instantiations may or may not be in the same phase.

It is assumed that the internal operations in one instance of the process fragment place their calls with the manager STDs belonging to that instance. This assumption is made to make the model conceptually more clear. The assumption is not strictly needed because the model will also work if the internal operations place their calls with a manager of another instance of the process fragment.

6.3.1 Class diagrams (integration)

This paragraph describes the ‘class diagrams’ of the model, as they apply to the integration. The integration of the four phase-models into one model of the process fragment ‘writing project management documents’ results in the following additions to the class diagram of the process fragment. The control class ‘project’ with the internal (control) operations ‘pr_project_life_cycle’, ‘pr_phase_ended’ and ‘pr_count_two_phase_ended’ is added to the class diagram.

The operation ‘pr_phase_ended’ is called by the last operation of particular phase. In this way the last operation signals to the control object that the phase has ended. The operation ‘pr_phase_ended’ then places a simultaneous call (sim_call) to all the classes participating in the phase. All these classes will then leave their current phase-external STD and this phase is then ended.

The operation ‘pr_count_two_phase_ended’ is used to in situations where there are two ‘last’ operations. Both ‘last’ operations must finish before the phase can end. The use of the counting operation ‘pr_count_two_phase_ended’ is explained later on in this chapter.

For the 19 participating classes the ‘phase changing’-operations ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and ‘phase_ended’ are added. Also the operation ‘cu_project_life_cycle’ is added to the class ‘customer’.

figure 6.1 Class diagram : classes, attributes and operations (additions wrt integration)

Import-export diagram

The execution sequence of the process fragment as explained above results in the following import-export diagram. The diagam shows the use of the ‘phase-changing’- and ‘phase-ended’-operations of the classes participating in the process fragment by the control class ‘project’. Also shown are the use of the operation ‘pr_project_life_cycle’ and ‘cu_project_ life_cycle’ by the class ‘customer’, the use of the operation ‘pr_phase_ended’ by the classes ‘make or buy meeting’, ‘head production section’, ‘project meeting minus’ and ‘project’ and the use of the operation ‘pr_count_two_ phase_ ended’ by the classes ‘head computer support section’ and ‘had personnel section’.

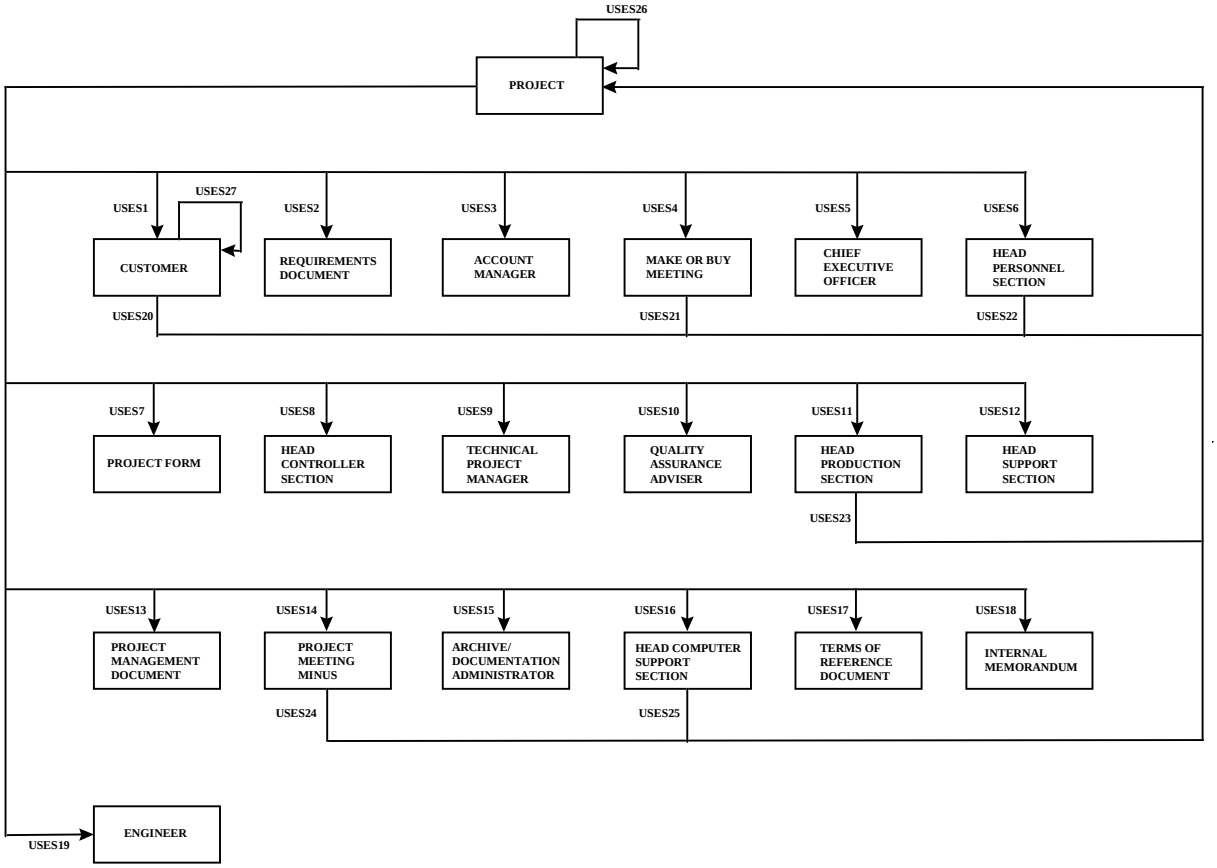


figure 6.2 Class diagram : classes and uses associations (new wrt integration)

The customer uses it own autonomous operation ‘cu_ project_life_cycle’ to call ‘pr_ project_life_cycle’. This starts the process fragment.

The operation ‘pr_ project_life_cycle’ then ‘sim_calls’ the operations ‘change_to_phase_1’ of all the 19 participating classes. All 19 total external STD will transit (eventually) from their state ‘start’ to their state ‘phase 1’.

The process fragments enters in its phase 1 behavior. At the end of phase 1 the operation ‘mb_request_decision_(x)’ of the class ‘make or buy meeting’ calls the operation ‘pr_phase_ended’ of the control class ‘project’. The operation ‘pr_phase_ended’ starts executing and it ‘sim_calls’ the operations ‘phase_ended’ of all the 19 participating classes. All 19 total external STD of the participating classes will (eventually) enter their state ‘intermediate_1’.

The ‘sim_call’ of the operation ‘pr_project_life_cycle’ (sim_call_change_to_phase_2) can then be handled by the 19 participating classes. This ‘sim_call’ may already been placed or it may still have to come. When the 19 participating classes service the ‘sim_call_change_to_phase_2’, their total external STDs transit to their phase 2 behavior.

The process fragments enters in its phase 2 behavior. At the end of phase 2 the operation ‘hprs_pmm_ request_ approval_(x)’ of the class ‘head production section’ calls the operation ‘pr_phase_ended’ of the control class ‘project’. The operation ‘pr_phase_ended’ starts executing and it ‘sim_calls’ the operations ‘phase_ended’ of all the 19 participating classes. All 19 total external STD of the participating classes will (eventually) enter their state ‘intermediate_2’.

The ‘sim_call’ of the operation ‘pr_project_life_cycle’ (sim_call_change_to_phase_3) can then be handled by the 19 participating classes. This ‘sim_call’ may already been placed or it may still have to come. When the 19 participating classes service the ‘sim_call_change_to_phase_3’, their total external STDs transit to their phase 3 behavior.

The process fragments enters in its phase 3 behavior. At the end of phase 3 the operation ‘pmm_request_approval_(x)’ of the class ‘project meeting minus calls the operation ‘pr_phase_ended’ of the control class ‘project’. The operation ‘pr_phase_ended’ starts executing and it ‘sim_calls’ the operations ‘phase_ended’ of all the 19 participating classes. All 19 total external STD of the participating classes will (eventually) enter their state ‘intermediate_3’.

The ‘sim_call’ of the operation ‘pr_project_life_cycle’ (sim_call_change_to_phase_4) can then be handled by the 19 participating classes. This ‘sim_call’ may already been placed or it may still have to come. When the 19 participating classes service the ‘sim_call_change_to_phase_4’, their total external STDs transit to their phase 4 behavior.

The process fragments enters in its phase 4 behavior. At the end of phase 4 both the operation ‘hcss_allocate_resource’ of the class ‘head computer support section’ and the operation ‘hps_allocate_resource’ of the class ‘head personnel section’ call the operation ‘pr_count_two_phase_ended’ of the control class ‘project’. The control object ‘project’ checks (via the counting states in its manager STD) that both operations have placed their call. The control object ‘project’ then starts autonomously its own operation ‘pr_phase_ended’. This operation ‘sim_calls’ the operations ‘phase_ended’ of all the 19 participating classes. All 19 total external STD of the participating classes will (eventually) transit from their state ‘phase 4’ to their next state (the execution of the process fragment has ended). When the ended process fragment is the last fragment in the project life cycle, this next state is the final state. When the ended process fragment has a successor process fragment in the project life cycle, the next state is the start state of this successor process fragment.

In terms of the values of the ‘import-list’-attributes of the uses- and event-associations this amounts to the following :

uses1 until uses 19 :

imported operation	imported by
change_to_phase_1	pr_project_life_cycle
change_to_phase_2	pr_project_life_cycle
change_to_phase_3	pr_project_life_cycle
change_to_phase_4	pr_project_life_cycle
phase_ended	pr_phase_ended

uses20 : imported operation	imported by
pr_project_life_cycle	cu_project_life_cycle

uses21 : imported operation	imported by
pr_phase_ended	mb_request_decision_(x)

uses22 : imported operation	imported by
pr_count_two_phase_ended	hps_allocate_resource

uses23 : imported operation	imported by
pr_phase_ended	hprs_pmm_ request_approval_(x)

uses24 : imported operation	imported by
pr_phase_ended	pmm_request_approval_(x)

uses25 : imported operation	imported by
pr_count_two_phase_ended	hcss_allocate_resource

uses26 : imported operation	imported by
pr_phase_ended	autonomous

uses26 : imported operation	imported by
cu_project_life_cycle	autonomous

6.3.2 State Transition Diagrams (integration)

In the integration of the process fragment 'writing project management documents' the behavior that is relevant for the integration is modeled for the following classes :

- project
- customer
- requirements document
- account manager
- make or buy meeting
- chief executive officer
- head personnel section
- project form
- head controller section
- technical project manager
- quality assurance adviser
- head production section
- head support section
- project management document
- project meeting minus
- archive/documentation administrator
- head computer support section
- terms of reference document
- internal memorandum
- engineer

6.3.2.1 Project (control class)

6.3.2.1.1 Project (control class) : external behavior-STD, organizational view

The class ‘project’ is used as the ‘control’ class for the process fragment ‘writing project management documents’. It has one operation relevant for the organizational view. This is the operation ‘pr_project_life_cycle’. A project in the real world has one (1) project life cycle. So for each project object the operation ‘pr_project_life_cycle’ can only be called once. This is modeled in the external STD.

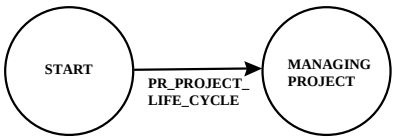


figure 6.3 project : external behavior STD, organizational view

When the state ‘managing project’ is more shown in more detail, the following detailed external STD is the result.

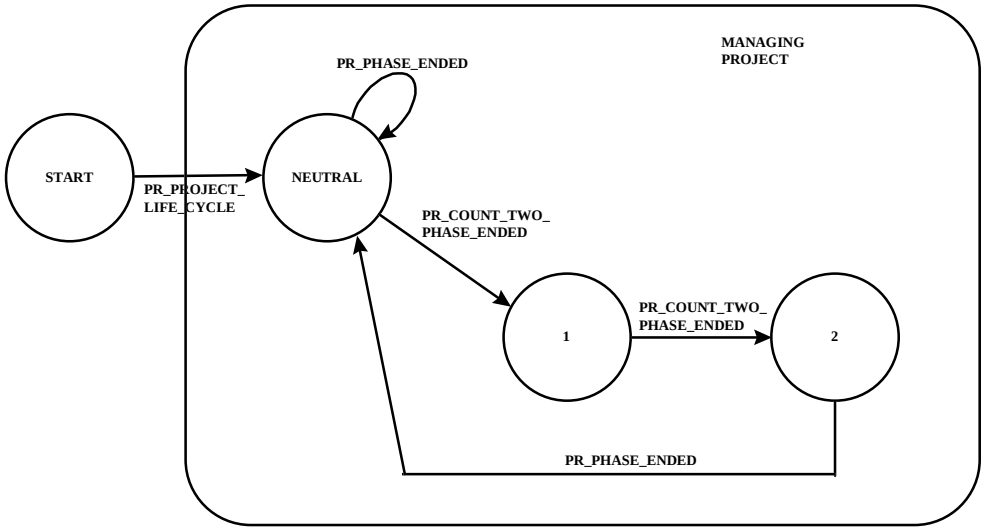


figure 6.4 project : external behavior STD, organizational view (detailed)

The operations ‘pr_phase_ended’ and ‘pr_count_two_phase_ended’ take, together with the states ‘neutral’ and the ‘counting’ states ‘1’ and ‘2’, care of the ‘end’ signal of a process phase. The ‘end’ signal of a phase is a call by the last operation in that process phase to either ‘pr_phase_ended’ or ‘pr_count_two_phase_ended’.

‘pr_phase_ended’ is called when only one (last) operation in a process phase has to reach a certain point in its execution to indicate that the phase has ended. ‘pr_count_two_phase_ended’ is called when two operations in a process phase have to reach a certain point in their execution for the phase to end.

6.3.2.1.2 Project (control class) : external behavior STD, communicative view

The communicative view of the external behavior STD shows the communication details between the external STD and the called operations.

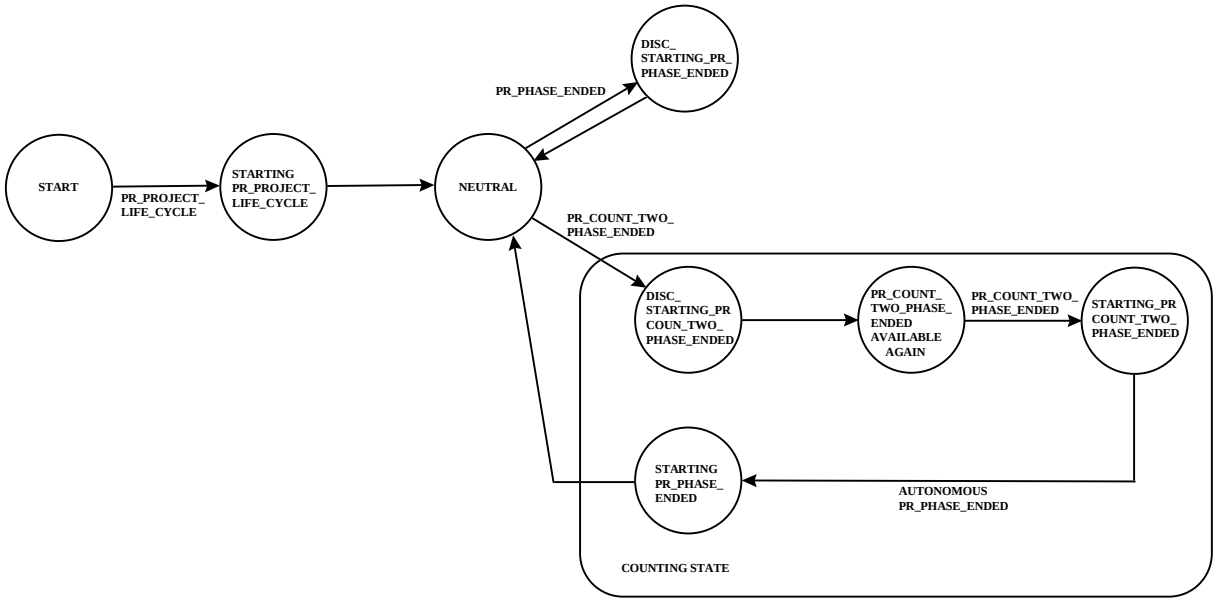


figure 6.5 project : external behavior STD, communicative view

When the customer calls ‘pr_ project_life_cycle’ the manager can (and will) transit to its state ‘starting pr_project_ life_cycle’. When the internal operation ‘pr_project_life_cycle’ of the object ‘project’ is started, the external STD can (and will) transit to its state ‘neutral’. The internal operation ‘pr_project_life_cycle’ is now executing. This operation will take care of the entering of the sequential phases by the total external STDs of the classes that participate in the process fragment. It does so by sequentially sim_calling the operations ‘change_to_phase_x’ of the participating classes.

The ending of a phase (The transition from the state ‘phase x’ to the next intermediate state in the total external STDs of the participating classes) is taken care of by the operations ‘pr_phase_ended’ and ‘pr_count_two_phase_ended.

The external STD of the object ‘project’ is in its state ‘neutral’. Now the phase 1 is ending. I.e. the last operation of phase 1, ‘mb_request_decision_(x)’ of the class ‘make or buy meeting’, calls the operation ‘pr_phase_ended’. The external STD of the object ‘project can (and will) take the transition to the state ‘disc_starting_pr_phase_ended’. The operation ‘pr_phase_ended’ is started and the external STD of ‘project’ transits back to the neutral state. In the execution of ‘pr_phase_ended’ it will ‘sim_call’ the operation ‘phase_ended’ of all the (19) participating classes. The external STDs of the participating classes can (and will) transit to their next ‘intermediate’ state. I.e. the phase is ended.

The still executing operation ‘pr_project_life_cycle’ of ‘project’ can now have its ‘sim_call’ to ‘change_to_phase_2’ serviced by the 19 participating classes. The total external STDs of the participating classes can (and will) enter their next phase.

This mechanism is also used at the end of phase 2. The last operation of phase 2 is the operation ‘hprs_pmm_ request_approval_(x)’ of the class ‘head production section’. The same mechanism is also used at the end of phase 3. The last operation of phase 3 is the operation ‘pmm_request_approval_(x)’ of the class ‘project meeting minus’.

To distinguish between the four callers of the internal operation ‘pr_phase_ended’, the starting state of this operation in the external STD is a discriminator state. The four callers are ‘mb_request_decision_(x)’, ‘hprs_pmm_ request_ approval_(x)’, ‘pmm_request_approval_(x)’ (see above) and the autonomous call of the object itself (see below).

The external STD of ‘project’ is again in its neutral state. Now phase 4 ends. This means that either ‘hcss_allocate_ resource’ of the class ‘head computer support section’ places a call to ‘pr_count_two_phase_ended’ or ‘hps_allocate_ resource’ of the class ‘head personnel section’ places a call to ‘pr_count_two_phase_ended’. When the external STD of ‘project’ detects that someone has made a call to ‘pr_count_two_phase_ended, it can (and will) transit to the state ‘disc_starting_pr_count_two_phase_ended’. This is a discriminator state which distinguishes between the two possible callers. In this state the operation ‘pr_count_two_phase_ended’ is started. This is a so-called ‘nop’, a no-operation. This means that the operation performs no function while executing. The sole purpose of such an operation is to enable the corresponding transition of the external STD within the existing SOCCA framework.
The external STD of ‘project’ is now in its state ‘disc_starting_pr_counting_two_phase_ended’. Here it waits for the started instance of the operation ‘pr_counted_two_phase_ended’ to reach its trap T-2. This means that the operation is

really executing and a second instance of the operation could be started. When the started instance reaches its trap T-2, the external STD of ‘project’ transits to the state ‘pr_count_two_phase_ended available again’. Here it waits for the second call of the operation ‘pr_count_two_phase_ended’. When this calls comes, the external STD can (and will) transit to its state ‘starting_pr_count_two_phase_ended’. When it does so it will start another instance of the internal ‘pr_count_two_phase_ended’-operation. Note : the first instance may still be executing when the second instance is started.

Now the external STD has counted two calls to its operation ‘pr_counted_two_phase_ended’. It knows now that phase 4 can be ended. The ‘project’ object now calls autonomously it own operation ‘pr_phase_ended’ and transits to its state ‘starting_pr_phase_ended’. When this operation is started, the external STD transits back to its state ‘neutral’. The operation ‘pr_phase_ended’ sim_calls the operation ‘phase_ended’ of the 19 participating classes. The participating classes then end their ‘phase 4’ behavior. I.e. phase 4 ends.

6.3.2.1.3 Project (control class) : internal behavior-STDs

The class ‘project’ has three internal behavior STDs. These are the operations ‘pr_phase_ended’, ‘pr_count_two_ phase_ended’ and ‘pr_project_life_cycle’.

The operation ‘pr_phase_ended’ places a ‘sim_call’ to the operation ‘phase_ended’ of all the classes that are participating in the process fragment ‘writing project management documents’. The participating classes then exit from their current phase behavior. The multiplicity of concurrent executing STDs is zero or 1. In this way only one ‘phase-ending’ at any one time can take place.

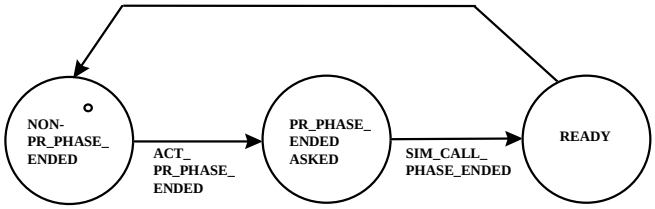


figure 6.6 int-pr_phase_ended : internal behavior STD

The operation ‘pr_count_two_phase_ended’ is a no-operation (nop). This means that the operation performs no function while executing. The sole purpose of such an operation is to enable the corresponding transition of the external STD within the existing SOCCA framework. The multiplicity of concurrency of the STD is zero or more. There may be concurrently executing instances. This means that the two callers of this operation can be serviced more quickly.

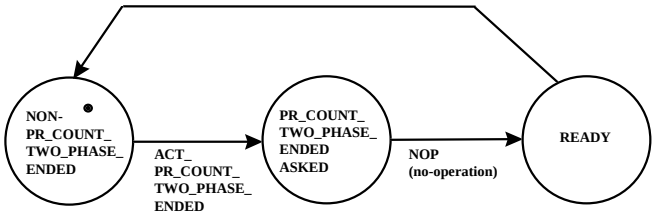


figure 6.7 int-pr_count_two_phase_ended : internal behavior STD

As seen by the external STD , the internal operation ‘pr_project_life_cycle’ appears as only two states. The state ‘non-active’ and the state ‘project life cycle’ (active). The muliplicity of the operation is zero or 1. Per object there can only be one instance of this STD in execution.

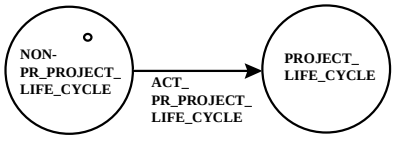


figure 6.8 int-pr_project_life_cycle : internal behavior STD

The class ‘project’ is not only the control class of the process fragment ‘writing project management documents’. It is also the control class of the process fragment ‘changing project management documents’ and of the process fragment ‘closing project’. These two process fragments are not modeled in detail this thesis. They will be represented here by the highest possible aggregation level. Namely one state ‘changing project management documents’ for the process fragment ‘changing project management documents’ and one state ‘closing project’ for the process fragment ‘closing project’. The integration of the models of the process fragments ‘writing project management documents’, ‘changing project management documents’ and ‘closing project’ into one bigger process model for the total process (fragment) ‘Software Project Planing’ (SPP-process) is shown in a more detailed view of the state ‘project life cycle’ of the internal operation ‘pr_project_life_cycle’.

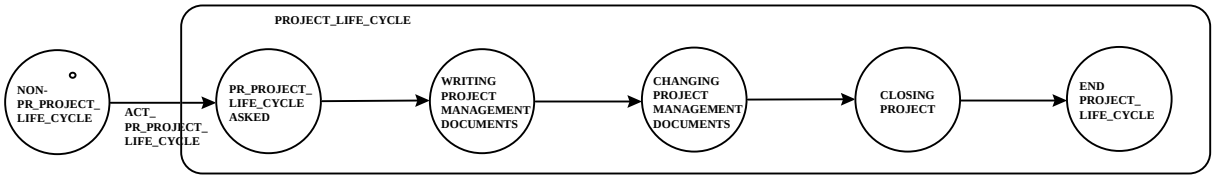


figure 6.9 int-pr_project_life_cycle : internal behavior STD

After being started the operation enters the state ‘writing project management documents’. When this state is finished the operation continues in the state ‘changing project management documents’. When this is finished the operation proceeds with ‘closing project’. After the closing of a project its life cycle is ended. The project enters its final state ‘end project life cycle’.

It can be seen from the operation that the sequential dependency between the process fragments is modeled by the sequention of their respective aggregate states. The internal operation goes through these states (in which it places ‘phase changing’ calls to the total external STDs of the classes participating in the process fragments ‘writing project management’, ‘changing project management documents’ and ‘closing project’ respectively. It places these calls without knowing in which states the total external STDs of the participating classes are in. It does no have to know this. It can just place its calls. The ‘phase-ended’ calls (of the last operation of a phase (or bigger process fragment) in conjunction with the ‘intermediate’ states of the total external STDs will guarantee that a phase (or bigger process fragment) is correctly finished before the next phase (or bigger process fragment) can be entered (by servicing the ‘phase changing’ calls of the operation ‘pr_project_life_cycle’).

The three process fragments are sequential in real life. This is modeled by the sequential states in the operation ‘pr_project_life_cycle’. But something more is needed. All three process fragments have constructed ‘total external STDs’ for their participating classes. These ‘total external STDs’ of the three fragments have to be constructed into ‘grand total external STDs’ for the total process ‘Software Project Planning’. The construction of a ‘grand total external STD’ is the same as the construction of a ‘total external STD’. The final state of the model of one process fragment is just merged with the start state of the model of the next process fragment to form an intermediate state.

So, the integration algorithm is the same for every level in the model. Namely, the method for the integration of phases into one process fragment is the same as the method used for the integration of process fragments into a bigger process (fragment). This ensures that the model can be scaled up in a transparant and easy manner.

When the process fragment in real life are parallel instead of sequential the integration is slightly different. In this case the ‘total external STDs’ of the process fragments are not coupled via an intermediate state. They remain separate. Both ‘total external STDs’ are now valid at the same time. The control operation of a control object still shows the process fragments as consecutive states. When the internal operation now executes and performs its ‘phase changing’ calls in sequence, first the one process fragment is started and the the second. This second process fragment will execute in parallel with the first process fragment. This is conform the parallel execution of different projects (process fragments) as explained in the beginning of this chapter. When using this concept (‘parallel integration’) plus the ‘sequential integration’ the operation ‘pr_project_life_cycle’ can be extended in such a way that the class ‘project’ becomes the control class of the total ‘Corporate Process Model (CPM)’. This model integrates the behavior models of all the KPA-processes of the WBU organization. Another, more modular way, to scale up, is to apply the integration algorithm to the ‘control’ classes of the different, already integrated sub-models. These ‘control’ classes, managing the different sub-models, are then managed by a ‘master’ control class.

The operation ‘pr_project_life_cycle’ is also the control operation for the integration of the phases 1, 2, 3, and 4 into the process fragment ‘writing project management document’. To look more closely into this integration, we will first

aggregate (abstract) the not relevant states of the operation into the states ‘pre writing project management documents’ and ‘post writing project management documents’.

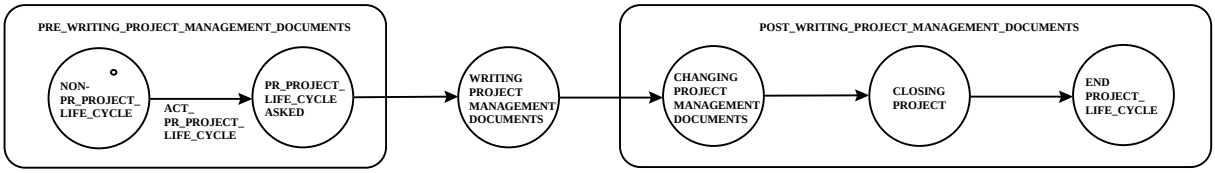


figure 6.10 int-pr_project_life_cycle : internal behavior STD

Then we will look deeper into the state ‘writing project management document’ by giving an exploded view of this state.

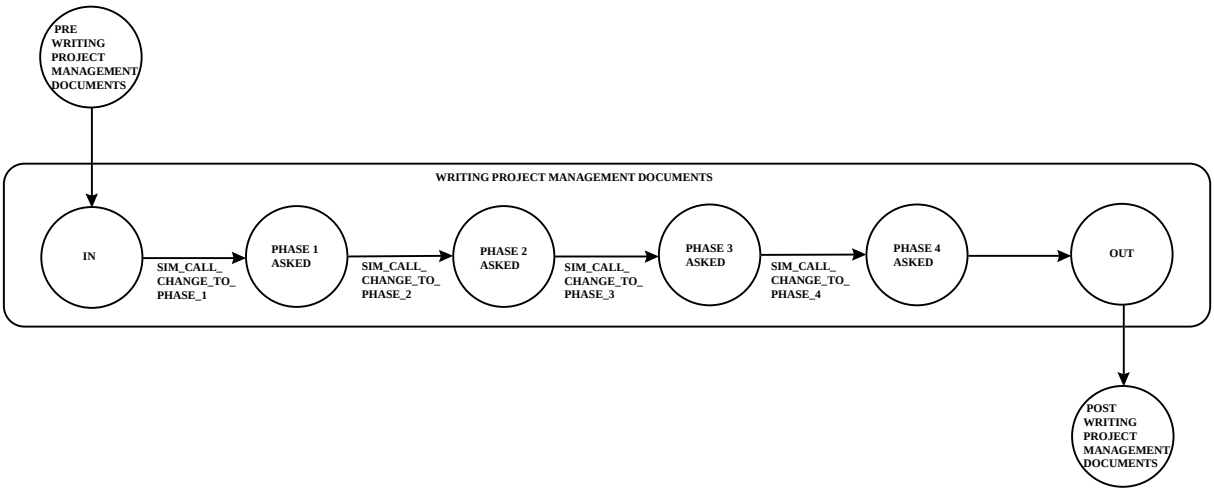


figure 6.11 int-pr_project_life_cycle : internal behavior STD

Inside the state ‘writing project management documents’ the four ‘sim_calls’ to the phase-changing operations take place. As already explained the phase-ended calls (of the last operation inside a phase) will ensure that the phase-changing calls are serviced only after a phase has ended. The function of the states ‘in’ and ‘out’ is the ‘de-coupling’ of the process fragment model inside the state ‘writing project management documents’ from the rest of the operation ‘pr_project_life_cycle’. The operation ‘pr_project_life_cycle’ can be constructed (on the higher aggregate level) without knowledge (of the names of) the operations that are called inside the state ‘writing project management documents’.

6.3.2.1.4 Project (control class) : manager-STD

The manager STD is the same as the external STD, communicative view. Therefore no separate figure is needed for the manager STD. In it states it prescribes the subprocesses for its employees. The transitions are guarded with a combination of traps. The employees have to be in these traps for the transition to be enabled.

The notation CPSx stands for Consolidated Prescribed Subprocesses and is a set of CCx’s. The notation CCx stands for ‘Caller(s)-Callee’ and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for ‘Traps Logical Formula’ and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

The CPSx’s have the same name as the state for which they are valid. E.g. ‘disc_starting_pr_count_two_phase_ended’ is the name of a state in the external STD and it is also the name of the CPS for that state in the manager STD. The TLF-x’ have the same name as the operation that is started when the transition is taken. E.g. ‘pr_count_two_phase_ ended’ is the name of the operation that is started (indicated in the external STD) and it is also the name of the TLF that is guarding the transition in the manager STD. Transitions with no TLF (with no name in the external STD) are automatic transitions (unless otherwise indicated in the definitions below). Traps that are not mentioned in the TLF are don’t cares.

The name of CCx's will include the full names of the caller and callee operations separated by a tilde (~) character. E.g. the CC of the caller operation 'cu_project_life_cycle' and the callee 'pr_project_life_cycle' will be named 'cu_project_life_cycle ~ pr_project_life_cycle'.

Subprocesses and traps will be given as much of their full name as is necessary to avoid ambiguity. This full name is constructed with the 'dot'-notation. E.g. 'cu_project_life_cycle.S1' is subprocess S1 of the operation cu_project_life_cycle'. If there are more subprocesses with the same name prescribed to an operation, the manager prescribing each subprocess will be added. E.g. 'cu_project_life_cycle.S1_wrt_project. This is the subprocess S1 with respect to (wrt, prescribed by) the manager of the class 'project'. In the same way the naming of the traps is handled. E.g. 'cu_project_life_cycle.S1.T1_wrt_project' is the trap T-1 prescribed by the 'project' manager STD in the subprocess S1 (which is also prescribed by the 'project' manager STD) of the operation 'cu_project_life_ cycle'. If the class of an operation is not clear, the class will be prefixed to the operation name. E.g. 'customer.cu_project_ life_cycle' is the operation 'cu_start_project_life_cycle' of the class 'customer'.

The CPSs, CCs and TLFs for this manager are :

CPS 'start' =
{cu_project_life_cycle.S1_wrt_project ~ pr_project_life_cycle.S1_wrt_project,
{mb_request_decision_(x).S1_wrt_project,
hprs_pmm_request_approval_(x).S1_wrt_project,
pmm_request_approval_(x).S1_wrt_project,
autonomous} ~ pr_phase_ended.S1_wrt_project,
{hcss_allocate_resource.S1_wrt_project,
hps_allocate_resource.S1_wrt_project} ~ pr_count_two_phase_ended.S1_wrt_project}

TLF 'pr_project_life_cycle' = cu_project_life_cycle.T1.wrt_project and pr_project_life_cycle.T1_wrt_project

CPS 'starting_pr_project_life_cycle' =
{cu_project_life_cycle.S2_wrt_project ~ pr_project_life_cycle.S2_wrt_project,
{mb_request_decision_(x).S1_wrt_project,
hprs_pmm_request_approval_(x).S1_wrt_project,
pmm_request_approval_(x).S1_wrt_project,
autonomous} ~ pr_phase_ended.S1_wrt_project,
{hcss_allocate_resource.S1_wrt_project,
hps_allocate_resource.S1_wrt_project} ~ pr_count_two_phase_ended.S1_wrt_project}

TLF 'no name' = cu_project_life_cycle.T2_wrt_project and pr_project_life_cycle.T2_wrt_project

CPS 'neutral' =
{cu_project_life_cycle.S1_wrt_project ~ pr_project_life_cycle.S1_wrt_project,
{mb_request_decision_(x).S1_wrt_project,
hprs_pmm_request_approval_(x).S1_wrt_project,
pmm_request_approval_(x).S1_wrt_project,
autonomous} ~ pr_phase_ended.S1_wrt_project,
{hcss_allocate_resource.S1_wrt_project,
hps_allocate_resource.S1_wrt_project} ~ pr_count_two_phase_ended.S1_wrt_project}

TLF 'pr_phase_ended' (1^e) = pr_phase_ended.T1_wrt_project
and
(mb_request_decision_(x).T-1_wrt_project
or hprs_pmm_request_approval_(x).T-1_wrt_project
or pmm_request_approval_(x).T-1_wrt_project)

CPS 'disc_starting_pr_phase_ended' = (note : this is a discriminator state)
{cu_project_life_cycle.S1_wrt_project ~ pr_project_life_cycle.S1_wrt_project,
{{mb_request_decision_(x).S2_wrt_project,
hprs_pmm_request_approval_(x).S1_wrt_project,
pmm_request_approval_(x).S1_wrt_project,
autonomous} or
{mb_request_decision_(x).S1_wrt_project,
hprs_pmm_request_approval_(x).S2_wrt_project,

pmm_request_approval_(x).S1_wrt_project,
autonomous} or
{mb_request_decision_(x).S1_wrt_project,
hprs_pmm_request_approval_(x).S1_wrt_project,
pmm_request_approval_(x).S2_wrt_project,
autonomous}} ~ pr_phase_ended.S2_wrt_project,
{hcss_allocate_resource.S1_wrt_project,
hps_allocate_resource.S1_wrt_project} ~ pr_count_two_phase_ended.S1_wrt_project}

TLF 'no name' = pr_phase_ended.T1_wrt_project
and
(mb_request_decision_(x).T-2_wrt_project
or hprs_pmm_request_approval_(x).T-2_wrt_project
or pmm_request_approval_(x).T-2_wrt_project)

TLF 'pr_count_two _phase_ended' (1^e) = pr_count_two_phase_ended.T1_wrt_project
and
(hcss_allocate_resource.T-1_wrt_project
or hps_allocate_resource.T-1_wrt_project)

CPS 'disc_starting_pr_count_two_phase_ended' = (note : this is a discriminator state)
{cu_project_life_cycle.S1_wrt_project ~ pr_project_life_cycle.S1_wrt_project,
{mb_request_decision_(x).S1_wrt_project,
hprs_pmm_request_approval_(x).S1_wrt_project,
pmm_request_approval_(x).S1_wrt_project,
autonomous} ~ pr_phase_ended.S1_wrt_project,
{{hcss_allocate_resource.S2_wrt_project,
hps_allocate_resource.S1_wrt_project} or
{hcss_allocate_resource.S1_wrt_project,
hps_allocate_resource.S2_wrt_project}} ~ pr_count_two_phase_ended.S2_wrt_project}

TLF 'no name' = pr_count_two_phase_ended.T2_wrt_project
(note : the entering of T-2 means that another instance could be started)

CPS 'pr_count_two_phase_ended available again' = (note : this is also a discriminator state)
{cu_project_life_cycle.S1_wrt_project ~ pr_project_life_cycle.S1_wrt_project,
{mb_request_decision_(x).S1_wrt_project,
hprs_pmm_request_approval_(x).S1_wrt_project,
pmm_request_approval_(x).S1_wrt_project,
autonomous} ~ pr_phase_ended.S1_wrt_project,
{{hcss_allocate_resource.S2_wrt_project,
hps_allocate_resource.S1_wrt_project} or
{hcss_allocate_resource.S1_wrt_project,
hps_allocate_resource.S2_wrt_project}} ~ pr_count_two_phase_ended.S1_wrt_project}

TLF 'pr_count_two _phase_ended' (2^e) = pr_count_two_phase_ended.T1_wrt_project
and
(hcss_allocate_resource.T-1_wrt_project
and hps_allocate_resource.T-1_wrt_project)

(note : this pr_count_two_phase_ended.T-1 is on object-level. It indicates that the act-mechanism can start another instance of the operation. The first instance of the operation may still be executing and not yet be in its STD instance-level trap T-1.)

CPS 'starting_pr_count_two_phase_ended' =
{cu_project_life_cycle.S1_wrt_project ~ pr_project_life_cycle.S1_wrt_project,
{mb_request_decision_(x).S1_wrt_project,
hprs_pmm_request_approval_(x).S1_wrt_project,
pmm_request_approval_(x).S1_wrt_project,
autonomous} ~ pr_phase_ended.S1_wrt_project,
{hcss_allocate_resource.S2_wrt_project,

hps_allocate_resource.S2_wrt_project} ~ pr_count_two_phase_ended.S2_wrt_project}

TLF 'pr_phase_ended' (2^e) = pr_phase_ended.T1_wrt_project

CPS 'starting_pr_phase_ended' =
{cu_project_life_cycle.S1_wrt_project ~ pr_project_life_cycle.S1_wrt_project,
{mb_request_decision_(x).S1_wrt_project,
hprs_pmm_request_approval_(x).S1_wrt_project,
pmm_request_approval_(x).S1_wrt_project,
autonomous} ~ pr_phase_ended.S2_wrt_project,
{hcss_allocate_resource.S2_wrt_project,
hps_allocate_resource.S2_wrt_project} ~ pr_count_two_phase_ended.S2_wrt_project}

TLF 'no name' = pr_count_two_phase_ended.T2_wrt_project
and pr_count_two_phase_ended.T2_wrt_project (note : this trap requirement is delayed one state to allow for more parallelism)

6.3.2.1.5 Project (control class) : employee-STDs

The manager STD has 9 employees. These are the callee 'pr_project_life_cycle' and its caller 'cu_project_life_cycle'; the callee 'pr_phase_ended' and its three possible callers 'mb_request_decision_(x)', 'hprs_pmm_request_approval_(x)' and 'pmm_request_approval_(x)'; and the callee 'pr_count_two_phase_ended' and its two possible callers 'hcss_allocate_resource" and 'hps_allocate_resource'.

The employee 'pr_project_life_cycle' has two subprocesses S1 and S2, and two traps T-1 and T-2 with respect to the manager STD of the class 'project. This is according to the caller_callee-construct. In the next three figures first the internal STD of the operation is shown, followed by the two subprocesses S1 and S2.

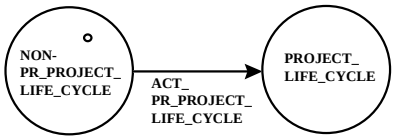


figure 6.12 employee int-pr_project_life_cycle : internal behavior STD

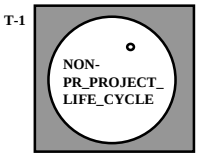


figure 6.13 employee int-pr_project_life_cycle : subprocess S1_wrt_project

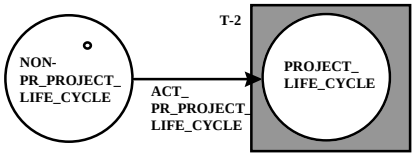


figure 6.14 employee int-pr_project_life_cycle : subprocess S2_wrt_project

The operation 'cu_project_life_cycle' is the caller of 'pr_project_life_cycle'. The employee 'cu_project_life_cycle' has two subprocesses S1 and S2 and two traps T1 and T2 with respect to the manager STD of 'project'. This is according to the caller_callee-construct. In the next three figures first the internal STD of the operation is shown, followed by the two subprocesses S1 and S2.

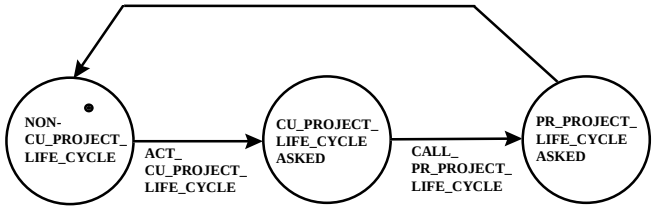


figure 6.15 employee int-cu_project_life_cycle : internal behavior STD

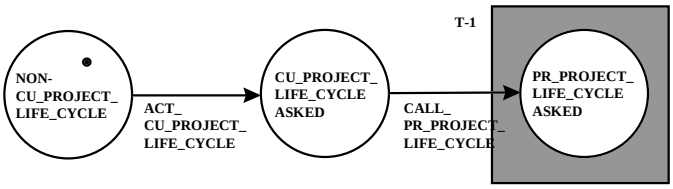


figure 6.16 employee int-cu_project_life_cycle : subprocess S1_wrt_project

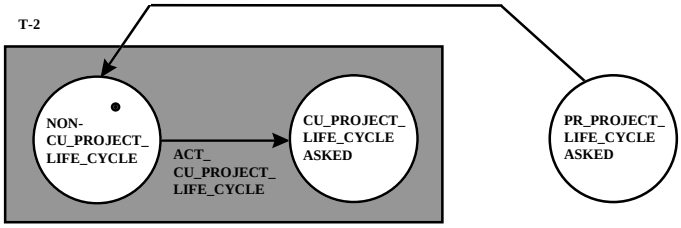


figure 6.17 employee int-cu_project_life_cycle : subprocess S2_wrt_project

The callee operation ‘pr_phase_ended’ has two subprocesses S1 and S2, and two traps T-1 and T-2 with respect to the manager STD of ‘project’. This is according to the caller_callee-construct. In the next three figures first the internal STD of the operation is shown, followed by the two subprocesses S1 and S2.

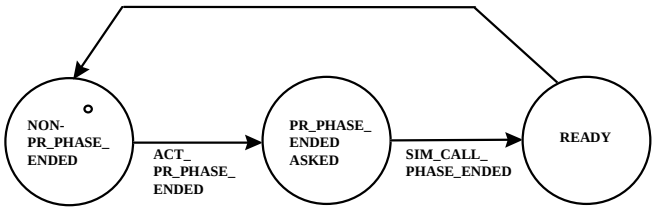


figure 6.18 employee int-pr_phase_ended : internal behavior STD

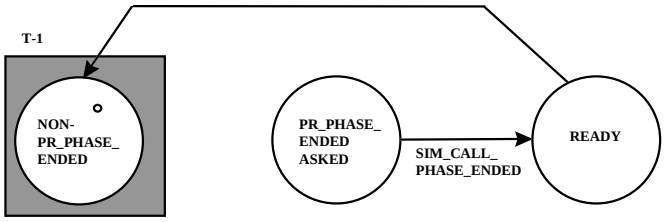


figure 6.19 employee int-pr_phase_ended : subprocess S1_wrt_project

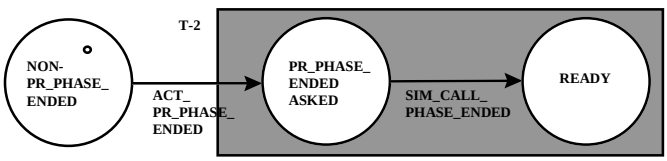


figure 6.20 employee int-pr_phase_ended : subprocess S2_wrt_project

The employee ‘mb_request_decision_(x)’ is one of the possible callers of ‘pr_phase_ended’. It is the last operation of phase 1 of the process fragment ‘writing project management documents’. The call to ‘pr_phase_ended’ is a signal for the control object ‘project’ that the current phase can be terminated. The operation ‘mb_request_decision_(x)’ has already been modeled in phase 1, without regard to later integration. Therefore the call to ‘pr_phase_ended’ is not yet incorporated in the internal STD of ‘mb_request_decision_(x)’. Here, in this paragraph, the call to ‘pr_phase_ended’ will be explicitly modeled into the internal STD of the operation. This is done by viewing deeper in one of the states of the STD and putting the call there. In a real life situation, in a big model, this will result in a lot of rework on an already finished sub-model. It will also cause a high ‘coupling’ between the sub-models and the integrated model. This situation is not advisable. It is therefore recommended to introduce a special ‘finishing state’-indicator. The notation of this ‘finishing state’-indicator is an asterix (*). The modeler of sub-model will then use this ‘finishing state’-indicator. He will place it in the ‘finishing’ state of his sub-model. The modeler of the sub-model is the person with the knowledge which state is the ‘finishing’ state. The integrator (the modeler of the integrated model) inspects the sub-models that he has to integrate. He then knows in (which state of) which operation of a sub-model a ‘phase_ended’ call takes place. He will not remodel the original operation, but he will just add this operation to the list of employees of the control object. In such a way the ‘finishing state’-indicator is a shorthand notation for the caller-part of the caller_callee construct of the operation ‘pr_phase_ended’ and it callers.

In the next four figures first the internal STD of the operation ‘mb_request_decision_(x)’ is shown with the ‘finishing state’-indicator. Then there follows a figure which looks ‘inside’ the ‘finishing’-state and which shows the call to the operation ‘pr_phase_ended’. Then the two subprocesses S1 and S2 with respect to the manager STD of the control object ‘project’ will be given.

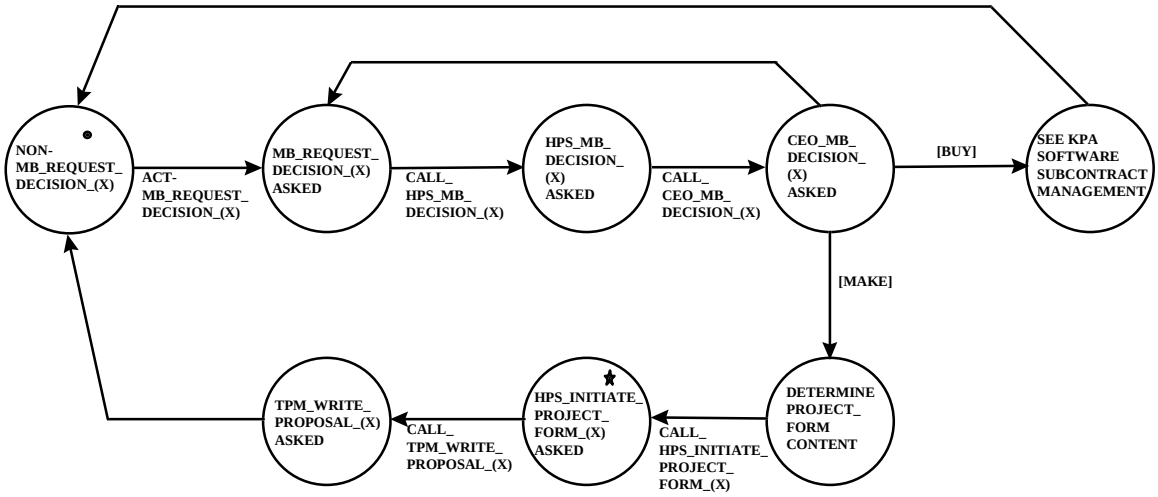


figure 6.21 employee int-mb_request_decision_(x) : internal behavior STD, with ‘finishing state’-indicator

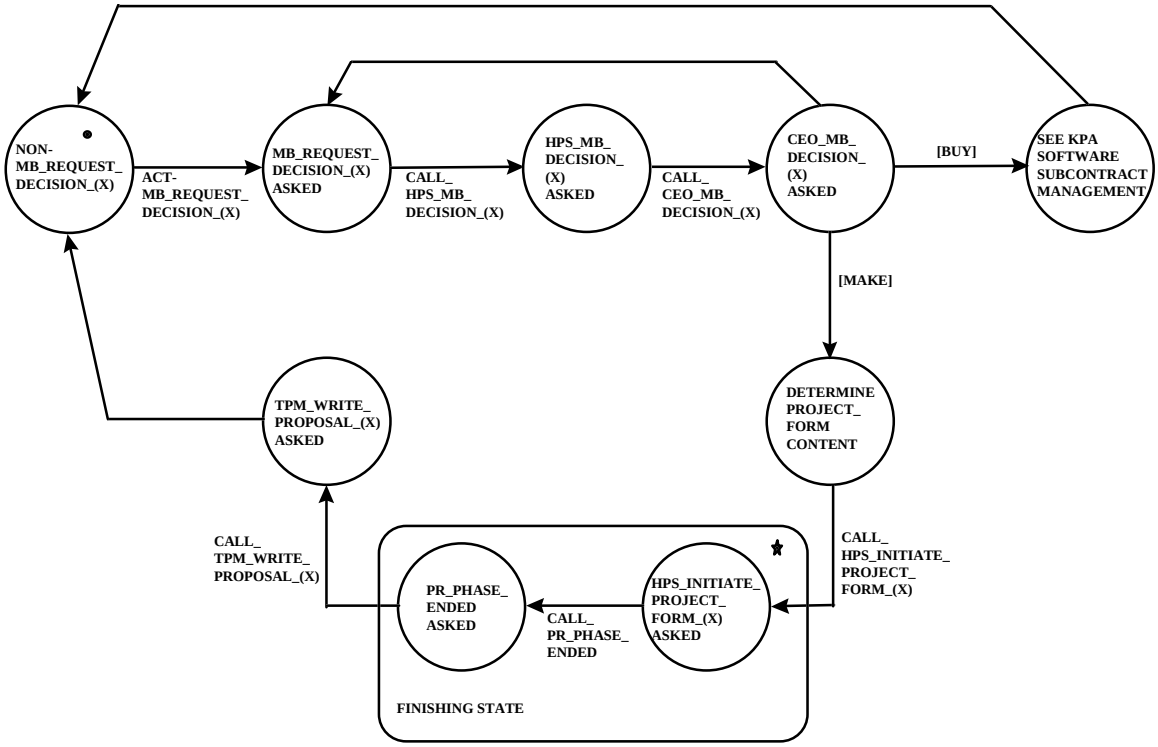


figure 6.22 employee int-mb_request_decision_(x) : internal behavior STD, with exploded view of ‘finishing state’

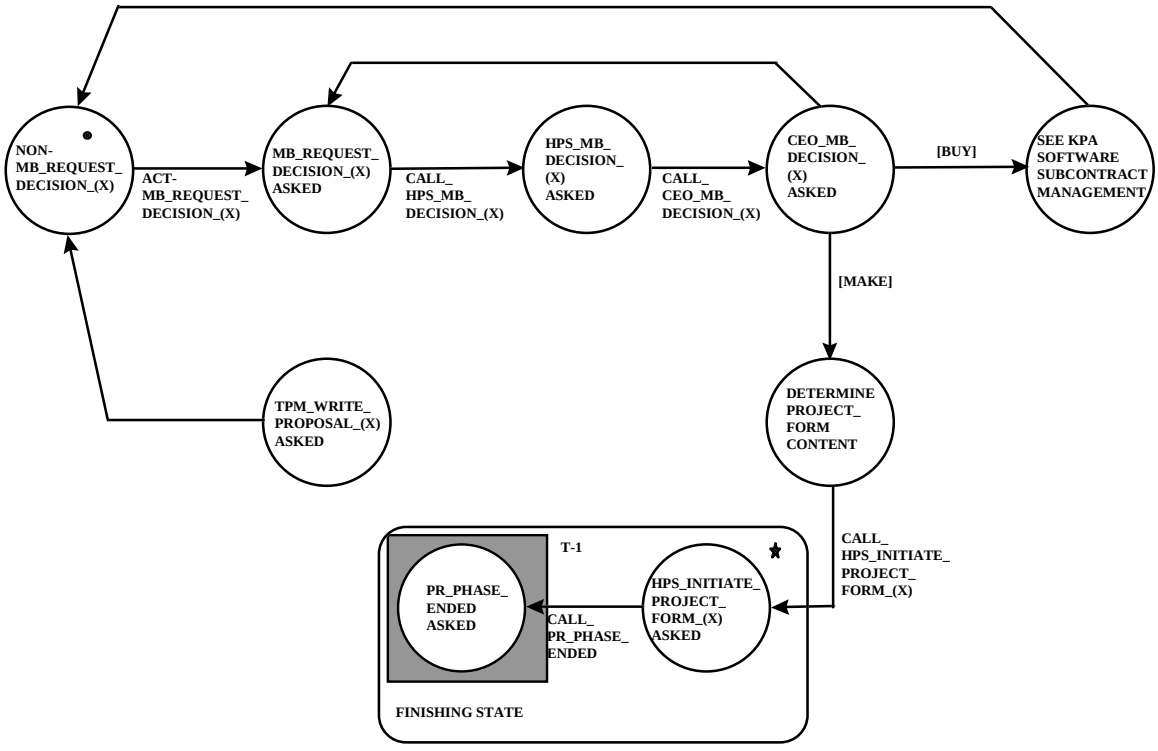


figure 6.23 employee int-mb_request_decision_(x) : subprocess S1_wrt_project

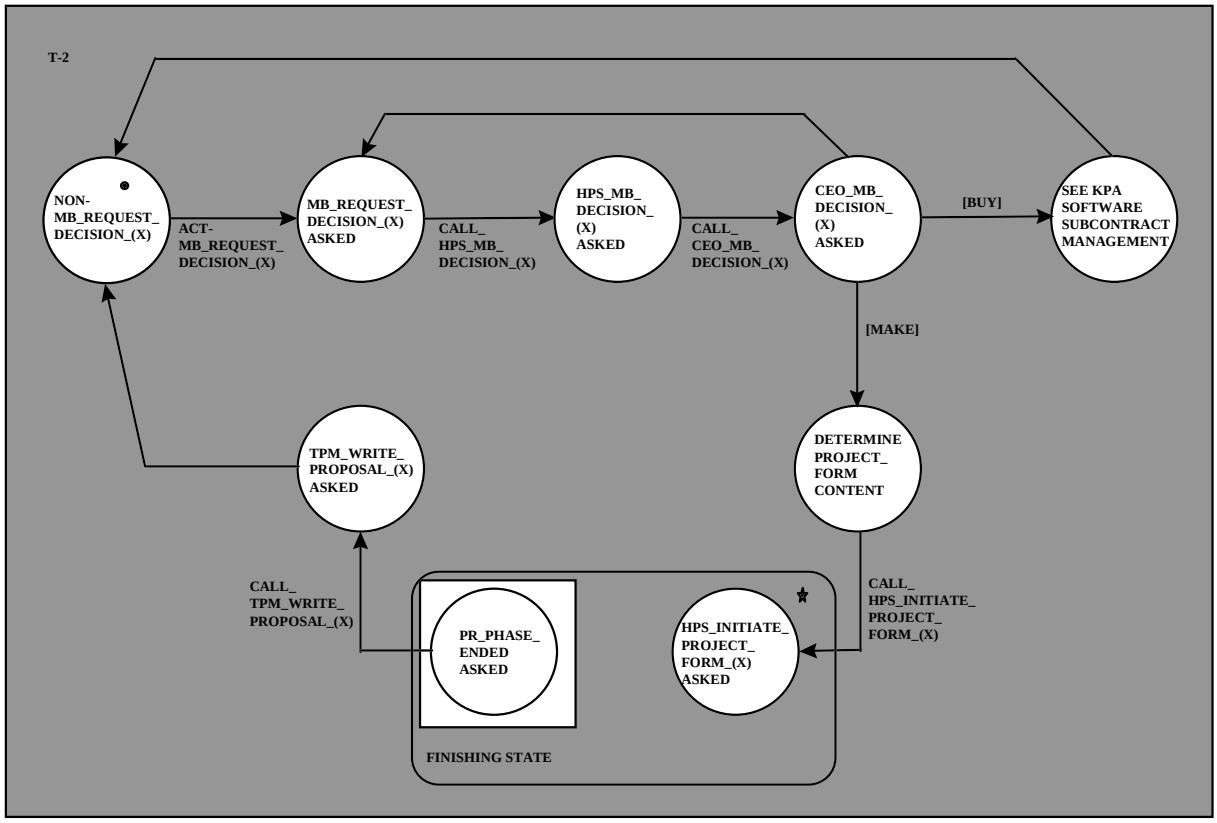


figure 6.24 employee int-mb_request_decision_(x) : subprocess S2_wrt_project

The employee ‘hprs_pmm_request_approval_(x)’ is one of the possible callers of ‘pr_phase_ended’. It is the last operation of phase 2 of the process fragment ‘writing project management documents’. The call to ‘pr_phase_ended’ is a signal for the control object ‘project’ that the current phase can be terminated. The operation ‘hprs_pmm_request_approval_(x)’ has already been modeled in phase 2, without regard to later integration. Therefore the call to ‘pr_phase_ended’ is not yet incorporated in the internal STD of ‘hprs_pmm_request_approval_(x)’. Here, in this paragraph, the call to ‘pr_phase_ended’ will be explicitly modeled into the internal STD of the operation.

In the next four figures first the internal STD of the operation ‘hprs_pmm_request_approval_(x)’ is shown with the ‘finishing state’-indicator. Then there follows a figure which looks ‘inside’ the ‘finishing’-state and which shows the call to the operation ‘pr_phase_ended’. Then the two subprocesses S1 and S2 with respect to the manager STD of the control object ‘project’ will be given.

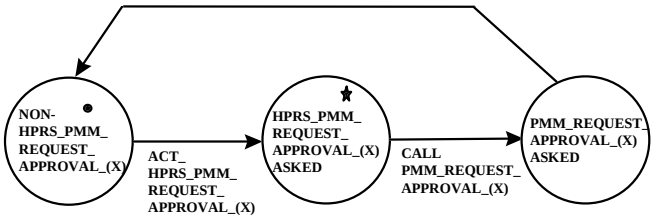


figure 6.25 employee int-hprs_pmm_request_approval_(x) : internal behavior STD, with ‘finishing state’-indicator

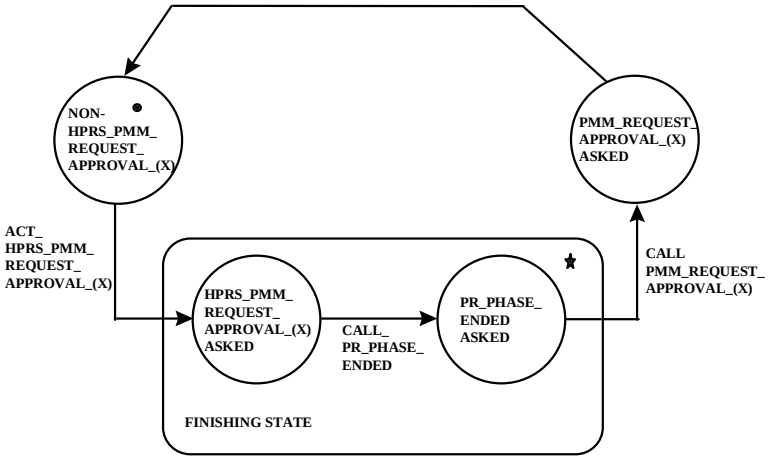


figure 6.26 employee int-hprs_pmm_request_approval_(x) : internal behavior STD, with exploded view of 'finishing state'

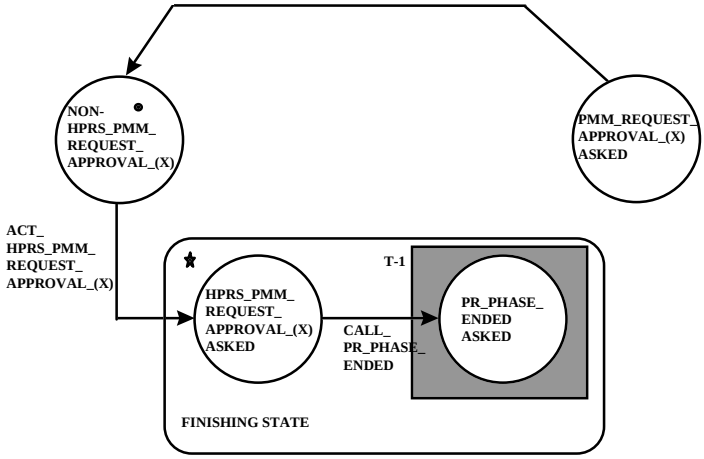


figure 6.27 employee int-hprs_pmm_request_approval_(x) : subprocess S1_wrt_project

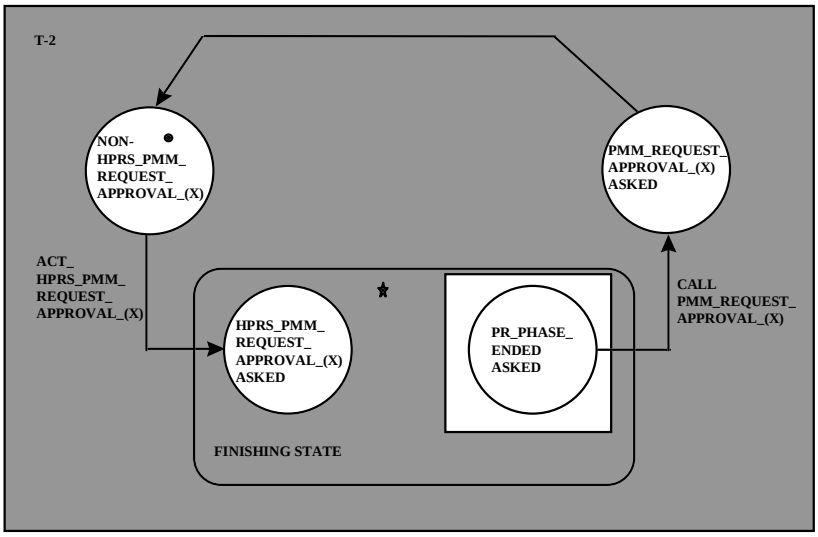


figure 6.28 employee int-hprs_pmm_request_approval_(x) : subprocess S2_wrt_project

The employee 'pmm_request_approval_(x)' is one of the possible callers of 'pr_phase_ended'. It is the last operation of phase 3 of the process fragment 'writing project management documents'. The call to 'pr_phase_ended' is a signal for the control object 'project' that the current phase can be terminated. The operation 'pmm_request_approval_(x)' has already been modeled in phase 3, without regard to later integration. Therefore the call to 'pr_phase_ended' is not yet incorporated

in the internal STD of ‘pmm_request_approval_(x)’. Here, in this paragraph, the call to ‘pr_phase_ended’ will be explicitly modeled into the internal STD of the operation.

In the next four figures first the internal STD of the operation ‘pmm_request_approval_(x)’ is shown with the ‘finishing state’-indicator. Then there follows a figure which looks ‘inside’ the ‘finishing’-state and which shows the call to the operation ‘pr_phase_ended’. Then the two subprocesses S1 and S2 with respect to the manager STD of the control object ‘project’ will be given.

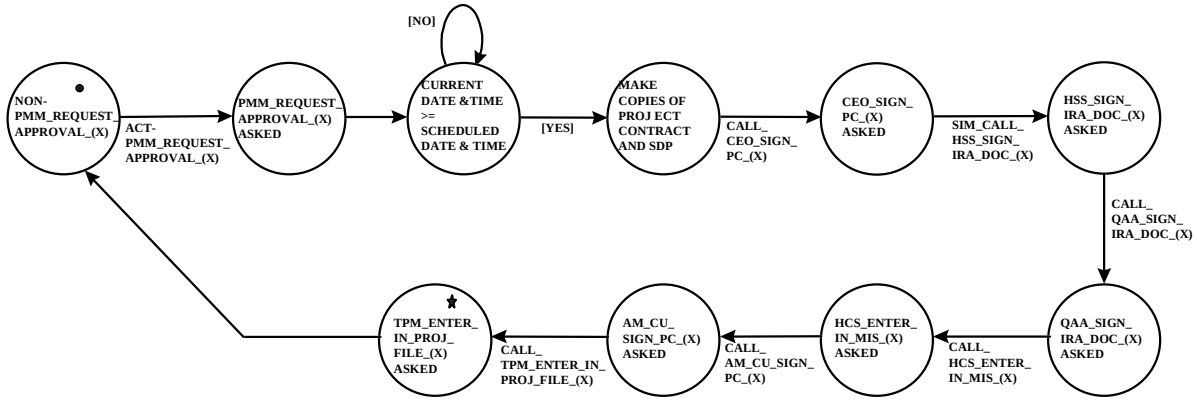


figure 6.29 employee int-pmm_request_approval_(x) : internal behavior STD, with ‘finishing state’-indicator

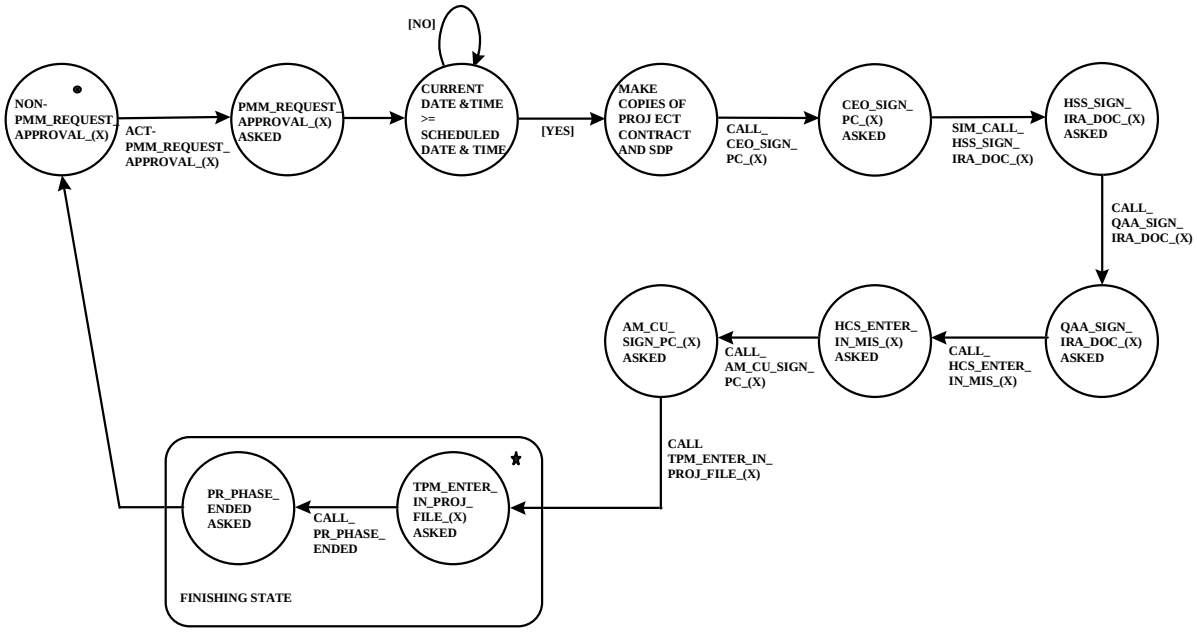


figure 6.30 employee int-pmm_request_approval_(x) : internal behavior STD, with exploded view of ‘finishing state’

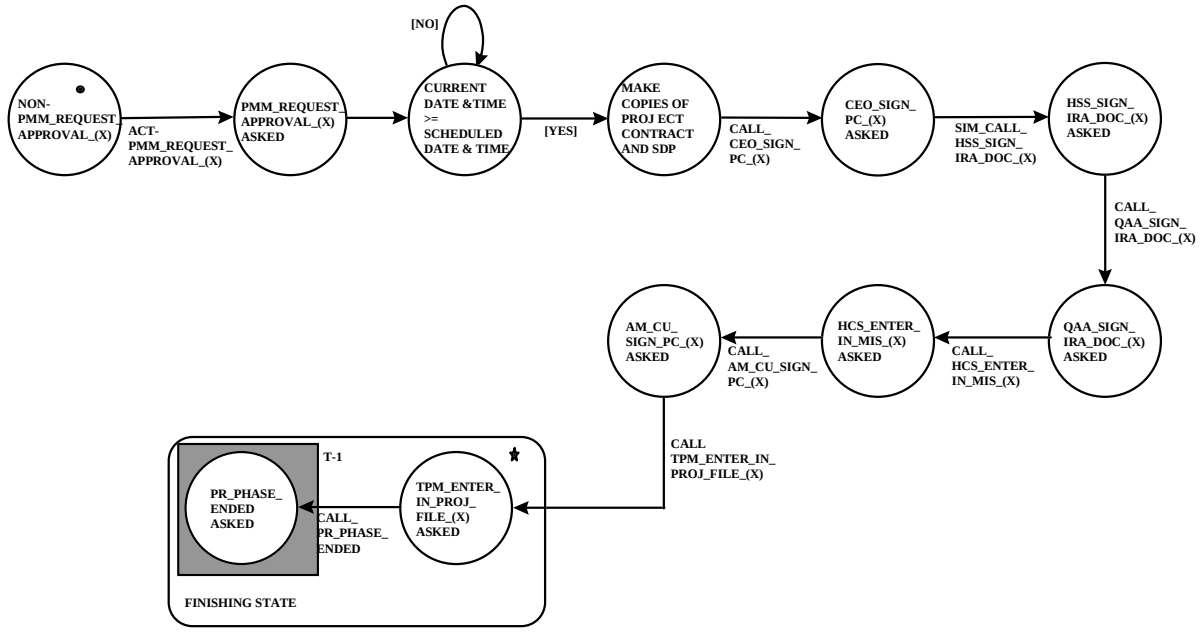


figure 6.31 employee int-pmm_request_approval_(x) : subprocess S1_wrt_project

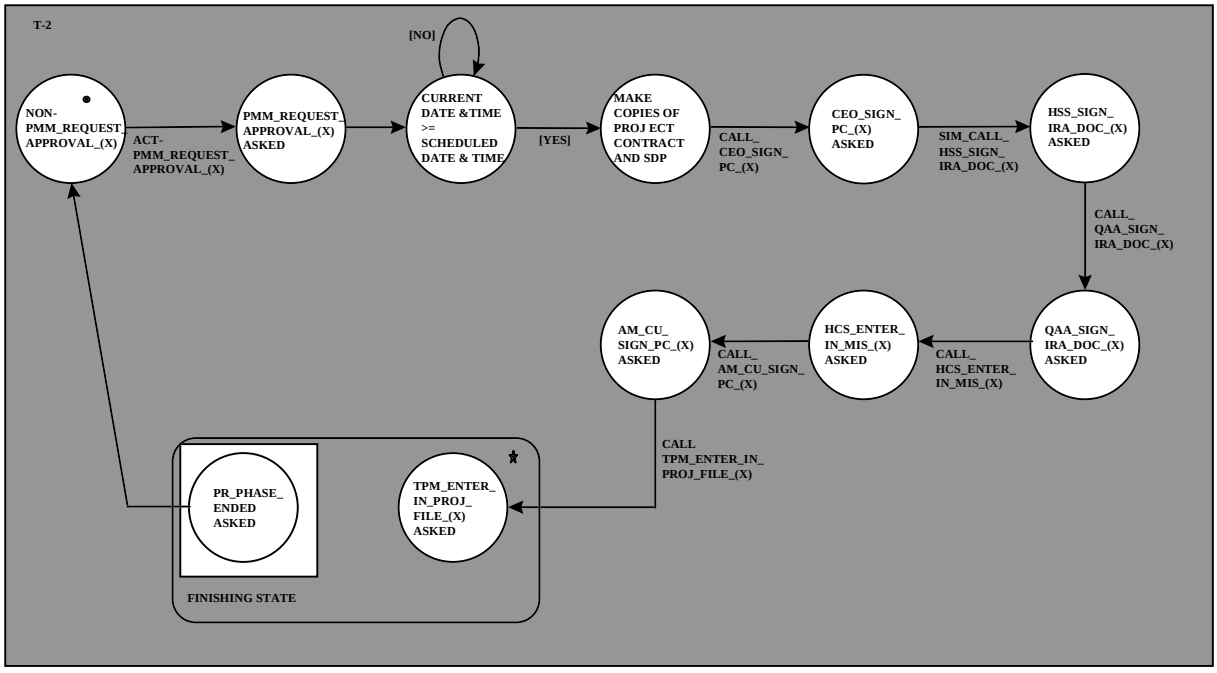


figure 6.32 employee int-pmm_request_approval_(x) : subprocess S2_wrt_project

The employee ‘hps_allocate_resource’ is one of the two possible callers of ‘pr_count_two_phase_ended’. It is one of the two ‘last’ operations of phase 4 of the process fragment ‘writing project management documents’. The other ‘last’ operation of phase 4 is the operation ‘hcss_allocate_resource’. Both operations have to signal to the manager STD of the control object before the manager STD finishes phase 4. The manager STD counts the number of calls to ‘pr_count_two_phase_ended’ in its ‘counting’ state.

The operation ‘hps_allocate_resource’ has already been modeled in phase 4, without regard to later integration. Therefore the call to ‘pr_count_two_phase_ended’ is not yet incorporated in the internal STD of ‘hps_allocate_resource’. Here, in this paragraph, the call to ‘pr_count_two_phase_ended’ will be explicitly modeled into the internal STD of the operation.

Again in a real life situation this re-modeling will not take place. The modeler of sub-model will use the ‘finishing state’-indicator. He will place it in both the ‘finishing’ states of his sub-model. The integrator (the modeler of the integrated

model) inspects the sub-models that he has to integrate. He finds two (2) ‘finishing’ states in the sub-model. In his integrated model he then uses a ‘two-counting’ state, and adds the operation ‘pr_count_two_phase_ended’ to the operations of integrated model. He will not remodel the original operation ‘hps_allocate_resource’, but he will just add this operation to the list of employees of the control object.

This integration mechanism can easily be scaled up. If the modeler of the integrated model finds three ‘finishing’ states in some sub-model, he will add a ‘three-counting’ state to the manager of the control object. Plus he will add the operation ‘pr_count_three_phase_ended’, etc.

In the next four figures first the internal STD of the operation ‘hps_allocate_resource’ is shown with the ‘finishing state’-indicator. Then there follows a figure which looks ‘inside’ the ‘finishing’-state and which shows the call to the operation ‘pr_count_two_phase_ended’. Then the two subprocesses S1 and S2 with respect to the manager STD of the control object ‘project’ will be given.

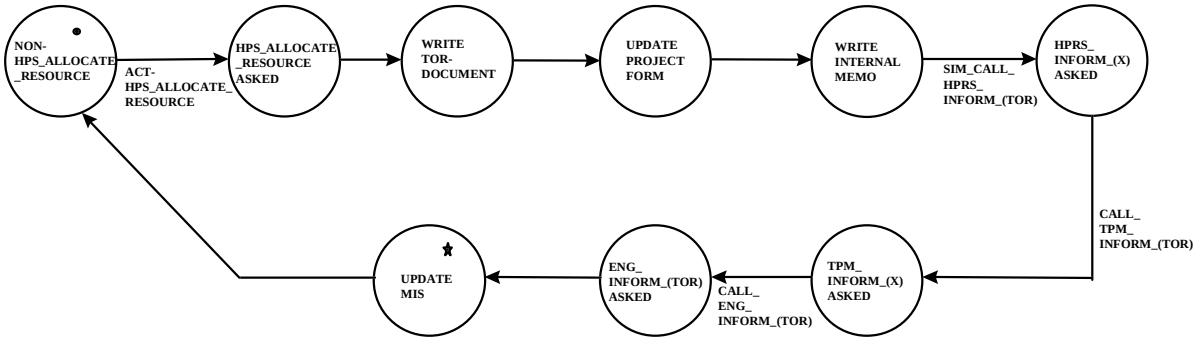


figure 6.33 employee int-hps_allocate_resource : internal behavior STD, with ‘finishing state’-indicator

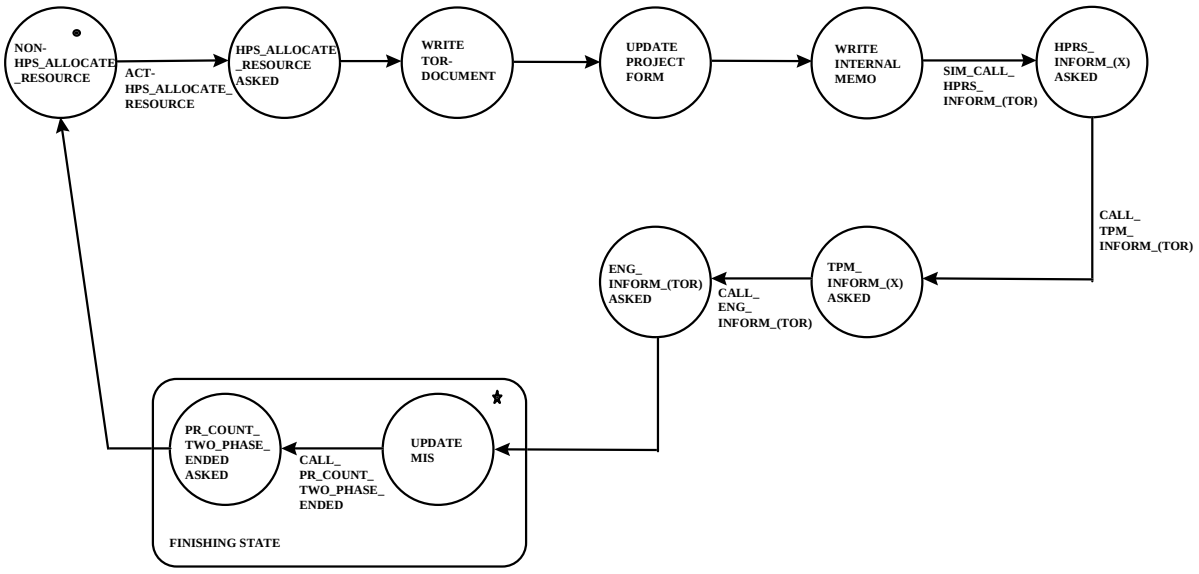


figure 6.34 employee int-hps_allocate_resource : internal behavior STD, with exploded view of ‘finishing state’

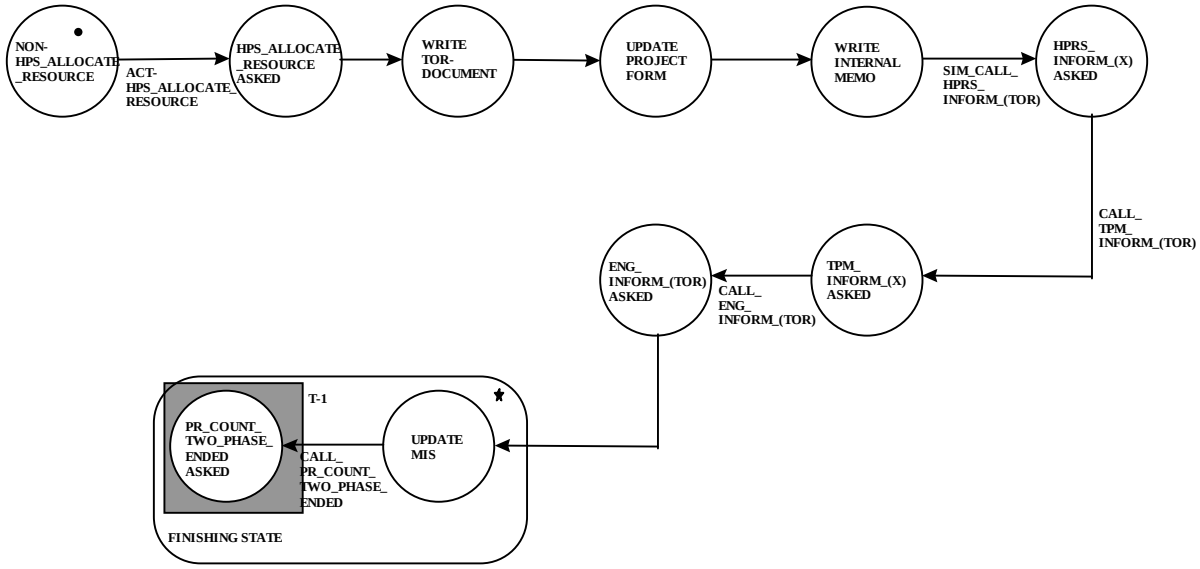


figure 6.35 employee int-hps_allocate_resource : subprocess S1_wrt_project

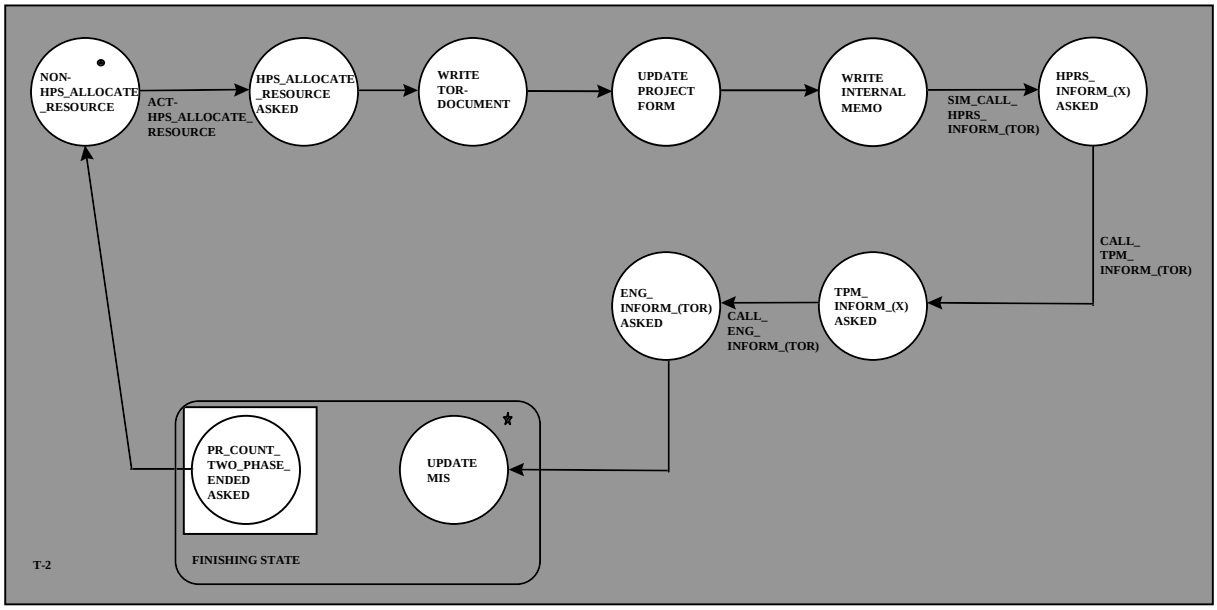


figure 6.36 employee int-hps_allocate_resource : subprocess S2_wrt_project

The employee ‘hcss_allocate_resource’ is one of the two possible callers of ‘pr_count_two_phase_ended’. It is one of the two ‘last’ operations of phase 4 of the process fragment ‘writing project management documents’. The other ‘last’ operation of phase 4 is the operation ‘hps_allocate_resource’. Both operations have to signal to the manager STD of the control object before the manager STD finishes phase 4. The manager STD counts the number of calls to ‘pr_count_two _phase_ended’ in its ‘counting’ state.

The operation ‘hcss_allocate_resource’ has already been modeled in phase 4, without regard to later integration. Therefore the call to ‘pr_count_two_phase_ended’ is not yet incorporated in the internal STD of ‘hcss_allocate_resource’. Here, in this paragraph, the call to ‘pr_count_two_phase_ended’ will be explicitly modeled into the internal STD of the operation.

In the next four figures first the internal STD of the operation ‘hcss_allocate_resource’ is shown with the ‘finishing state’-indicator. Then there follows a figure which looks ‘inside’ the ‘finishing’-state and which shows the call to the operation ‘pr_count_two_phase_ended’. Then the two subprocesses S1 and S2 with respect to the manager STD of the control object ‘project’ will be given.

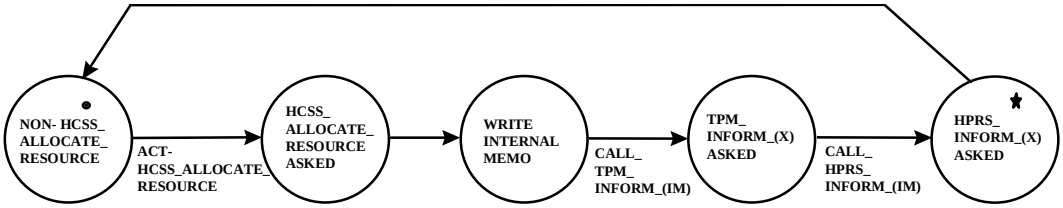


figure 6.37 employee int-hcss_allocate_resource : internal behavior STD, with 'finishing state'-indicator

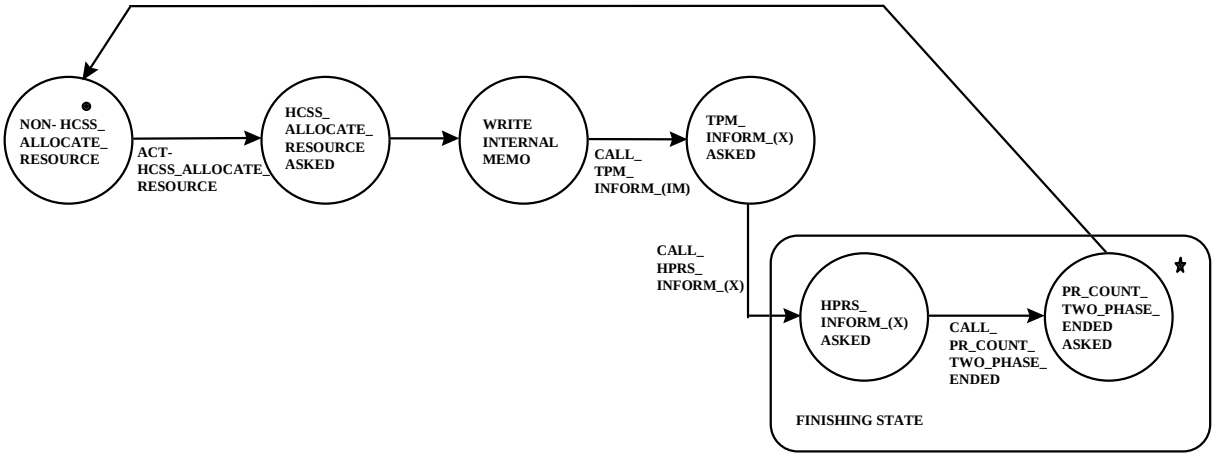


figure 6.38 employee int-hcss_allocate_resource : internal behavior STD, with exploded view of 'finishing state'

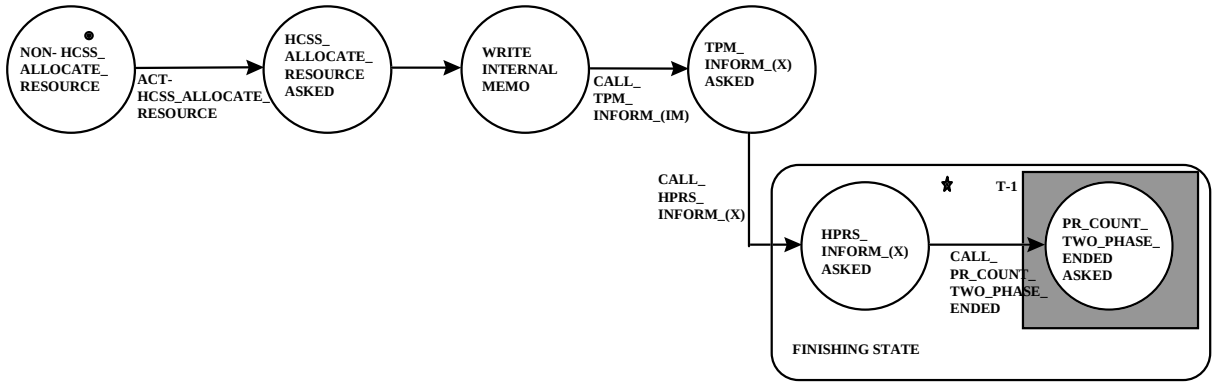


figure 6.39 employee int-hcss_allocate_resource : subprocess S1_wrt_project

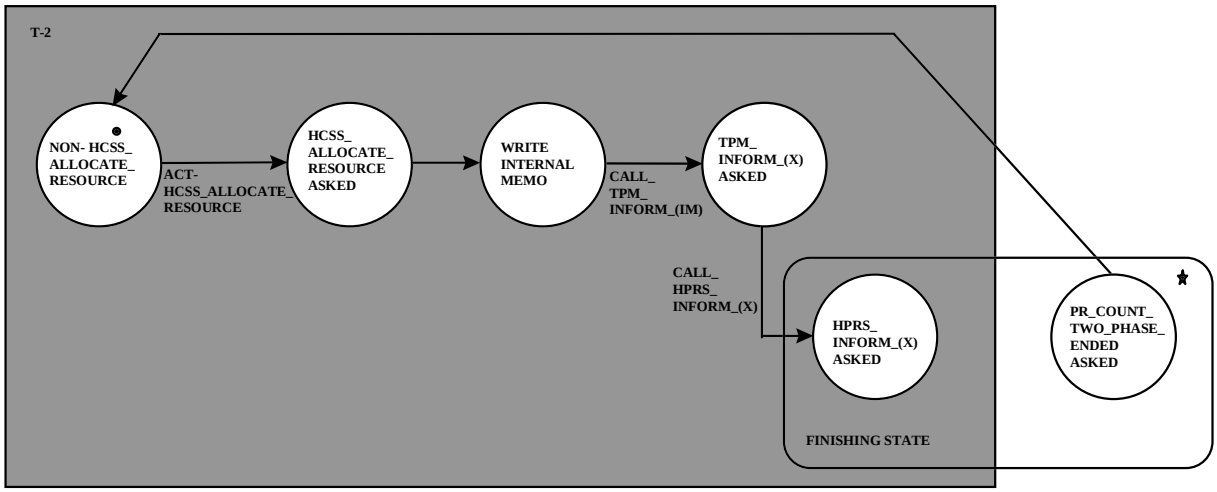


figure 6.40 employee int-hcss_allocate_resource : subprocess S2_wrt_project

6.3.2.2 Customer (phase 1, changed)

6.3.2.2.1 Customer (phase 1, changed) : external behavior-STD, organizational view

In the original sub-model of phase 1 it was assumed that the customer could start phase 1 (and therefore the total process fragment 'writing project management documents') repetitively by sequential calls to its autonomous operation 'cu_request_proposal'. In this way more than one request for proposal could be handled in parallel. In the integration of the four phase-submodels this parallel functionality is handled in another way. The customer now calls it autonomous operation 'cu_project_life_cycle' repetitively to start parallel executions of the process fragment. Within one execution of the process fragment 'writing project management documents' the customer can only once call its operation 'cu_request_proposal'. The external STD for phase 1 is changed to reflect this.

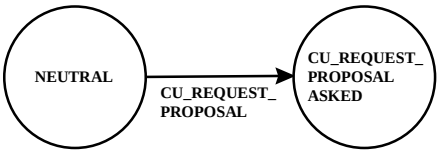


figure 6.41 customer : external behavior STD (phase 1) , organizational view

6.3.2.2.2 Customer (phase 1, changed) : external behavior-STD, communicative view

The changes in the external STD, organizational view, are reflected in the external STD, communicative view. Because the operation 'cu_request_proposal' can only be called once, the STD does not return to its neutral state after the call has been accepted. The 'starting' state, in which the called operation is started, still exists. After that, when the operation has been started, the STD goes to its (end)-state 'cu_request_proposal started'.

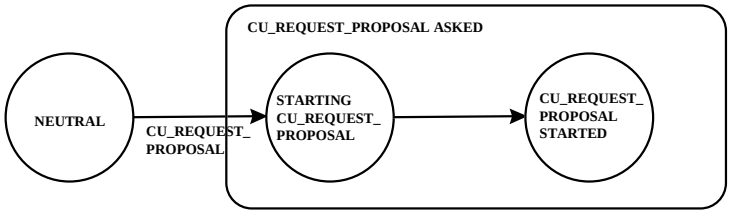


figure 6.42 customer : external behavior STD (phase 1), communicative view

6.3.2.2.3 Customer (phase 1, changed) : internal behavior-STDs

For phase 1 still only the operation 'cu_request_proposal' of the customer is relevant. Its internal behavior STD has not been changed with respect to the way it was modeled in the original phase 1-submodel.

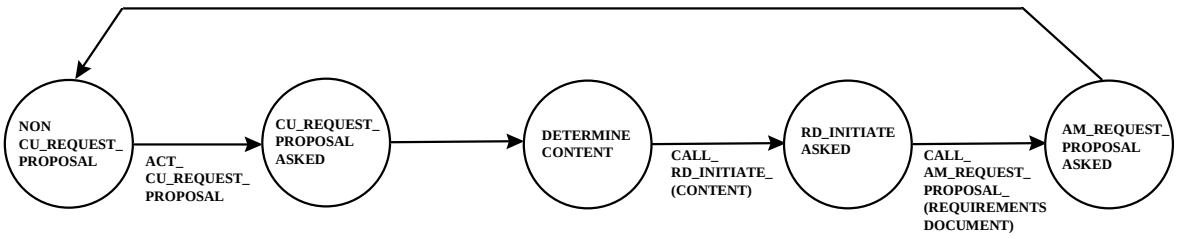


figure 6.43 int-cu_request_proposal : internal behavior STD

With the operation 'cu_request_proposal' the customer considers his requirements (= determines the contents of the requirements document). Then he creates a requirements document and initiates it with his requirements (call_ rd_initiate_(content)). With this requirements document he approaches the account manager with a request for a proposal based on the requirements (call_am_request_proposal_(requirements document)).

6.3.2.2.4 Customer (phase 1, changed) : manager-STD

The manager STD is the same as the external STD, communicative view. In it states the manager STD prescribes the subprocesses for its employees. The transitions are guarded with a combination of traps. The employees have to be in these traps for the transition to be enabled.

The notation CPSx stands for Consolidated Prescribed Subprocesses and is a set of CCx's. The notation CCx stands for 'Caller(s)-Callee' and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for 'Traps Logical Formula' and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

The CPSx's have the same name as the state for which they are valid. The TLF-x' have the same name as the operation that is started when the transition is taken. Transitions with no TLF (with no name in the external STD) are automatic transitions (unless otherwise indicated). The name of CCx's will include the full names of the caller and callee operations separated by a '~' character. Subprocesses and traps will be given as much of their full name as is necessary to avoid ambiguity. This full name is constructed with the 'dot'-notation.

For the (changed) 'phase 1'-manager STD of the class customer, the CPSs, CCs and the TLFS are :

CPS 'neutral' = {autonomous ~ cu_request_proposal.S1_wrt_customer}
TLF 'cu_request_proposal' = cu_request_proposal.T1_wrt_customer
CPS 'starting cu_request_proposal' = {autonomous ~ cu_request_proposal.S2_wrt_customer}
TLF 'no name' = cu_request_proposal.T2_wrt_customer
CPS 'cu_request_proposal started' = {autonomous ~ cu_request_proposal.S1_wrt_customer}

6.3.2.2.5 Customer (phase 1, changed) : employee-STDs

For phase 1 still only the employee 'cu_request_proposal' of the customer is relevant. The subprocesses S1 and S2, and two traps T-1 and T-2 are the same as modeled in the original 'phase 1'-submodel. S1, S2 ,T-1 and T-2 (all with respect to customer) are according to the caller-callee construct.

6.3.2.3 Customer (corporate)

6.3.2.3.1 Customer (corporate) : external behavior-STD, organizational view

The autonomous operation ‘cu_start_project_life_cycle’ is always available to the customer. It can always start a new project life cycle. This is modeled by adding to the total ‘constructed’ external STD of the customer (constructed during the integration of the process fragments ‘writing project management documents’, ‘changing project management documents’and ‘closing project’) a parallel STD that includes the transition ‘cu_project_life_cycle’. This constitutes the total ‘corporate’ external STD of the class customer.

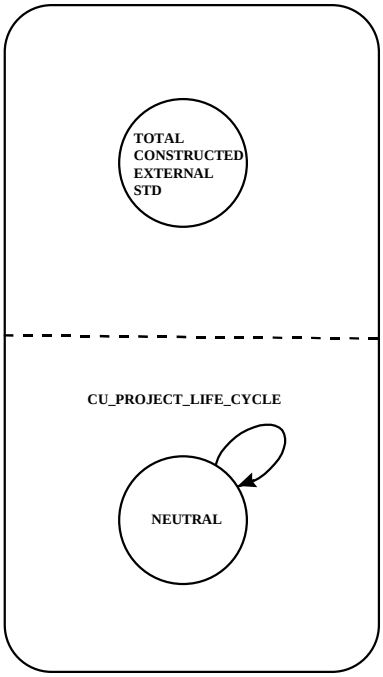


figure 6.44 customer : total corporate external STD, organizational view

The fact that both external STDs ‘total constructed external STD’ and ‘neutral’ can run in parallel, is modeled by the AND-superstate (the big rectangle with the rounded corners, with the dashed line dividing it into two compartments). The AND-superstate is part of the UML notation for state diagrams, which is essentially the Harel statechart notation [HAR].

6.3.2.3.2 Customer (corporate) : external behavior-STD, communicative view

The autonomous operation ‘cu_project_life_cycle’ can be called repetetively by the customer. The external STD has a ‘neutral’ state in which the STD waits for a call to the operation. The STD also has a ‘starting’ state. After the operation is started, the STD transits back to neutral. Here it can service the next call to ‘cu_project_life_cycle’. I.e. the customer can start the next project (life cycle).

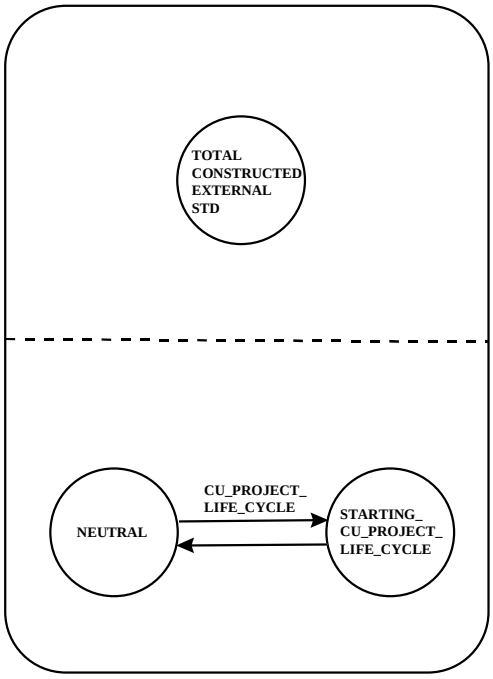


figure 6.45 customer : total corporate external STD, communicative view

6.3.2.3.3 Customer (corporate) : internal behavior-STDs

To the total corporate model of the customer the internal operation ‘cu_project_life_cycle’ is added. It has the following internal STD.

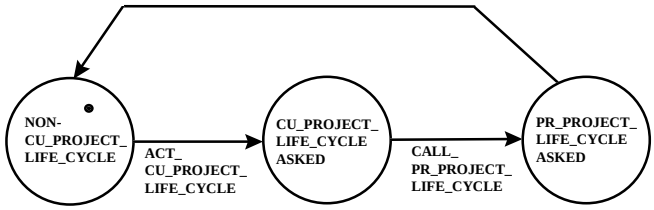


figure 6.46 int-cu_project_life_cycle : internal behavior STD

The operation simply calls ‘pr_project_life_cycle’ of the class project. With this operation the customer starts a new project (life cycle).

6.3.2.3.4 Customer (corporate) : manager-STD

The manager STD is the same as the external STD, communicative view. In it states the manager STD prescribes the subprocesses for its employees. The transitions are guarded with a combination of traps. The employees have to be in these traps for the transition to be enabled.

The notation CPSx stands for Consolidated Prescribed Subprocesses and is a set of CCx’s. The notation CCx stands for ‘Caller(s)-Callee’ and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for ‘Traps Logical Formula’ and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

For the ‘cu_project_life_cycle’-part of the total corporate external STD of the class customer, the CPSs, CCs and the TLFS are :

CPS ‘neutral’ = {autonomous ~ cu_project_life_cycle.S1_wrt_customer}

TLF ‘cu_project_life_cycle’ = cu_project_life_cycle.T1_wrt_customer
CPS ‘starting cu_project_life_cycle’ = {autonomous ~ cu_project_life_cycle.S2_wrt_customer}
TLF ‘no name’ = ‘cu_project_life_cycle.T2_wrt_customer

6.3.2.3.5 Customer (corporate) : employee-STDs

To the total corporate model of the customer the employee ‘cu_project_life_cycle’ is added. It has the subprocesses S1 and S2, and the traps T-1 and T-2 (all with respect to customer), according to the caller-callee construct.

6.3.2.4 Customer (integration)

6.3.2.4.1 Customer (integration) : extended phase-external behavior-STDs

The first step in the construction of the total external STD for the process fragment ‘writing project management documents’ is the extension of the constituent phase-external STDs with a start state and an end (final) state. The start state has a transition leaving it and entering the phase-external STD. The final state has a transition coming into it from the phase-external STD. This construction is performed in the next figures for the phase 1-external STD, the phase 2-external STD, the phase 3-external STD and the phase 4-external STD of the class ‘customer’.

The class ‘customer’ does not participate in phase 4. Therefore the phase 4-external STD consists of a dummy state. The phase 1-external STD has been changed with respect to the original sub-model of phase 1.

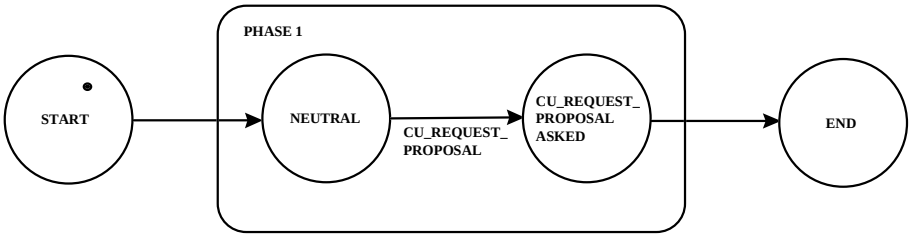


figure 6.47 customer : extended phase 1-external STD

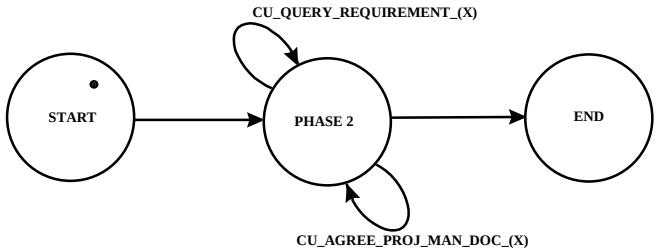


figure 6.48 customer : extended phase 2-external STD

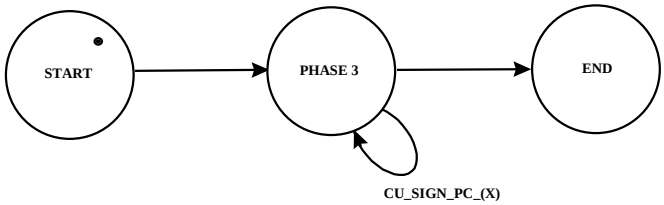


figure 6.49 customer : extended phase 3-external STD

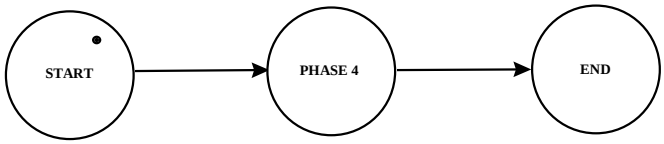


figure 6.50 customer : extended phase 4-external STD

6.3.2.4.2 Customer (integration) : total external behavior-STD

The second step in the construction of the total external STD for the process fragment ‘writing project management documents’ is to connect the extended phase-external STDs with each other in such a way that the final state of one phase-external STD coincides with the start state of the next phase-external STD. This ‘coinciding’ state between two phase-external STDs is called an ‘intermediate’ state. This models the sequential dependency between the phases.

The transitions leaving the start states get the label ‘change_to_phase_x’. These represent the ‘phase-changing’-operations. The transitions entering the final states get the label ‘pr_phase_ended’. These represent the ‘phase-ended’ operation. The construction is performed in the next figure.

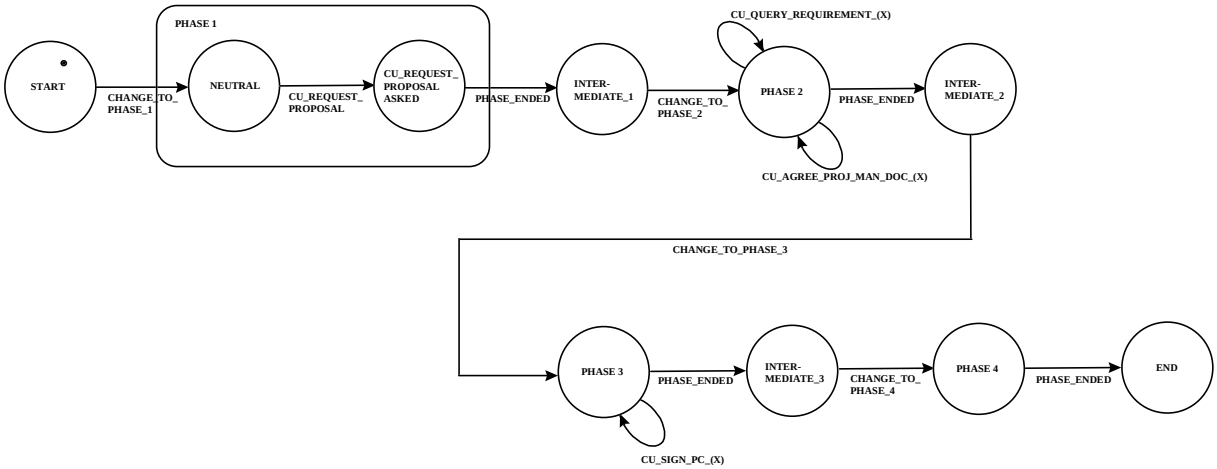


figure 6.51 customer : ext_wpmd_customer, total external STD

The naming convention for a total external STD is as follows. The name starts with the prefix ‘EXT’. This is conform the normal external STD notation. Then follows the indication of the process fragment for which the total external STD will be valid. In this case ‘WPMD’. This stands for ‘Writing Project Management Documents’. This is followed by the class name. E.g. EXT_WPMD_CUSTOMER is the external STD of the class ‘customer’ valid for the process fragment ‘writing project management documents’.

The total external STD is initially in its state ‘start’. When the control operation ‘pr_project_life_cyle’ of the control object ‘project’ (sim_)calls the operation ‘change_to_phase_1’ of the ‘customer’, the total external STD will (eventually) transit to its state ‘phase 1’. Here it manages its phase 1-employees. I.e. phase 1 takes place. At the end of phase 1, the last operation of this phase signals the control object that this phase can be ended. The operation ‘pr_phase_ended’ of the control object then (sim_)calls the operation ‘phase_ended’ of the ‘customer’. The total STD will (eventually) transit to the state ‘intermediate_1’. I.e. phase 1 has ended.

In the state ‘intermediate_1’ the total external STD is ready to accept a call to ‘change_to_phase_2’. When the control operation ‘pr_project_life_cycle’ (sim_)calls the operation ‘change_to_phase_2’ of the ‘customer’, the total external STD will (eventually) transit to its state ‘phase 2’. Here it manages its phase 2-employees. I.e. phase 2 takes place. At the end of phase 2, the last operation of this phase signals the control object that this phase can be ended. The operation ‘pr_phase_ended’ of the control object then (sim_)calls the operation ‘phase_ended’ of the ‘customer’. The total STD will (eventually) transit to the state ‘intermediate_2’. I.e. phase 2 has ended.

In the state ‘intermediate_2’ the total external STD is ready to accept a call to ‘change_to_phase_3’. When the control operation ‘pr_project_life_cycle’ (sim_)calls the operation ‘change_to_phase_3’ of the ‘customer’, the total external STD will (eventually) transit to its state ‘phase 3’. Here it manages its phase 3-employees. I.e. phase 3 takes place. At the end of phase 3, the last operation of this phase signals the control object that this phase can be ended. The operation ‘pr_phase_ended’ of the control object then (sim_)calls the operation ‘phase_ended’ of the ‘customer’. The total STD will (eventually) transit to the state ‘intermediate_3’. I.e. phase 3 has ended.

In the state ‘intermediate_3’ the total external STD is ready to accept a call to ‘change_to_phase_4’. When the control operation ‘pr_project_life_cycle’ (sim_)calls the operation ‘change_to_phase_4’ of the ‘customer’, the total external STD will (eventually) transit to its state ‘phase 4’. Since the class ‘customer’ does not participate in phase 4, no action of ‘customer’ will take place here. At the end of phase 4, both the ‘last’ operations of this phase signal the control object that this phase can be ended. The manager of the control object counts the number of calls to the operation ‘pr_count_two_phase_ended’. When this operation has been called twice, the manager STD of the control object autonomously starts its own operation ‘pr_phase_ended’. This operation ‘pr_phase_ended’ of the control object then (sim_)calls the operation ‘phase_ended’ of the ‘customer’. The total external STD will (eventually) transit to its final state ‘end’. I.e. phase 4 has ended. I.e. the total process fragment ‘writing project management documents’ has ended.

6.3.2.4.3 Customer (integration) : internal behavior-STDs

The internal operations of the class ‘customer’ are in the first place the internal operations as modeled in the phase 1, 2 and 3. Added to these operations is the operation ‘cu_project_life_cycle’ as modeled in the corporate model of the class ‘customer’.

During the integration the following internal operations are added to the class ‘customer’ : ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and ‘phase_ended’.

These 5 additional operations are so-called ‘no-operations’ (nops). A nop is an operation that performs no function while executing. The sole purpose of such an operation is to enable the corresponding transition of the external STD within the existing SOCCA framework.

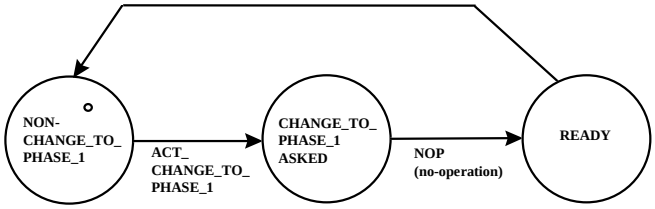


figure 6.52 int-change_to_phase_1 : internal behavior STD

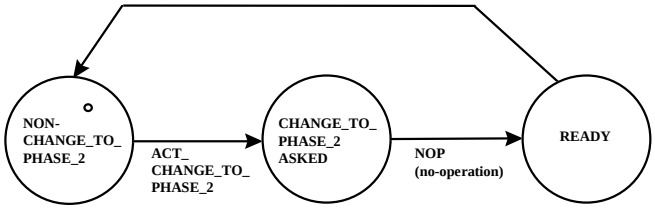


figure 6.53 int-change_to_phase_2 : internal behavior STD

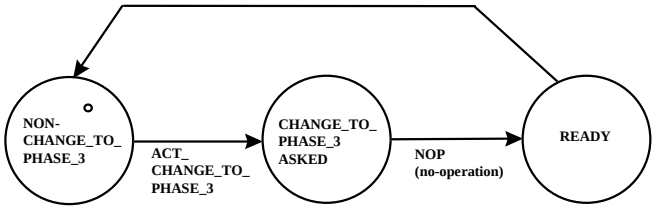


figure 6.54 int-change_to_phase_3 : internal behavior STD

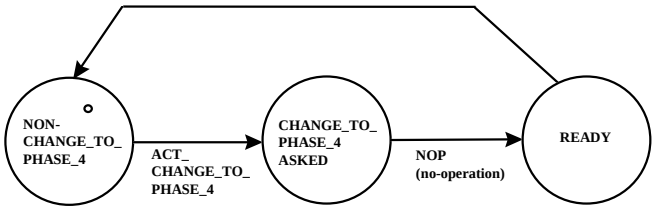


figure 6.55 int-change_to_phase_4 : internal behavior STD

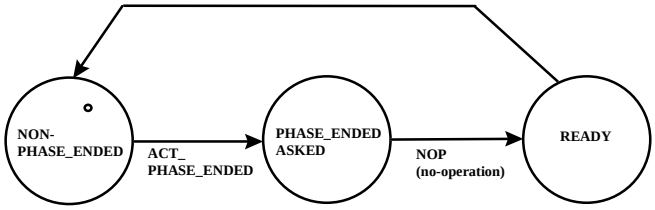


figure 6.56 int-phase_ended : internal behavior STD

6.3.2.4.4 Customer (integration) : manager-STD

The total manager STD is the same as the total external STD. Therefore no separate figure is needed for the total manager STD. The total manager STD of the process fragment 'writing project management documents' prescribes in its 'phase 1', 'phase 2' and 'phase 3' states still the same subprocesses for its phase-employees as was modeled in the phase 1-, phase 2- and phase 3-sub-models. These may be called 'old prescribed subprocesses'. Subprocess prescriptions for an employee in states (of the total manager STD) that are not relevant for that employee are 'don't cares'.

In the integration the total manager gets new employees. For this new employees it is also prescribing subprocesses. These may be called 'new prescribed subprocesses'. The sum total of the prescribed subprocesses by the total manager STD is therefore (per state) : {old prescribed subprocesses} ∪ {don't cares} ∪ {new prescribed subprocesses}.

In this paragraph only the {new prescribed subprocesses} will be specified. Also only the new traps will be specified. The 'old' traps as modeled in the 'phase'-sub-models are still valid in the integrated model.

The notation CPSx stands for Consolidated Prescribed Subprocesses and is a set of CCx's. The notation CCx stands for 'Caller(s)-Callee' and is the set of subprocesses prescribed for a certain caller-callee combination. The notation TLF-x stands for 'Traps Logical Formula' and describes the combination of traps that enables the transition. The numbering of the subprocesses and traps corresponds to the one used in the next paragraph.

The CPSx's have the same name as the state for which they are valid. The TLF-x' have the same name as the operation that is started when the transition is taken. Transitions with no TLF (with no name in the external STD) are automatic transitions (unless otherwise indicated in the definitions below). Traps that are not mentioned in the TLF are don't cares. The name of CCx's will include the full names of the caller and callee operations separated by a '~' character. Subprocesses and traps will be given as much of their full name as is necessary to avoid ambiguity. This full name is constructed with the 'dot'-notation.

The new CPSs, CCs and TLFs for this total manager STD are :

CPS 'start' =

{pr_project_life_cycle.S1_wrt_customer	~ change_to_phase_1.S1_wrt_customer,
	~ change_to_phase_2.S1_wrt_customer,
	~ change_to_phase_3.S1_wrt_customer,
	~ change_to_phase_4.S1_wrt_customer,
pr_phase_ended.S1_wrt_customer	~ phase_ended.S1_wrt_customer}

TLF 'change_to_phase_1' = pr_project_life_cycle.T1.wrt_customer and change_to_phase_1.T1_wrt_customer

CPS 'phase_1' =

{pr_project_life_cycle.S2_wrt_customer	~ change_to_phase_1.S2_wrt_customer,
	~ change_to_phase_2.S1_wrt_customer,
	~ change_to_phase_3.S1_wrt_customer,
	~ change_to_phase_4.S1_wrt_customer,
pr_phase_ended.S1_wrt_customer	~ phase_ended.S1_wrt_customer}

TLF 'phase_ended' (1^e) = pr_phase_ended.T1_wrt_customer and phase_ended.T1_wrt_customer

CPS 'intermediate_1' =

{pr_project_life_cycle.S2_wrt_customer	~ change_to_phase_1.S2_wrt_customer,
	~ change_to_phase_2.S1_wrt_customer,
	~ change_to_phase_3.S1_wrt_customer,
	~ change_to_phase_4.S1_wrt_customer,
pr_phase_ended.S2_wrt_customer	~ phase_ended.S2_wrt_customer}

TLF 'change_to_phase_2' = pr_project_life_cycle.T2.wrt_customer and change_to_phase_2.T1_wrt_customer
and pr_phase_ended.T2_wrt_customer and phase_ended.T2_wrt_customer

CPS ‘phase_2’ =
{pr_project_life_cycle.S3_wrt_customer ~ change_to_phase_1.S2_wrt_customer,
~ change_to_phase_2.S2_wrt_customer,
~ change_to_phase_3.S1_wrt_customer,
~ change_to_phase_4.S1_wrt_customer,
pr_phase_ended.S1_wrt_customer ~ phase_ended.S1_wrt_customer}

TLF ‘phase_ended’ (2^e) = pr_phase_ended.T1_wrt_customer and phase_ended.T1_wrt_customer

CPS ‘intermediate_2’ =
{pr_project_life_cycle.S3_wrt_customer ~ change_to_phase_1.S2_wrt_customer,
~ change_to_phase_2.S2_wrt_customer,
~ change_to_phase_3.S1_wrt_customer,
~ change_to_phase_4.S1_wrt_customer,
pr_phase_ended.S2_wrt_customer ~ phase_ended.S2_wrt_customer}

TLF ‘change_to_phase_3’ = pr_project_life_cycle.T3.wrt_customer and change_to_phase_3.T1_wrt_customer
and pr_phase_ended.T2_wrt_customer and phase_ended.T2_wrt_customer

CPS ‘phase_3’ =
{pr_project_life_cycle.S4_wrt_customer ~ change_to_phase_1.S2_wrt_customer,
~ change_to_phase_2.S2_wrt_customer,
~ change_to_phase_3.S2_wrt_customer,
~ change_to_phase_4.S1_wrt_customer,
pr_phase_ended.S1_wrt_customer ~ phase_ended.S1_wrt_customer}

TLF ‘phase_ended’ (3^e) = pr_phase_ended.T1_wrt_customer and phase_ended.T1_wrt_customer

CPS ‘intermediate_3’ =
{pr_project_life_cycle.S4_wrt_customer ~ change_to_phase_1.S2_wrt_customer,
~ change_to_phase_2.S2_wrt_customer,
~ change_to_phase_3.S2_wrt_customer,
~ change_to_phase_4.S1_wrt_customer,
pr_phase_ended.S2_wrt_customer ~ phase_ended.S2_wrt_customer}

TLF ‘change_to_phase_4’ = pr_project_life_cycle.T4.wrt_customer and change_to_phase_4.T1_wrt_customer
and pr_phase_ended.T2_wrt_customer and phase_ended.T2_wrt_customer

CPS ‘phase_4’ =
{pr_project_life_cycle.S5_wrt_customer ~ change_to_phase_1.S2_wrt_customer,
~ change_to_phase_2.S2_wrt_customer,
~ change_to_phase_3.S2_wrt_customer,
~ change_to_phase_4.S2_wrt_customer,
pr_phase_ended.S1_wrt_customer ~ phase_ended.S1_wrt_customer}

TLF ‘phase_ended’ (4^e) = pr_phase_ended.T1_wrt_customer and phase_ended.T1_wrt_customer
and pr_project_life_cycle.T5.wrt_customer
and change_to_phase_1.T2_wrt_customer
and change_to_phase_2.T2_wrt_customer
and change_to_phase_3.T2_wrt_customer
and change_to_phase_4.T2_wrt_customer

CPS ‘end’ =
{pr_project_life_cycle.S5_wrt_customer ~ change_to_phase_1.S1_wrt_customer,
~ change_to_phase_2.S1_wrt_customer,
~ change_to_phase_3.S1_wrt_customer,
~ change_to_phase_4.S1_wrt_customer,
pr_phase_ended.S2_wrt_customer ~ phase_ended.S2_wrt_customer}

Note : the operations ‘pr_phase_ended’ and ‘phase_ended’ are prescribed their subprocess S2 in the final state ‘end’. If there is a successor process fragment then the final (end) state of the total manager STD of the process fragment ‘writing project management documents’ is ‘merged’ with the start state of the total manager STD of the next process fragment. The guard on the transition out of this new intermediate state will be [pr_phase_ended.T2_wrt_customer and phase_ended.T2_wrt_customer] and the internal STDs will be switched back to their S1 subprocess. So they can be used again in the next process fragment.

6.3.2.4.5 Customer (integration) : employee-STDs

The employees of the manager STD of the class ‘customer’ are in the first place the employees as modeled in the phase 1, 2 and 3. Added to these is the employee ‘cu_project_life_cycle’ as modeled in the corporate model of the class ‘customer’.

During the integration the following 7 employees are added: ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and their caller ‘pr_project_life_cycle’ and ‘phase_ended’ and its caller ‘pr_phase_ended’.

The employee ‘change_to_phase_1’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to customer), according to the caller-callee construct.

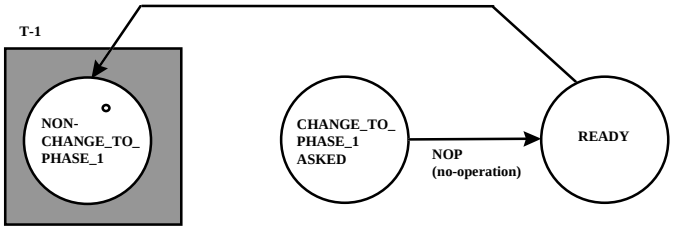


figure 6.57 employee int-change_to_phase_1 : subprocess S1_wrt_customer

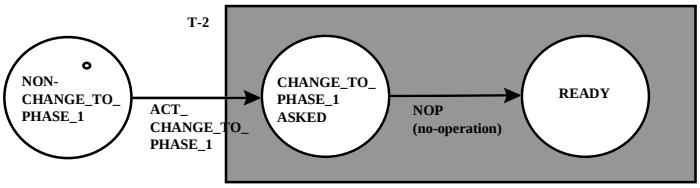


figure 6.58 employee int-change_to_phase_1 : subprocess S2_wrt_customer

The employee ‘change_to_phase_2’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to customer), according to the caller-callee construct.

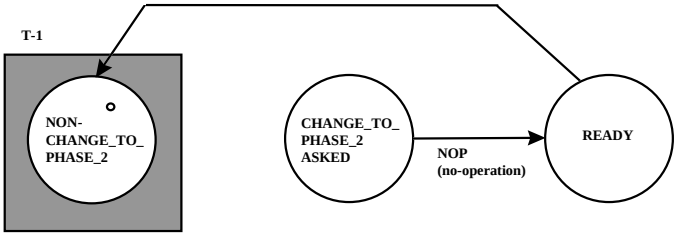


figure 6.59 employee int-change_to_phase_2 : subprocess S1_wrt_customer

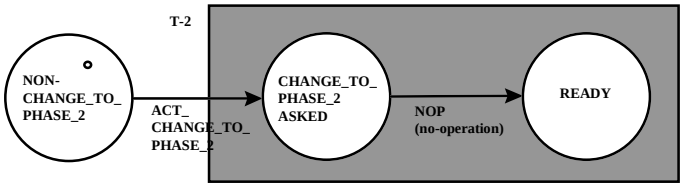


figure 6.60 employee int-change_to_phase_2 : subprocess S2_wrt_customer

The employee ‘change_to_phase_3’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to customer), according to the caller-callee construct.

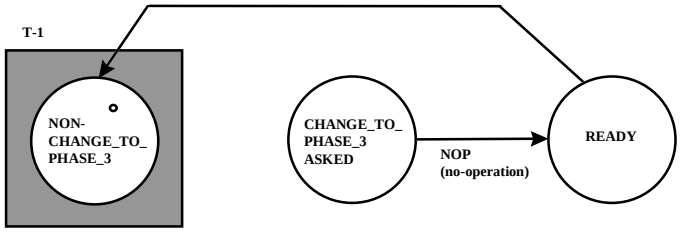


figure 6.61 employee int-change_to_phase_3 : subprocess S1_wrt_customer

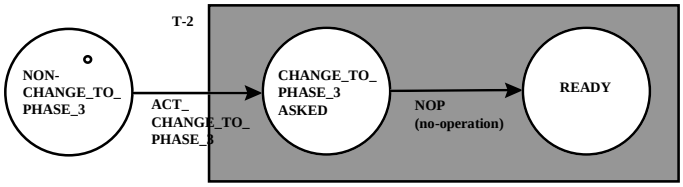


figure 6.62 employee int-change_to_phase_3 : subprocess S2_wrt_customer

The employee ‘change_to_phase_4’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to customer), according to the caller-callee construct.

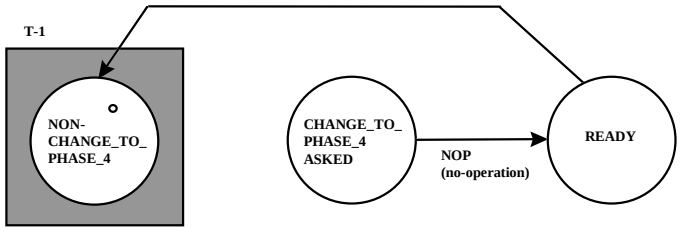


figure 6.63 employee int-change_to_phase_4 : subprocess S1_wrt_customer

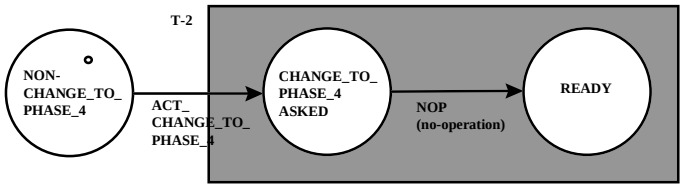


figure 6.64 employee int-change_to_phase_4 : subprocess S2_wrt_customer

The employee ‘pr_project_life_cycle’ is the caller of the four callees ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’ and ‘change_to_phase_4’ of the class ‘customer’. Consequently it has five subprocesses and traps with respect to the manager STD of the class ‘customer’ : S1, S2, S3, S4 and S5 and T-1, T-2, T-3, T-4 and T-5. The next two figures show the internal operation ‘pr_project_life_cycle’ and its five subprocesses.

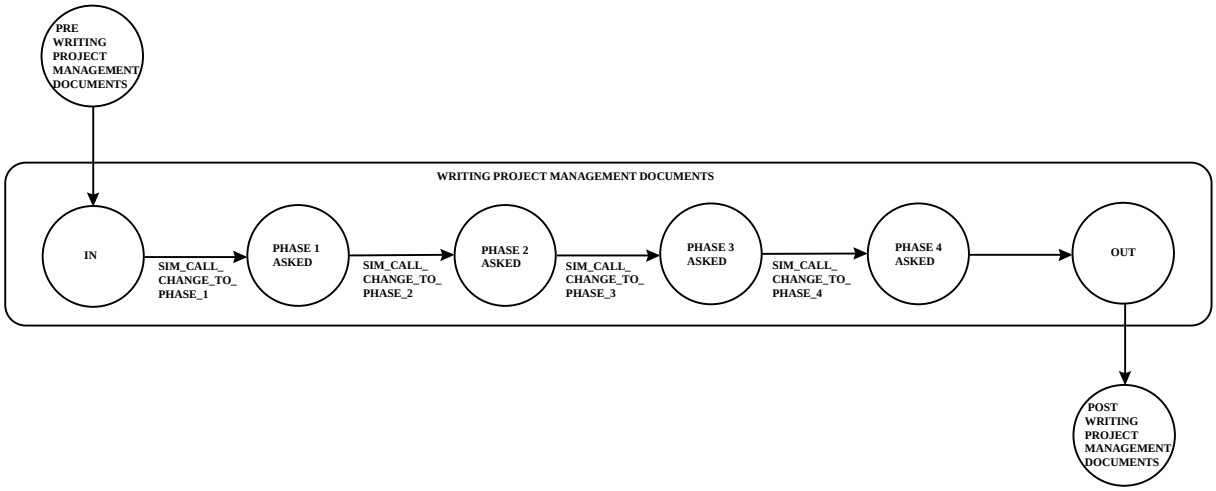


figure 6.65 employee int-pr_project_life_cycle : internal behavior STD

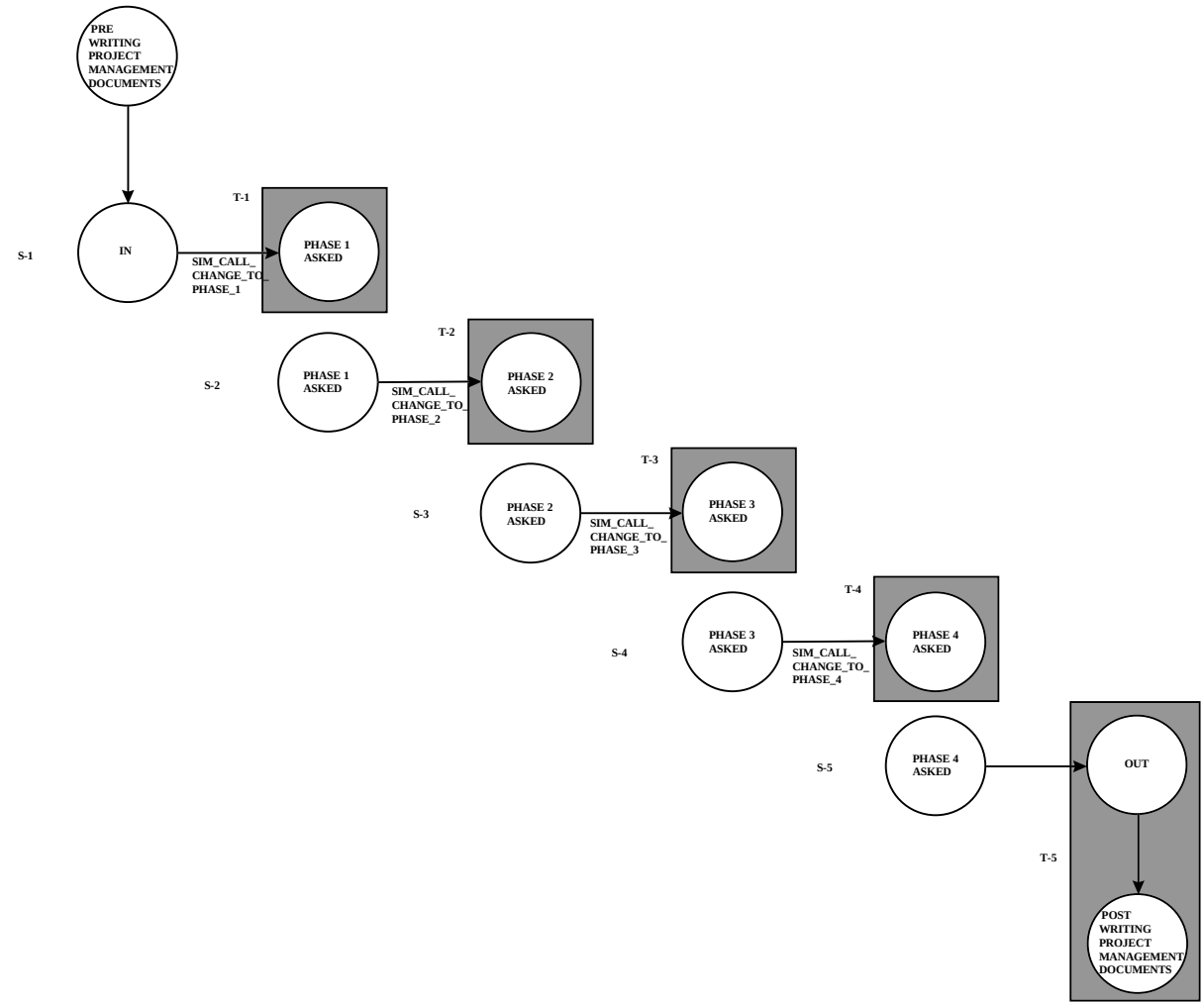


figure 6.66 employee int-pr_project_life_cycle : subprocesses S1, S2, S3, S4, S5 wrt_customer

This operation ‘pr_project_life_cycle’ is not only an employee of ‘customer’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1, S2, S3, S4 and S5. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses. At the start the operation has the prescribed subprocesses :

S1_wrt_customer,

S1_wrt_requirements_document,
S1_wrt_account_manager,
S1_wrt_make_or_buy_meeting,
S1_wrt_chief_executive_officer,
S1_wrt_head_personnel section,
S1_wrt_project_form,
S1_wrt_head_controller_section,
S1_wrt_technical_project_manager,
S1_wrt_quality_assurance_adviser,
S1_wrt_head_production_section,
S1_wrt_head_support_section,
S1_wrt_project_management_document,
S1_wrt_project_meeting_minus,
S1_wrt_archive/documentation_administrator,
S1_wrt_head_computer_support_section,
S1_wrt_terms_of_reference_document,
S1_wrt_internal_memorandum,
S1_wrt_engineer.

All these subprocesses are the same. The actual subprocess (the intersection) is thus S1. Then the operation places the sim_call_change_to_phase_1 to all 19 participating classes and it transits to its state ‘phase 1 asked’. The participating classes will service the call to their operation ’change_to_phase_1’. They do this not all at the same time. So some of the manager STDs of the participating classes will still be in their state where they prescribe S1, while others are already in a state where they prescribe S2. The actual subprocess of the operation ‘pr_project_life_cycle’ will then be the intersection of S1-subprocesses and S2-subprocesses. This results in an actual subprocess consisting only of the state ‘phase 1 asked’ for the operation ‘pr_project_life_cycle’. As the operation ‘pr_project_life_cycle’ has placed its sim_call_change_to_phase_1 it is already in this state. It wil now be confined to this state. When all 19 participating classes have serviced the call and are in their state where they prescribe S2, the actual subprocess of ‘pr_project_life_cycle’ will become S2. It can then place the sim_call_change_to_phase_2. This mechanism applies also to the other prescribed subprocesses. In this way the operation ‘pr_poject_life_cycle’ is ‘clocked’ through its states.

The employee ‘phase_ended’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to customer), according to the caller-callee construct.

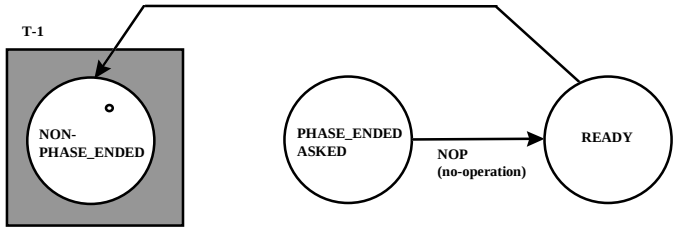


figure 6.67 employee int-phase_ended : subprocess S1_wrt_customer

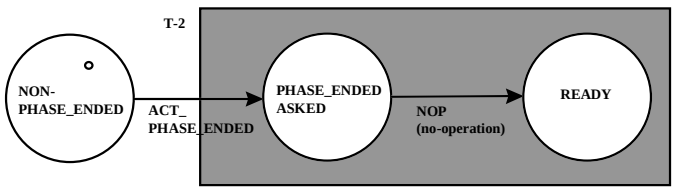


figure 6.68 employee int-phase_ended : subprocess S2_wrt_customer

The employee ‘pr_phase_ended’ is the caller of the callee ‘phase_ended’ of the class ‘customer’. It has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to customer) according to the caller_callee-construct.

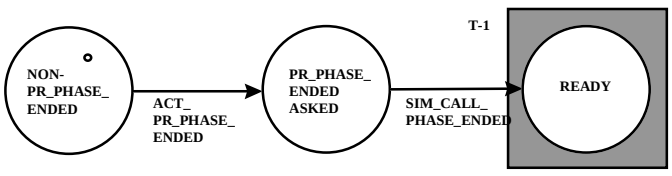


figure 6.69 employee int-pr_phase_ended : subprocess S1_wrt_customer

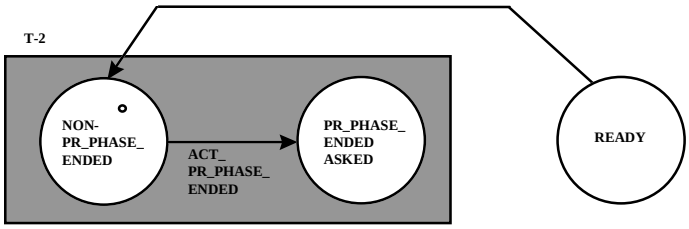


figure 6.70 employee int-pr_phase_ended : subprocess S2_wrt_customer

This operation ‘pr_phase_ended’ is not only an employee of ‘customer’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1 and S2. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses. Before the operation has placed its sim_call_phase_ended, it has prescribed to it the subprocesses S1_wrt_customer, S1_wrt_requirements_document, S1_wrt_account_manager, etc. The actual subprocess is thus S1. Then the operation places the sim_call_phase_ended to all 19 participating classes and it transits to its state ‘ready’. The participating classes will service the call to their operation ‘phase_ended’. They do this not all at the same time. So some of the manager STDs of the participating classes will still be in their state where they prescribe S1, while others are already in a state where they prescribe S2. The actual subprocess of the operation ‘pr_phase_ended’ will then be the intersection of S1-subprocesses and S2-subprocesses. This results in an actual subprocess consisting only of the state ‘ready’ for the operation ‘pr_phase_ ended’. As the operation ‘pr_phase_ended’ has placed its sim_call_phase_ended it is already in this state. It wil now be confined to this state. When all 19 participating classes have serviced the call and are in their state where they prescribe S2, the actual subprocess of ‘pr_phase_ended’ will become S2. The operation ‘pr_phase_ended’ can then continue with its next subprocess S2. This S2 being the ‘intermediate’ subprocess on the way to the next S1 in which it can place the next sim_call_phase_ended.

6.3.2.5 Requirements document (integration)

6.3.2.5.1 Requirements document (integration) : extended phase-external behavior-STDs

The first step in the construction of the total external STD for the process fragment ‘writing project management documents’ is the extension of the constituent phase-external STDs with a start state and an end (final) state. The start state has a transition leaving it and entering the phase-external STD. The final state has a transition coming into it from the phase-external STD. This construction is performed in the next figures for the phase 1-external STD, the phase 2-external STD, the phase 3-external STD and the phase 4-external STD of the class ‘requirements document’.

The class ‘requirements document’ does not participate in phase 2, 3 or 4. Therefore the phase 2-, phase 3- and phase 4-external STDs consist of a dummy state.

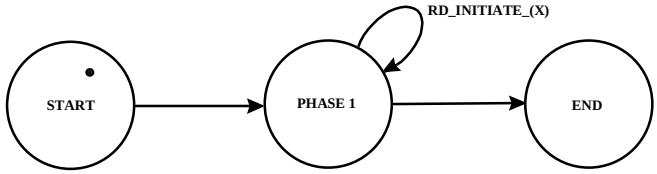


figure 6.71 requirements document : extended phase 1-external STD

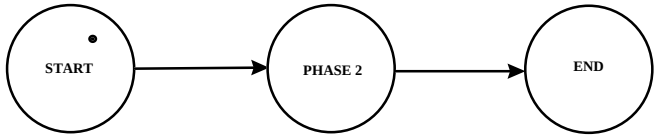


figure 6.72 requirements document : extended phase 2-external STD

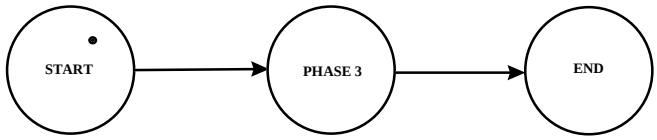


figure 6.73 requirements document : extended phase 3-external STD

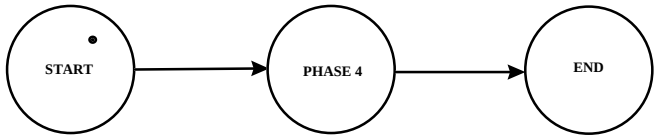


figure 6.74 requirements document : extended phase 4-external STD

6.3.2.5.2 Requirements document (integration) : total external behavior-STD

The second step in the construction of the total external STD for the process fragment ‘writing project management documents’ is to connect the extended phase-external STDs with each other in such a way that the final state of one phase-external STD coincides with the start state of the next phase-external STD. This ‘coinciding’ state between two phase-external STDs is called an ‘intermediate’ state. This models the sequential dependency between the phases.

The transitions leaving the start states get the label ‘change_to_phase_x’. These represent the ‘phase-changing’-operations. The transitions entering the final states get the label ‘pr_phase_ended’. These represent the ‘phase-ended’ operation. The construction is performed in the next figure.

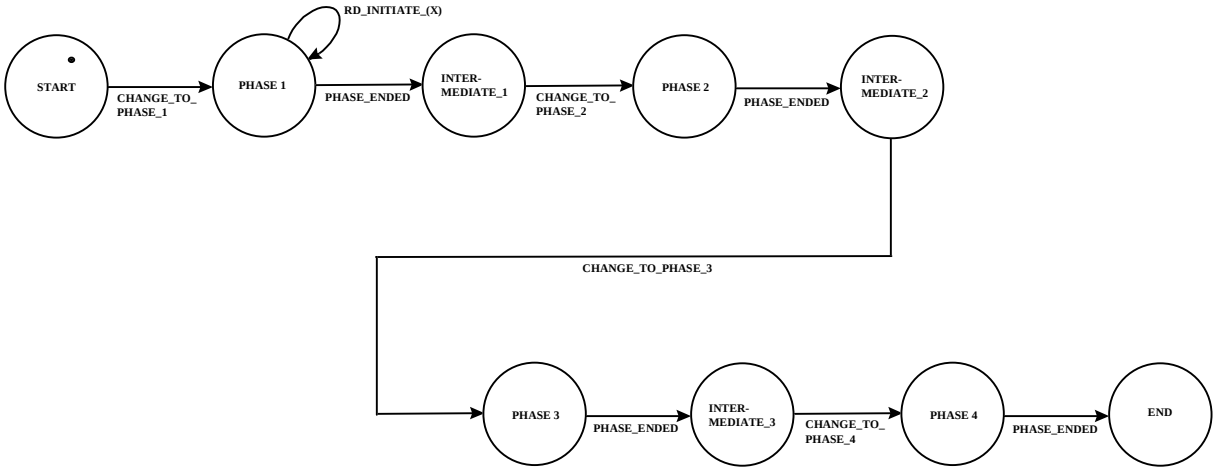


figure 6.75 requirements document : ext_wpmd_requirements_document, total external STD

The total external STD is initially in its state ‘start’. When the control operation ‘pr_project_life_cyle’ of the control object ‘project’ (sim_)calls the operation ‘change_to_phase_1’ of the ‘requirements document’, the total external STD will (eventually) transit to its state ‘phase 1’. Here it manages its phase 1-employees. I.e. phase 1 takes place. At the end of phase 1, the last operation of this phase signals the control object that this phase can be ended. The operation ‘pr_phase_ended’ of the control object then (sim_)calls the operation ‘phase_ended’ of the ‘requirements document’. The total STD will (eventually) transit to the state ‘intermediate_1’. I.e. phase 1 has ended.

In the state ‘intermediate_1’ the total external STD is ready to accept a call to ‘change_to_phase_2’. When the control operation ‘pr_project_life_cycle’ (sim_)calls the operation ‘change_to_phase_2’ of the ‘requirements document’, the total external STD will (eventually) transit to its state ‘phase 2’. Since the class ‘requirements document’ does not participate in phase 2, no action of ‘requirements document’ will take place here. At the end of phase 2, the last operation of this phase signals the control object that this phase can be ended. The operation ‘pr_phase_ended’ of the control object then (sim_)calls the operation ‘phase_ended’ of the ‘requirements document’. The total STD will (eventually) transit to the state ‘intermediate_2’. I.e. phase 2 has ended.

In the state ‘intermediate_2’ the total external STD is ready to accept a call to ‘change_to_phase_3’. When the control operation ‘pr_project_life_cycle’ (sim_)calls the operation ‘change_to_phase_3’ of the ‘requirements document’, the total external STD will (eventually) transit to its state ‘phase 3’. Since the class ‘requirements document’ does not participate in phase 3, no action of ‘requirements document’ will take place here. At the end of phase 3, the last operation of this phase signals the control object that this phase can be ended. The operation ‘pr_phase_ended’ of the control object then (sim_)calls the operation ‘phase_ended’ of the ‘requirements document’. The total STD will (eventually) transit to the state ‘intermediate_3’. I.e. phase 3 has ended.

In the state ‘intermediate_3’ the total external STD is ready to accept a call to ‘change_to_phase_4’. When the control operation ‘pr_project_life_cycle’ (sim_)calls the operation ‘change_to_phase_4’ of the ‘requirements document’, the total external STD will (eventually) transit to its state ‘phase 4’. Since the class ‘requirements document’ does not participate in phase 4, no action of ‘requirements document’ will take place here. At the end of phase 4, both the ‘last’ operations of this phase signal the control object that this phase can be ended. The manager of the control object counts the number of calls to the operation ‘pr_count_two_phase_ended’. When this operation has been called twice, the manager STD of the control object autonomously starts its own operation ‘pr_phase_ended’. This operation ‘pr_phase_ended’ of the control object then (sim_)calls the operation ‘phase_ended’ of the ‘requirements document’. The total external STD will (eventually) transit to its final state ‘end’. I.e. phase 4 has ended. I.e. the total process fragment ‘writing project management documents’ has ended.

6.3.2.5.3 Requirements document (integration) : internal behavior-STDs

The internal operations of the class ‘requirements document’ are in the first place the internal operation as modeled in the phase 1.

During the integration the following internal operations are added to the class ‘requirements document’ : ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and ‘phase_ended’.

These 5 additional operations are so-called 'no-operations' (nops). A nop is an operation that performs no function while executing. The sole purpose of such an operation is to enable the corresponding transition of the external STD within the existing SOCCA framework.

These internal operations have internal STDs equivalent to the corresponding operations of the class 'customer' (described in the paragraph 'Customer (integration) : internal behavior STDs').

6.3.2.5.4 Requirements document (integration) : manager-STD

The total manager STD is the same as the total external STD. Therefore no separate figure is needed for the total manager STD. The total manager STD of the process fragment 'writing project management documents' prescribes in its 'phase 1' state still the same subprocesses for its phase-employees as was modeled in the phase 1 sub-model. These may be called 'old prescribed subprocesses'. Subprocess prescriptions for an employee in states (of the total manager STD) that are not relevant for that employee are 'don't cares'.

In the integration the total manager gets new employees. For this new employees it is also prescribing subprocesses. These may be called 'new prescribed subprocesses'. The sum total of the prescribed subprocesses by the total manager STD is therefore (per state) : {old prescribed subprocesses} $\cup$ {don't cares} $\cup$ {new prescribed subprocesses}.

In this paragraph only the {new prescribed subprocesses} will be specified. Also only the new traps will be specified. The 'old' traps as modeled in the 'phase'-sub-models are still valid in the integrated model.

The new CPSs (Consolidated Prescribed Subprocesses), CCs (Caller_Callee combinations) and TLFs (Transition Logical Formulas) for this total manager STD are :

CPS 'start' =
{pr_project_life_cycle.S1_wrt_requirements_document ~ change_to_phase_1.S1_wrt_requirements_document,
~ change_to_phase_2.S1_wrt_requirements_document,
~ change_to_phase_3.S1_wrt_requirements_document,
~ change_to_phase_4.S1_wrt_requirements_document,
pr_phase_ended.S1_wrt_requirements_document ~ phase_ended.S1_wrt_requirements_document}

TLF 'change_to_phase_1' = pr_project_life_cycle.T1.wrt_requirements_document
and change_to_phase_1.T1_wrt_requirements_document

CPS 'phase_1' =
{pr_project_life_cycle.S2_wrt_requirements_document ~ change_to_phase_1.S2_wrt_requirements_document,
~ change_to_phase_2.S1_wrt_requirements_document,
~ change_to_phase_3.S1_wrt_requirements_document,
~ change_to_phase_4.S1_wrt_requirements_document,
pr_phase_ended.S1_wrt_requirements_document ~ phase_ended.S1_wrt_requirements_document}

TLF 'phase_ended' (1^e) = pr_phase_ended.T1_wrt_requirements_document
and phase_ended.T1_wrt_requirements_document

CPS 'intermediate_1' =
{pr_project_life_cycle.S2_wrt_requirements_document ~ change_to_phase_1.S2_wrt_requirements_document,
~ change_to_phase_2.S1_wrt_requirements_document,
~ change_to_phase_3.S1_wrt_requirements_document,
~ change_to_phase_4.S1_wrt_requirements_document,
pr_phase_ended.S2_wrt_requirements_document ~ phase_ended.S2_wrt_requirements_document}

TLF 'change_to_phase_2' = pr_project_life_cycle.T2.wrt_requirements_document
and change_to_phase_2.T1_wrt_requirements_document
and pr_phase_ended.T2_wrt_requirements_document
and phase_ended.T2_wrt_requirements_document

CPS 'phase_2' =

{pr_project_life_cycle.S3_wrt_requirements_document ~ change_to_phase_1.S2_wrt_requirements_document,
~ change_to_phase_2.S2_wrt_requirements_document,
~ change_to_phase_3.S1_wrt_requirements_document,
~ change_to_phase_4.S1_wrt_requirements_document,
pr_phase_ended.S1_wrt_requirements_document ~ phase_ended.S1_wrt_requirements_document}

TLF 'phase_ended' (2^e) = pr_phase_ended.T1_wrt_requirements_document
and phase_ended.T1_wrt_requirements_document

CPS 'intermediate_2' =
{pr_project_life_cycle.S3_wrt_requirements_document ~ change_to_phase_1.S2_wrt_requirements_document,
~ change_to_phase_2.S2_wrt_requirements_document,
~ change_to_phase_3.S1_wrt_requirements_document,
~ change_to_phase_4.S1_wrt_requirements_document,
pr_phase_ended.S2_wrt_requirements_document ~ phase_ended.S2_wrt_requirements_document}

TLF 'change_to_phase_3' = pr_project_life_cycle.T3.wrt_requirements_document
and change_to_phase_3.T1_wrt_requirements_document
and pr_phase_ended.T2_wrt_requirements_document
and phase_ended.T2_wrt_requirements_document

CPS 'phase_3' =
{pr_project_life_cycle.S4_wrt_requirements_document ~ change_to_phase_1.S2_wrt_requirements_document,
~ change_to_phase_2.S2_wrt_requirements_document,
~ change_to_phase_3.S2_wrt_requirements_document,
~ change_to_phase_4.S1_wrt_requirements_document,
pr_phase_ended.S1_wrt_requirements_document ~ phase_ended.S1_wrt_requirements_document}

TLF 'phase_ended' (3^e) = pr_phase_ended.T1_wrt_requirements_document
and phase_ended.T1_wrt_requirements_document

CPS 'intermediate_3' =
{pr_project_life_cycle.S4_wrt_requirements_document ~ change_to_phase_1.S2_wrt_requirements_document,
~ change_to_phase_2.S2_wrt_requirements_document,
~ change_to_phase_3.S2_wrt_requirements_document,
~ change_to_phase_4.S1_wrt_requirements_document,
pr_phase_ended.S2_wrt_requirements_document ~ phase_ended.S2_wrt_requirements_document}

TLF 'change_to_phase_4' = pr_project_life_cycle.T4.wrt_requirements_document
and change_to_phase_4.T1_wrt_requirements_document
and pr_phase_ended.T2_wrt_requirements_document
and phase_ended.T2_wrt_requirements_document

CPS 'phase_4' =
{pr_project_life_cycle.S5_wrt_requirements_document ~ change_to_phase_1.S2_wrt_requirements_document,
~ change_to_phase_2.S2_wrt_requirements_document,
~ change_to_phase_3.S2_wrt_requirements_document,
~ change_to_phase_4.S2_wrt_requirements_document,
pr_phase_ended.S1_wrt_requirements_document ~ phase_ended.S1_wrt_requirements_document}

TLF 'phase_ended' (4^e) = pr_phase_ended.T1_wrt_requirements_document
and phase_ended.T1_wrt_requirements_document
and pr_project_life_cycle.T5.wrt_requirements_document
and change_to_phase_1.T2_wrt_requirements_document
and change_to_phase_2.T2_wrt_requirements_document
and change_to_phase_3.T2_wrt_requirements_document
and change_to_phase_4.T2_wrt_requirements_document

CPS 'end' =
{pr_project_life_cycle.S5_wrt_requirements_document ~ change_to_phase_1.S1_wrt_requirements_document,
~ change_to_phase_2.S1_wrt_requirements_document,

	~ change_to_phase_3.S1_wrt_requirements_document,
	~ change_to_phase_4.S1_wrt_requirements_document,
pr_phase_ended.S2_wrt_requirements_document	~ phase_ended.S2_wrt_requirements_document }

Note : the operations ‘pr_phase_ended’ and ‘phase_ended’ are prescribed their subprocess S2 in the final state ‘end’. If there is a successor process fragment then the final (end) state of the total manager STD of the process fragment ‘writing project management documents’ is ‘merged’ with the start state of the total manager STD of the next process fragment. The guard on the transition out of this new intermediate state will be [pr_phase_ended.T2_wrt_requirements_ document and phase_ended.T2_wrt_ requirements_document] and the internal STDs will be switched back to their S1 subprocess. So they can be used again in the next process fragment.

6.3.2.5.5 Requirements document (integration) : employee-STDs

The employees of the manager STD of the class ‘requirements document’ are in the first place the employee as modeled in the phase 1. During the integration the following 7 employees are added: ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and their caller ‘pr_project_life_cycle’ and ‘phase_ended’ and its caller ‘pr_phase_ended’.

The employee ‘change_to_phase_1’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘requirements document’), according to the caller-callee construct.

The employee ‘change_to_phase_2’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘requirements document’), according to the caller-callee construct.

The employee ‘change_to_phase_3’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘requirements document’), according to the caller-callee construct.

The employee ‘change_to_phase_4’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘requirements document’), according to the caller-callee construct.

The employee ‘pr_project_life_cycle’ is the caller of the four callees ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’ and ‘change_to_phase_4’ of the class ‘requirements document’. Consequently it has five subprocesses and traps with respect to the manager STD of the class ‘requirements document’ : S1, S2, S3, S4 and S5 and T-1, T-2, T-3, T-4 and T-5. The next figure shows these five subprocesses of the internal operation ‘pr_project_ life_cycle’.

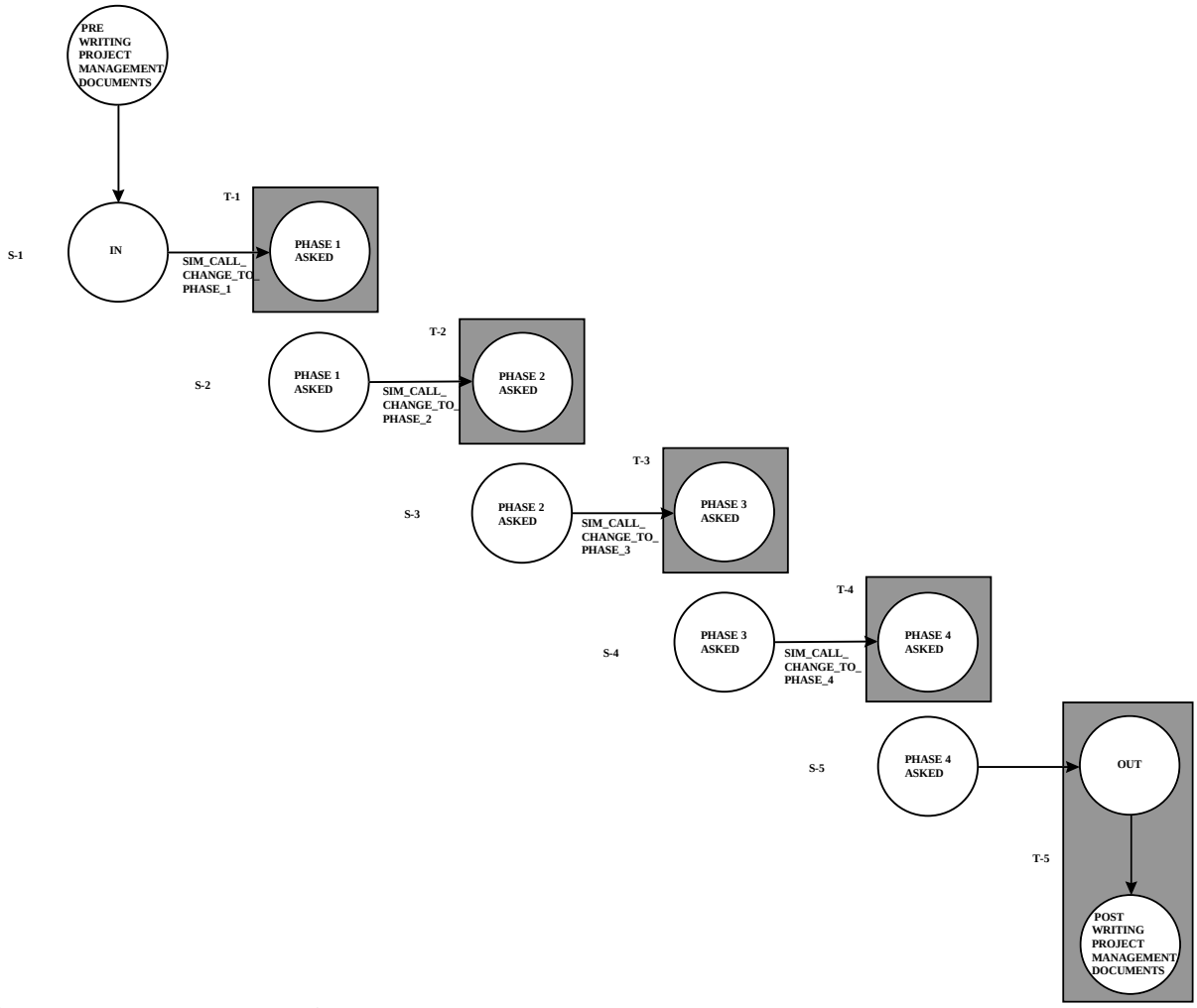


figure 6.76 employee int-pr_project_life_cycle : subprocesses S1, S2, S3, S4, S5 wrt_requirements_document

This operation ‘pr_project_life_cycle’ is not only an employee of ‘requirements document’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1, S2, S3, S4 and S5. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses. At the start the operation has the prescribed subprocesses :

S1_wrt_customer,
S1_wrt_requirements_document,
S1_wrt_account_manager,
S1_wrt_make_or_buy_meeting,
S1_wrt_chief_executive_officer,
S1_wrt_head_personnel section,
S1_wrt_project_form,
S1_wrt_head_controller_section,
S1_wrt_technical_project_manager,
S1_wrt_quality_assurance_adviser,
S1_wrt_head_production_section,
S1_wrt_head_support_section,
S1_wrt_project_management_document,
S1_wrt_project_meeting_minus,
S1_wrt_archive/documentation_administrator,
S1_wrt_head_computer_support_section,
S1_wrt_terms_of_reference_document,
S1_wrt_internal_memorandum,
S1_wrt_engineer.

All these subprocesses are the same. The actual subprocess (the intersection) is thus S1. Then the operation places the sim_call_change_to_phase_1 to all 19 participating classes and it transits to its state ‘phase 1 asked’. The participating classes will service the call to their operation ’change_to_phase_1’. They do this not all at the same time. So some of the manager STDs of the participating classes will still be in their state where they prescribe S1, while others are already in a state where they prescribe S2. The actual subprocess of the operation ‘pr_project_life_cycle’ will then be the intersection of S1-subprocesses and S2-subprocesses. This results in an actual subprocess consisting only of the state ‘phase 1 asked’ for the operation ‘pr_project_life_cycle’. As the operation ‘pr_project_life_cycle’ has placed its sim_call_change_to_phase_1 it is already in this state. It wil now be confined to this state. When all 19 participating classes have serviced the call and are in their state where they prescribe S2, the actual subprocess of ‘pr_project_life_cycle’ will become S2. It can then place the sim_call_change_to_phase_2. This mechanism applies also to the other prescribed subprocesses. In this way the operation ‘pr_poject_life_cycle’ is ‘clocked’ through its states.

The employee ‘phase_ended’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘requirements document’), according to the caller-callee construct.

The employee ‘pr_phase_ended’ is the caller of the callee ‘phase_ended’ of the class ‘requirements document’. It has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘requirements document’) according to the caller_callee-construct.

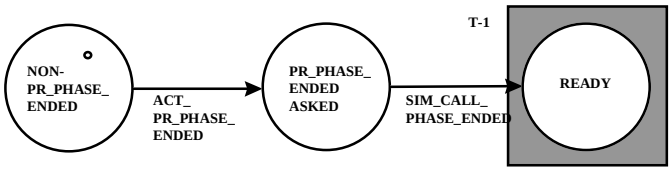


figure 6.77 employee int-pr_phase_ended : subprocess S1_wrt_requirements_document

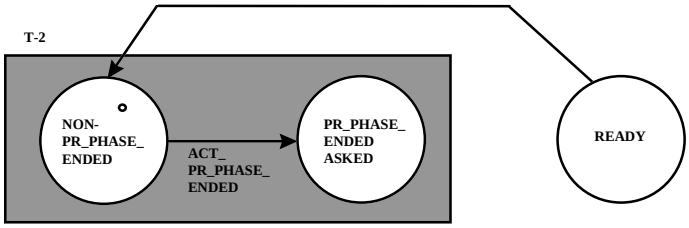


figure 6.78 employee int-pr_phase_ended : subprocess S2_wrt_requirements_document

This operation ‘pr_phase_ended’ is not only an employee of ‘requirements document’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1 and S2. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses. Before the operation has placed its sim_call_phase_ended, it has prescribed to it the subprocesses S1_wrt_customer, S1_wrt_requirements_document, S1_wrt_account_manager, etc. The actual subprocess is thus S1. Then the operation places the sim_call_phase_ended to all 19 participating classes and it transits to its state ‘ready’. The participating classes will service the call to their operation ‘phase_ended’. They do this not all at the same time. So some of the manager STDs of the participating classes will still be in their state where they prescribe S1, while others are already in a state where they prescribe S2. The actual subprocess of the operation ‘pr_phase_ended’ will then be the intersection of S1-subprocesses and S2-subprocesses. This results in an actual subprocess consisting only of the state ‘ready’ for the operation ‘pr_phase_ended’. As the operation ‘pr_phase_ended’ has placed its sim_call_phase_ended it is already in this state. It wil now be confined to this state. When all 19 participating classes have serviced the call and are in their state where they prescribe S2, the actual subprocess of ‘pr_phase_ended’ will become S2. The operation ‘pr_phase_ended’ can then continue with its next subprocess S2. This S2 being the ‘intermediate’ subprocess on the way to the next S1 in which it can place the next sim_call_phase_ended.

6.3.2.6 Account manager (integration)

6.3.2.6.1 Account manager (integration) : extended phase-external behavior-STDs

The first step in the construction of the total external STD for the process fragment ‘writing project management documents’ is the extension of the constituent phase-external STDs with a start state and an end (final) state. The start state has a transition leaving it and entering the phase-external STD. The final state has a transition coming into it from the phase-external STD. This construction is performed in the next figures for the phase 1-external STD, the phase 2-external STD, the phase 3-external STD and the phase 4-external STD of the class ‘account manager’.

The class ‘account manager’ does not participate in phase 4. Therefore the phase 4-external STD consists of a dummy state.

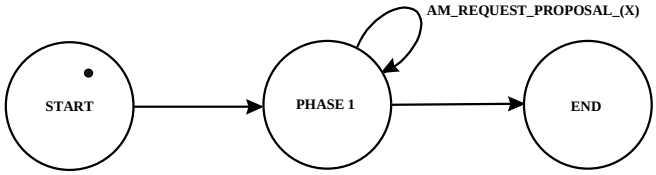


figure 6.79 account manager : extended phase 1-external STD

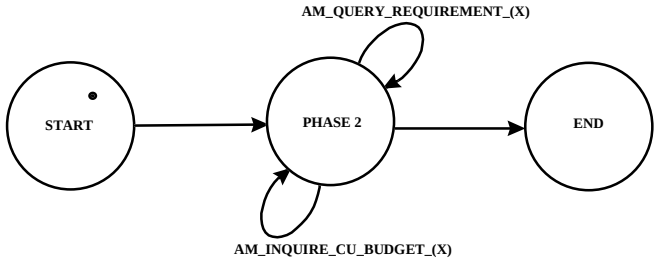


figure 6.80 account manager : extended phase 2-external STD

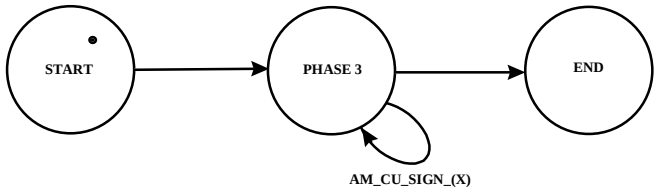


figure 6.81 account manager : extended phase 3-external STD

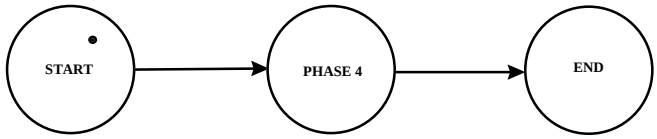


figure 6.82 account manager : extended phase 4-external STD

6.3.2.6.2 Account manager (integration) : total external behavior-STD

The second step in the construction of the total external STD for the process fragment ‘writing project management documents’ is to connect the extended phase-external STDs with each other in such a way that the final state of one phase-external STD coincides with the start state of the next phase-external STD. This ‘coinciding’ state between two phase-external STDs is called an ‘intermediate’ state. This models the sequential dependency between the phases.

The transitions leaving the start states get the label ‘change_to_phase_x’. These represent the ‘phase-changing’-operations. The transitions entering the final states get the label ‘pr_phase_ended’. These represent the ‘phase-ended’ operation. The construction is performed in the next figure.

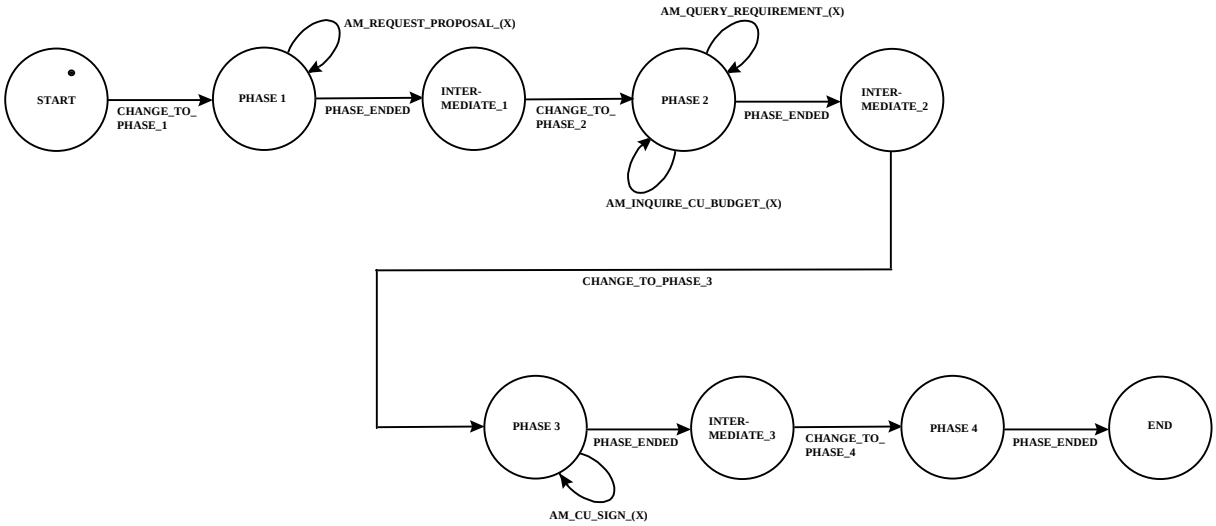


figure 6.83 account manager : ext_wpmd_account_manager, total external STD

The flow through this total external STD is analogous to the flow through the total external STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : total external behavior-STD).

With the difference that the class ‘account manager’ does not participate in phase 4. Therefore no action of ‘account manager’ will take place in that phase.

6.3.2.6.3 Account manager (integration) : internal behavior-STDs

The internal operations of the class ‘account manager’ are in the first place the internal operations as modeled in the phases 1, 2 and 3.

During the integration the following internal operations are added to the class: ‘change_to_phase_1’, ‘change_to_ phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and ‘phase_ended’.

These 5 additional operations are so-called ‘no-operations’ (nops). A nop is an operation that performs no function while executing. The sole purpose of such an operation is to enable the corresponding transition of the external STD within the existing SOCCA framework.

These internal operations have internal STDs equivalent to the corresponding operations of the class ‘customer’ (described in the paragraph ‘Customer (integration) : internal behavior STDs’).

6.3.2.6.4 Account manager (integration) : manager-STD

The total manager STD is the same as the total external STD. Therefore no separate figure is needed for the total manager STD. The total manager STD of the process fragment ‘writing project management documents’ prescribes in its ‘phase 1’, ‘phase 2’ and ‘phase 3’ states still the same subprocesses for its phase-employees as was modeled in the phase 1-, phase 2- and phase 3-sub-models. These may be called ‘old prescribed subprocesses’. Subprocess prescriptions for an employee in states (of the total manager STD) that are not relevant for that employee are ‘don’t cares’.

In the integration the total manager gets new employees. For this new employees it is also prescribing subprocesses. These may be called ‘new prescribed subprocesses’. The sum total of the prescribed subprocesses by the total manager STD is therefore (per state) : {old prescribed subprocesses} $\cup$ {don’t cares} $\cup$ {new prescribed subprocesses}.

The new CPSs (Consolidated Prescribed Subprocesses), CCs (Caller_Callee combinations) and TLFs (Transition Logical Formulas) for this total manager STD are the same as the CPSs, CCs and TLFs for the total manager STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : manager-STD).

With the difference that all the subprocesses and traps are with respect to ‘account manager’ instead of with respect to ‘requirements document’.

6.3.2.6.5 Account manager (integration) : employee-STDs

The employees of the manager STD of the class ‘account manager’ are in the first place the employees as modeled in the phase 1, 2 and 3. During the integration the following 7 employees are added: ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and their caller ‘pr_project_life_cycle’ and ‘phase_ended’ and its caller ‘pr_phase_ended’.

The employee ‘change_to_phase_1’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘account manager’), according to the caller-callee construct.

The employee ‘change_to_phase_2’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘account manager’), according to the caller-callee construct.

The employee ‘change_to_phase_3’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘account manager’), according to the caller-callee construct.

The employee ‘change_to_phase_4’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘account manager’), according to the caller-callee construct.

The employee ‘pr_project_life_cycle’ is the caller of the four callees ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’ and ‘change_to_phase_4’ of the class ‘account manager’. Consequently it has five subprocesses and traps with respect to the manager STD of the class ‘account manager’ : S1, S2, S3, S4 and S5 and T-1, T-2, T-3, T-4 and T-5. The next figure shows these five subprocesses of the internal operation ‘pr_project_ life_cycle’.

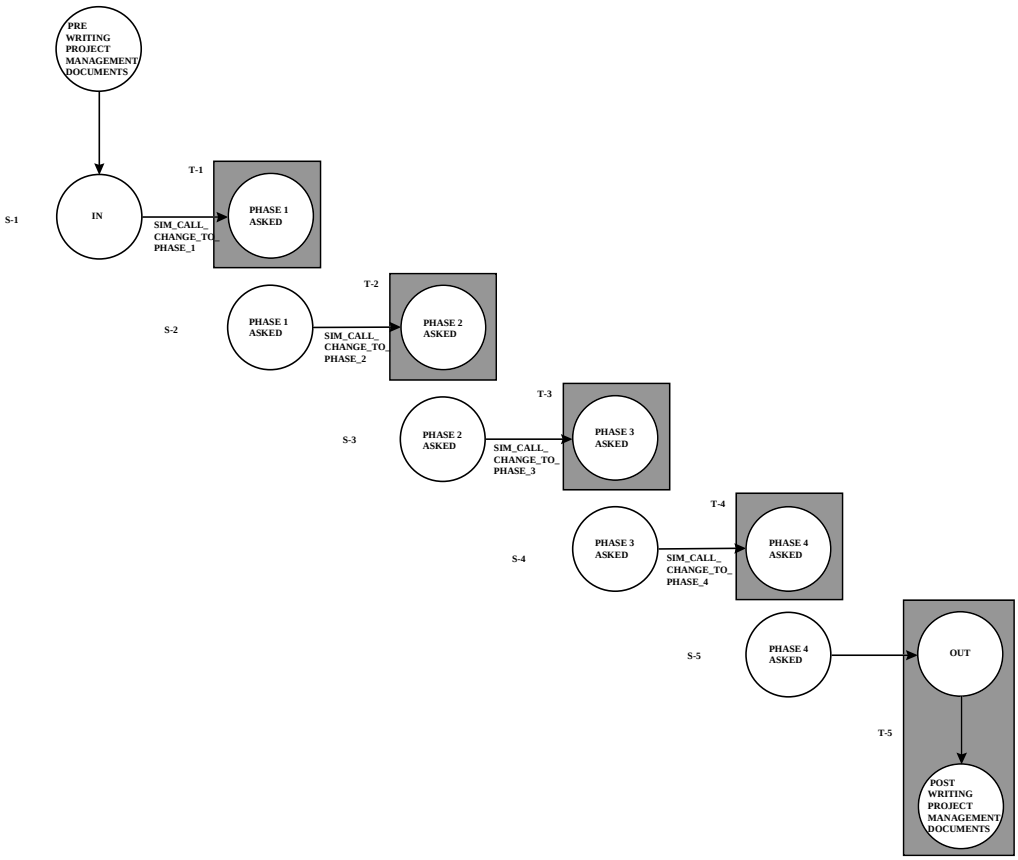


figure 6.84 employee int-pr_project_life_cycle : subprocesses S1, S2, S3, S4, S5 wrt_account_manager

This operation ‘pr_project_life_cycle’ is not only an employee of ‘account manager’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1, S2, S3, S4 and S5. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses. At the start the operation has the prescribed subprocesses S1_wrt_customer, S1_wrt_requirements_document, S1_wrt_account_manager, etc. The actual subprocess is thus S1. Then the operation places the sim_call_change_to_phase_1 to all 19 participating classes and it transits to its state ‘phase 1 asked’. The participating classes will service the call to their operation ’change_to_phase_1’. They do this not all at the same time. So some of the manager STDs of the participating classes will still be in their state where they prescribe S1, while others are already in a state where they prescribe S2. The actual subprocess of the operation ‘pr_project_life_cycle’ will then be the intersection of S1-subprocesses and S2-subprocesses. This results in an actual subprocess consisting only of the state ‘phase 1 asked’ for the operation ‘pr_project_life_cycle’. As the operation ‘pr_project_life_cycle’ has placed its sim_call_change_to_phase_1 it is already in this state. It wil now be confined to this state. When all 19 participating classes have serviced the call and are in their state where they prescribe S2, the actual subprocess of ‘pr_project_life_cycle’ will become S2. It can then place the sim_call_change_to_phase_2. This mechanism applies also to the other prescribed subprocesses. In this way the operation ‘pr_poject_life_cycle’ is ‘clocked’ through its states.

The employee ‘phase_ended’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘account manager’), according to the caller-callee construct. The employee ‘pr_phase_ended’ is the caller of the callee ‘phase_ended’ of the class ‘account manager’. It has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘account manager’) according to the caller_callee-construct.

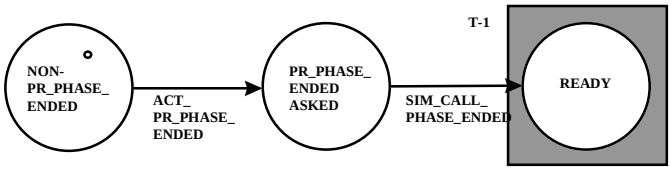


figure 6.85 employee int-pr_phase_ended : subprocess S1_wrt_account_manager

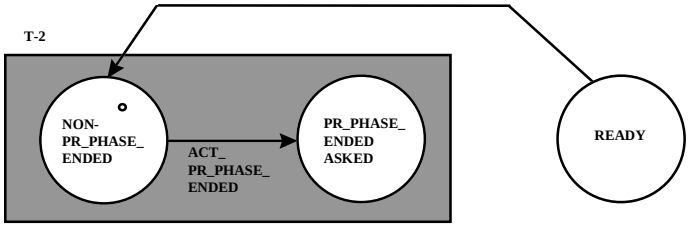


figure 6.86 employee int-pr_phase_ended : subprocess S2_wrt_account_manager

This operation ‘pr_phase_ended’ is not only an employee of ‘account manager’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1 and S2. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses. Before the operation has placed its sim_call_phase_ended, it has prescribed to it the subprocesses S1_wrt_customer, S1_wrt_requirements_document, S1_wrt_account_manager, etc. The actual subprocess is thus S1. Then the operation places the sim_call_phase_ended to all 19 participating classes and it transits to its state ‘ready’. The participating classes will service the call to their operation ‘phase_ended’. They do this not all at the same time. So some of the manager STDs of the participating classes will still be in their state where they prescribe S1, while others are already in a state where they prescribe S2. The actual subprocess of the operation ‘pr_phase_ended’ will then be the intersection of S1-subprocesses and S2-subprocesses. This results in an actual subprocess consisting only of the state ‘ready’ for the operation ‘pr_phase_ended’. As the operation ‘pr_phase_ended’ has placed its sim_call_phase_ended it is already in this state. It wil now be confined to this state. When all 19 participating classes have serviced the call and are in their state where they prescribe S2, the actual subprocess of ‘pr_phase_ended’ will become S2. The operation ‘pr_phase_ended’ can then continue with its next subprocess S2. This S2 being the ‘intermediate’ subprocess on the way to the next S1 in which it can place the next sim_call_phase_ended.

6.3.2.7 Make or buy-meeting (integration)

6.3.2.7.1 Make or buy-meeting (integration) : extended phase-external behavior-STDs

The first step in the construction of the total external STD for the process fragment ‘writing project management documents’ is the extension of the constituent phase-external STDs with a start state and an end (final) state. The start state has a transition leaving it and entering the phase-external STD. The final state has a transition coming into it from the phase-external STD. This construction is performed in the next figures for the phase 1-external STD, the phase 2-external STD, the phase 3-external STD and the phase 4-external STD of the class ‘make or buy-meeting’.

The class ‘make or buy meeting’ does not participate in phase 2, 3 or 4. Therefore the phase 2-, phase 3- and phase 4-external STDs consist of a dummy state.

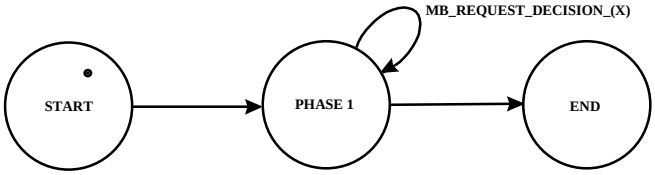


figure 6.87 make or buy-meeting : extended phase 1-external STD

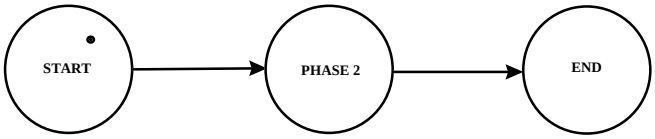


figure 6.88 make or buy-meeting : extended phase 2-external STD

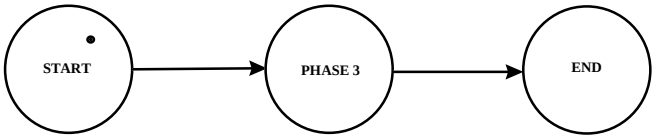


figure 6.89 make or buy-meeting : extended phase 3-external STD

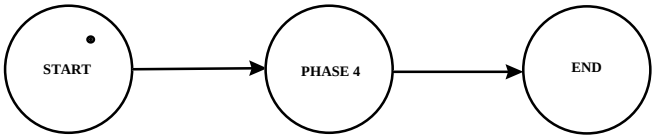


figure 6.90 make or buy-meeting : extended phase 4-external STD

6.3.2.7.2 Make or buy-meeting (integration) : total external behavior-STD

The second step in the construction of the total external STD for the process fragment ‘writing project management documents’ is to connect the extended phase-external STDs with each other in such a way that the final state of one phase-external STD coincides with the start state of the next phase-external STD. This ‘coinciding’ state between two phase-external STDs is called an ‘intermediate’ state. This models the sequential dependency between the phases.

The transitions leaving the start states get the label ‘change_to_phase_x’. These represent the ‘phase-changing’-operations. The transitions entering the final states get the label ‘pr_phase_ended’. These represent the ‘phase-ended’ operation. The construction is performed in the next figure.

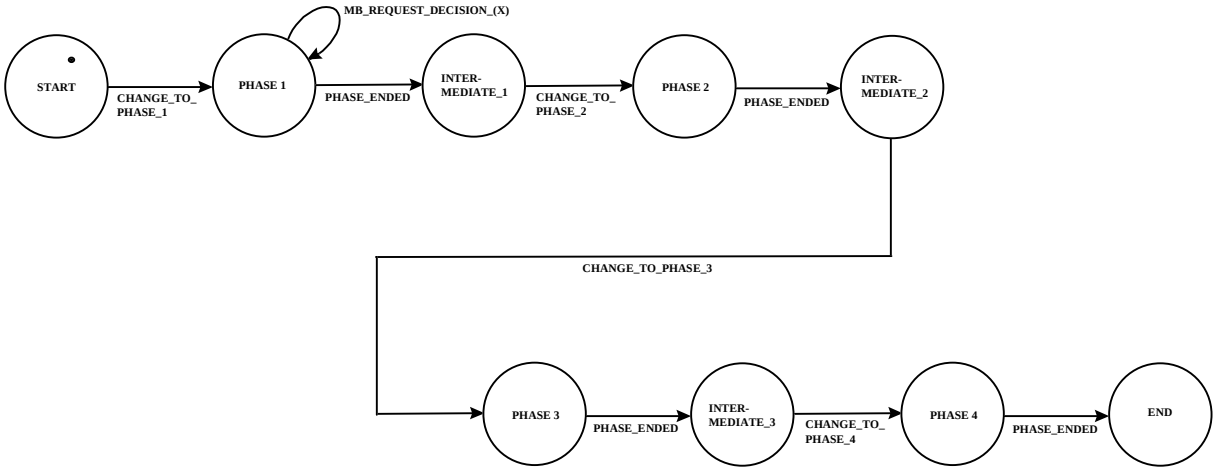


figure 6.91 make or buy-meeting : ext_wpmd_make_or_buy_meeting, total external STD

The flow through this total external STD is analogous to the flow through the total external STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : total external behavior-STD).

With the difference that the class ‘make or buy meeting’ does not participate in phase 2, 3 or 4. Therefore no action of ‘make or buy meeting’ will take place in these phases.

6.3.2.7.3 Make or buy-meeting (integration) : internal behavior-STDs

The internal operations of the class ‘make or buy meeting’ are in the first place the internal operation as modeled in the phase 1. During the integration the following internal operations are added to the class: ‘change_to_phase_1’, ‘change_to_ phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and ‘phase_ended’.

These 5 additional operations are so-called ‘no-operations’ (nops). A nop is an operation that performs no function while executing. The sole purpose of such an operation is to enable the corresponding transition of the external STD within the existing SOCCA framework.

These internal operations have internal STDs equivalent to the corresponding operations of the class ‘customer’ (described in the paragraph ‘Customer (integration) : internal behavior STDs’).

6.3.2.7.4 Make or buy-meeting (integration) : manager-STD

The total manager STD is the same as the total external STD. Therefore no separate figure is needed for the total manager STD. The total manager STD of the process fragment ‘writing project management documents’ prescribes in its ‘phase 1’ state still the same subprocesses for its phase-employees as was modeled in the phase 1-sub-model. These may be called ‘old prescribed subprocesses’. Subprocess prescriptions for an employee in states (of the total manager STD) that are not relevant for that employee are ‘don’t cares’.

In the integration the total manager gets new employees. For this new employees it is also prescribing subprocesses. These may be called ‘new prescribed subprocesses’. The sum total of the prescribed subprocesses by the total manager STD is therefore (per state) : {old prescribed subprocesses} $\cup$ {don’t cares} $\cup$ {new prescribed subprocesses}.

The new CPSs (Consolidated Prescribed Subprocesses), CCs (Caller_Callee combinations) and TLFs (Transition Logical Formulas) for this total manager STD are the same as the CPSs, CCs and TLFs for the total manager STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : manager-STD).

With the difference that all the subprocesses and traps are with respect to ‘make or buy meeting’ instead of with respect to ‘requirements document’.

6.3.2.7.5 Make or buy-meeting (integration) : employee-STDs

The employees of the manager STD of the class ‘make or buy meeting’ are in the first place the employees as modeled in the phase 1. During the integration the following 7 employees are added: ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and their caller ‘pr_project_life_cycle’ and ‘phase_ended’ and its caller ‘pr_phase_ended’.

The employee ‘change_to_phase_1’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘make or buy meeting’), according to the caller-callee construct.

The employee ‘change_to_phase_2’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘make or buy meeting’), according to the caller-callee construct.

The employee ‘change_to_phase_3’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘make or buy meeting’), according to the caller-callee construct.

The employee ‘change_to_phase_4’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘make or buy meeting’), according to the caller-callee construct.

The employee ‘pr_project_life_cycle’ is the caller of the four callees ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’ and ‘change_to_phase_4’ of the class ‘make or buy meeting’. Consequently it has five subprocesses and traps with respect to the manager STD of the class ‘make or buy meeting’ : S1, S2, S3, S4 and S5 and T-1, T-2, T-3, T-4 and T-5. The next figure shows these five subprocesses of the internal operation ‘pr_project_ life_cycle’.

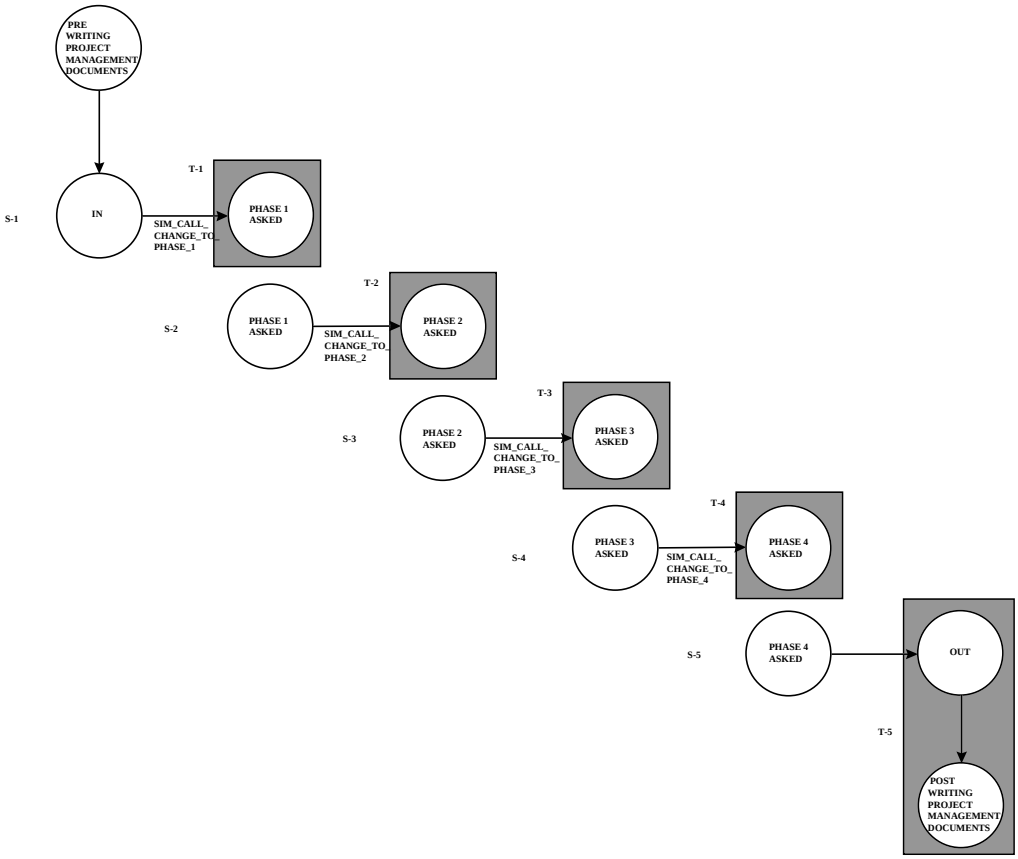


figure 6.92 employee int-pr_project_life_cycle : subprocesses S1, S2, S3, S4, S5 wrt_make_or_buy_meeting

This operation ‘pr_project_life_cycle’ is not only an employee of ‘make or buy meeting’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1, S2, S3, S4 and S5. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses. The actual subprocesses of this operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

The employee ‘phase_ended’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘make or buy meeting’), according to the caller-callee construct.

The employee ‘pr_phase_ended’ is the caller of the callee ‘phase_ended’ of the class ‘make or buy meeting’. It has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘make or buy meeting’) according to the caller_callee-construct.

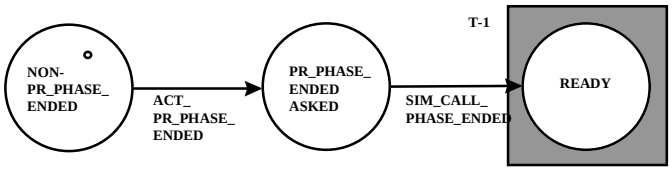


figure 6.93 employee int-pr_phase_ended : subprocess S1_wrt_make_or_buy_meeting

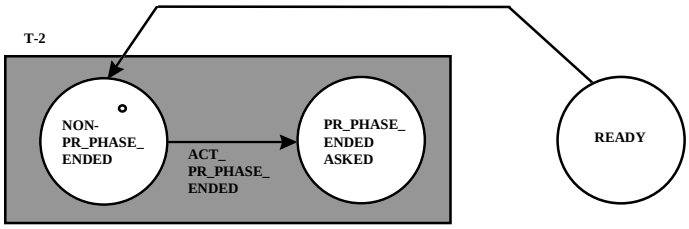


figure 6.94 employee int-pr_phase_ended : subprocess S2_wrt_make_or_buy_meeting

This operation ‘pr_phase_ended’ is not only an employee of ‘make or buy meeting’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1 and S2. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses.

The actual subprocesses of this operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

6.3.2.8 Chief executive officer (integration)

6.3.2.8.1 Chief executive officer (integration) : extended phase-external behavior-STDs

The first step in the construction of the total external STD for the process fragment ‘writing project management documents’ is the extension of the constituent phase-external STDs with a start state and an end (final) state. The start state has a transition leaving it and entering the phase-external STD. The final state has a transition coming into it from the phase-external STD. This construction is performed in the next figures for the phase 1-external STD, the phase 2-external STD, the phase 3-external STD and the phase 4-external STD of the class ‘chief executive officer’.

The class ‘chief executive officer’ does not participate in phase 2 or 4. Therefore the phase 2- and phase 4-external STDs consist of a dummy state.

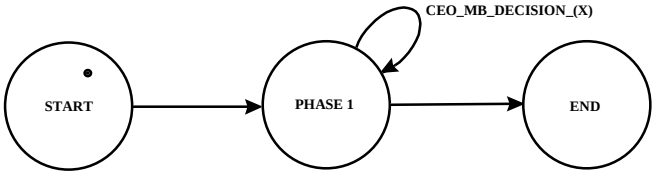


figure 6.95 chief executive officer : extended phase 1-external STD

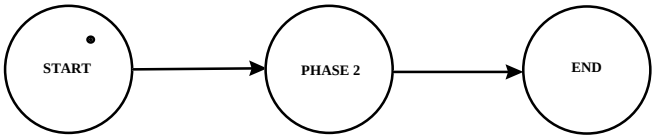


figure 6.96 chief executive officer : extended phase 2-external STD

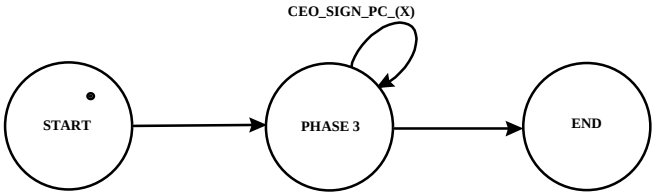


figure 6.97 chief executive officer : extended phase 3-external STD

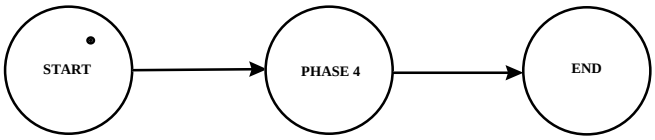


figure 6.98 chief executive officer : extended phase 4-external STD

6.3.2.8.2 Chief executive officer (integration) : total external behavior-STD

The second step in the construction of the total external STD for the process fragment ‘writing project management documents’ is to connect the extended phase-external STDs with each other in such a way that the final state of one phase-external STD coincides with the start state of the next phase-external STD. This ‘coinciding’ state between two phase-external STDs is called an ‘intermediate’ state. This models the sequential dependency between the phases.

The transitions leaving the start states get the label ‘change_to_phase_x’. These represent the ‘phase-changing’-operations. The transitions entering the final states get the label ‘pr_phase_ended’. These represent the ‘phase-ended’ operation. The construction is performed in the next figure.

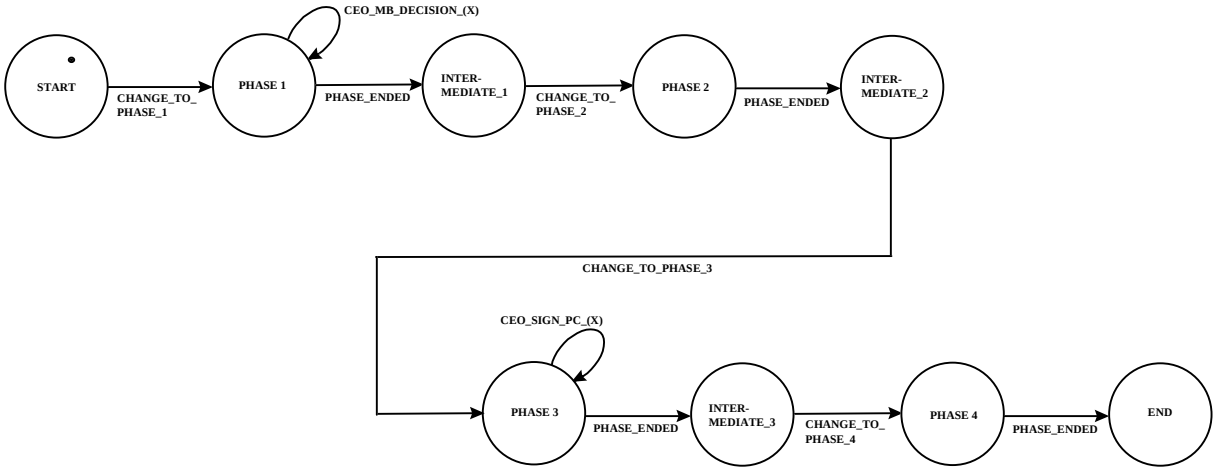


figure 6.99 chief executive officer : ext_wpmd_chief_executive_officer, total external STD

The flow through this total external STD is analogous to the flow through the total external STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : total external behavior-STD).

With the difference that the class ‘chief executive officer’ does not participate in phase 2 or 4. Therefore no action of ‘chief executive officer’ will take place in these phases.

6.3.2.8.3 Chief executive officer (integration) : internal behavior-STDs

The internal operations of the class ‘chief executive officer’ are in the first place the internal operations as modeled in the phase 1 and 3. During the integration the following internal operations are added to the class: ‘change_to_phase_1’, ‘change_to_ phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and ‘phase_ended’.

These 5 additional operations are so-called ‘no-operations’ (nops). A nop is an operation that performs no function while executing. The sole purpose of such an operation is to enable the corresponding transition of the external STD within the existing SOCCA framework.

These internal operations have internal STDs equivalent to the corresponding operations of the class ‘customer’ (described in the paragraph ‘Customer (integration) : internal behavior STDs’).

6.3.2.8.4 Chief executive officer (integration) : manager-STD

The total manager STD is the same as the total external STD. Therefore no separate figure is needed for the total manager STD. The total manager STD of the process fragment ‘writing project management documents’ prescribes in its ‘phase 1’ and ‘phase 3’ states still the same subprocesses for its phase-employees as was modeled in the phase 1- and phase 3-sub-model. These may be called ‘old prescribed subprocesses’. Subprocess prescriptions for an employee in states (of the total manager STD) that are not relevant for that employee are ‘don’t cares’.

In the integration the total manager gets new employees. For this new employees it is also prescribing subprocesses. These may be called ‘new prescribed subprocesses’. The sum total of the prescribed subprocesses by the total manager STD is therefore (per state) : {old prescribed subprocesses} $\cup$ {don’t cares} $\cup$ {new prescribed subprocesses}.

The new CPSs (Consolidated Prescribed Subprocesses), CCs (Caller_Callee combinations) and TLFs (Transition Logical Formulas) for this total manager STD are the same as the CPSs, CCs and TLFs for the total manager STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : manager-STD).

With the difference that all the subprocesses and traps are with respect to ‘chief executive officer’ instead of with respect to ‘requirements document’.

6.3.2.8.5 Chief executive officer (integration) : employee-STDs

The employees of the manager STD of the class ‘chief executive officer’ are in the first place the employees as modeled in the phase 1 and 3. During the integration the following 7 employees are added: ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and their caller ‘pr_project_life_cycle’ and ‘phase_ended’ and its caller ‘pr_phase_ended’.

The employee ‘change_to_phase_1’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘chief executive officer’), according to the caller-callee construct.

The employee ‘change_to_phase_2’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘chief executive officer’), according to the caller-callee construct.

The employee ‘change_to_phase_3’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘chief executive officer’), according to the caller-callee construct.

The employee ‘change_to_phase_4’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘chief executive officer’), according to the caller-callee construct.

The employee ‘pr_project_life_cycle’ is the caller of the four callees ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’ and ‘change_to_phase_4’ of the class ‘chief executive officer’. Consequently it has five subprocesses and traps with respect to the manager STD of the class ‘chief executive officer’ : S1, S2, S3, S4 and S5 and T-1, T-2, T-3, T-4 and T-5. The next figure shows these five subprocesses of the internal operation ‘pr_project_ life_cycle’.

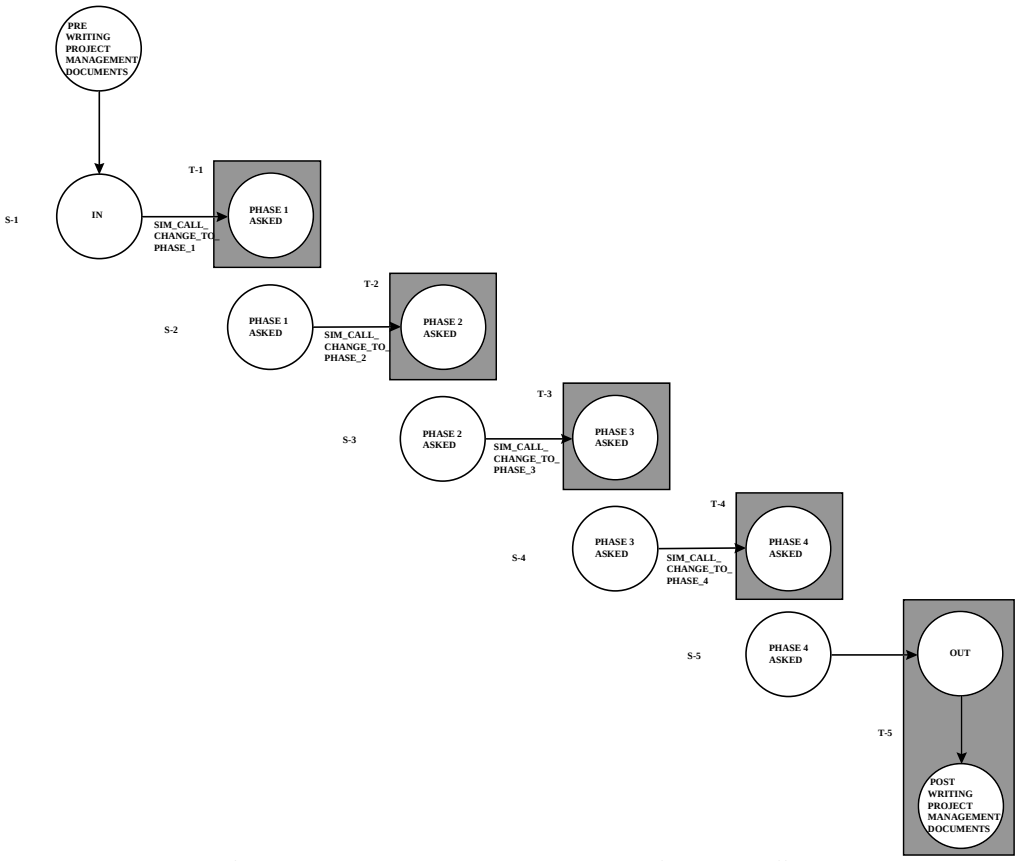


figure 6.100 employee int-pr_project_life_cycle : subprocesses S1, S2, S3, S4, S5 wrt_chief_executive_officer

This operation ‘pr_project_life_cycle’ is not only an employee of ‘chief executive officer’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1, S2, S3, S4 and S5. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses. The actual subprocesses of this

operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

The employee ‘phase_ended’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘chief executive officer’), according to the caller-callee construct.

The employee ‘pr_phase_ended’ is the caller of the callee ‘phase_ended’ of the class ‘chief executive officer’. It has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘chief executive officer’) according to the caller_callee-construct.

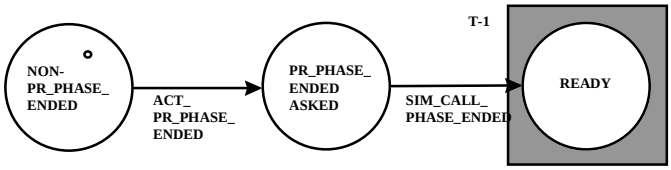


figure 6.101 employee int-pr_phase_ended : subprocess S1_wrt_chief_executive_officer

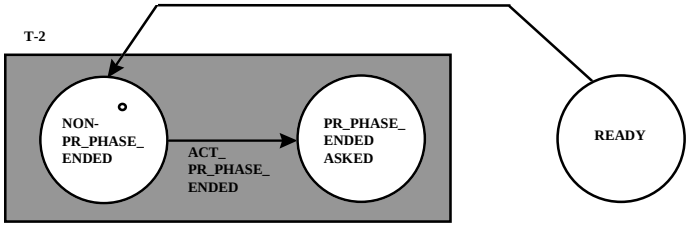


figure 6.102 employee int-pr_phase_ended : subprocess S2_wrt_chief_executive_officer

This operation ‘pr_phase_ended’ is not only an employee of ‘chief executive officer’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1 and S2. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses.

The actual subprocesses of this operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

6.3.2.9 Head personnel section (integration)

6.3.2.9.1 Head personnel section (integration) : extended phase-external behavior-STDs

The first step in the construction of the total external STD for the process fragment ‘writing project management documents’ is the extension of the constituent phase-external STDs with a start state and an end (final) state. The start state has a transition leaving it and entering the phase-external STD. The final state has a transition coming into it from the phase-external STD. This construction is performed in the next figures for the phase 1-external STD, the phase 2-external STD, the phase 3-external STD and the phase 4-external STD of the class ‘head personnel section’.

The class ‘head personnel section’ does not participate in phase 2 or 3. Therefore the phase 2- and phase 3-external STDs consist of a dummy state.

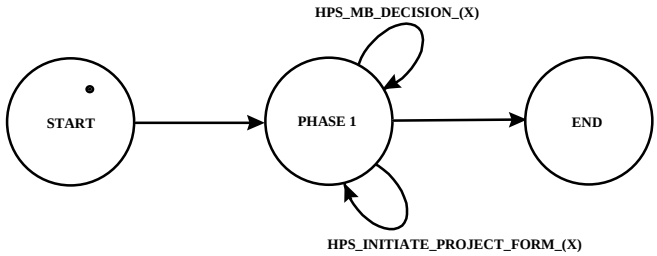


figure 6.103 head personnel section : extended phase 1-external STD

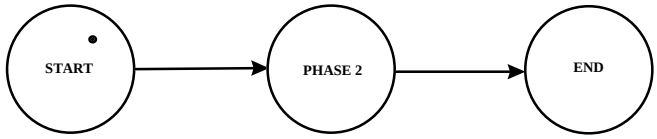


figure 6.104 head personnel section : extended phase 2-external STD

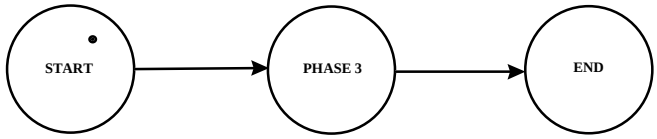


figure 6.105 head personnel section : extended phase 3-external STD

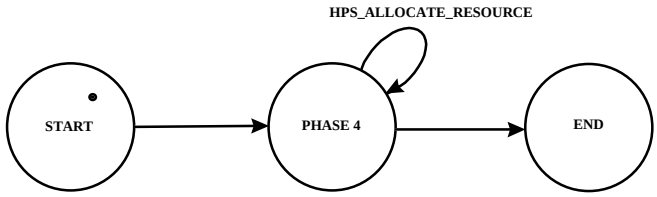


figure 6.106 head personnel section : extended phase 4-external STD

6.3.2.9.2 Head personnel section (integration) : total external behavior-STD

The second step in the construction of the total external STD for the process fragment ‘writing project management documents’ is to connect the extended phase-external STDs with each other in such a way that the final state of one phase-external STD coincides with the start state of the next phase-external STD. This ‘coinciding’ state between two phase-external STDs is called an ‘intermediate’ state. This models the sequential dependency between the phases.

The transitions leaving the start states get the label ‘change_to_phase_x’. These represent the ‘phase-changing’-operations. The transitions entering the final states get the label ‘pr_phase_ended’. These represent the ‘phase-ended’ operation. The construction is performed in the next figure.

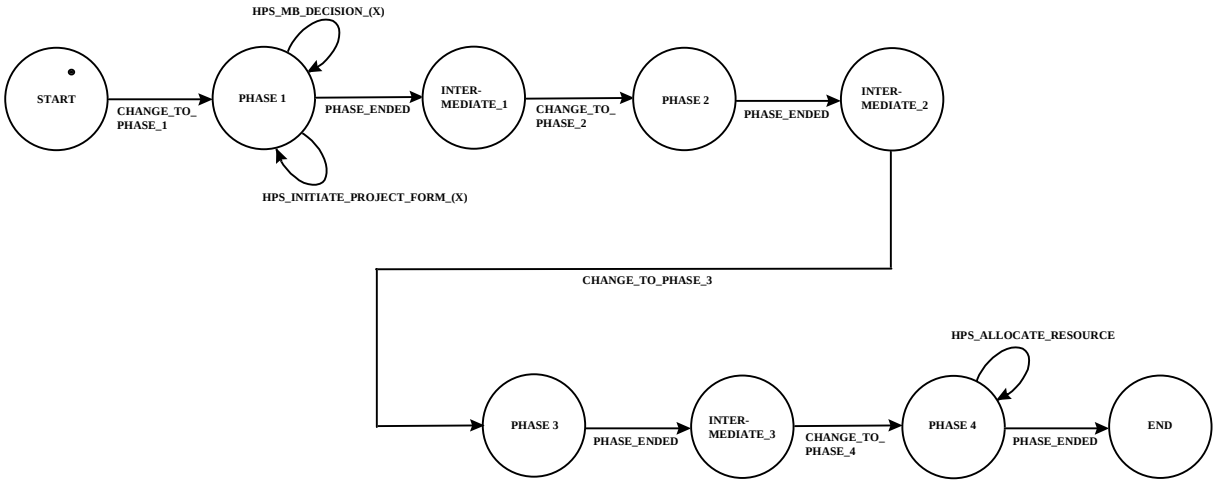


figure 6.107 head personnel section : ext_wpmd_head_personnel_section, total external STD

The flow through this total external STD is analogous to the flow through the total external STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : total external behavior-STD).

With the difference that the class ‘head personnel section’ does not participate in phase 2 or 3. Therefore no action of ‘head personnel section’ will take place in these phases.

6.3.2.9.3 Head personnel section (integration) : internal behavior-STDs

The internal operations of the class ‘head personnel section’ are in the first place the internal operations as modeled in the phase 1 and 4. During the integration the following internal operations are added to the class: ‘change_to_phase_1’, ‘change_to_ phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and ‘phase_ended’.

These 5 additional operations are so-called ‘no-operations’ (nops). A nop is an operation that performs no function while executing. The sole purpose of such an operation is to enable the corresponding transition of the external STD within the existing SOCCA framework.

These internal operations have internal STDs equivalent to the corresponding operations of the class ‘customer’ (described in the paragraph ‘Customer (integration) : internal behavior STDs’).

6.3.2.9.4 Head personnel section (integration) : manager-STD

The total manager STD is the same as the total external STD. Therefore no separate figure is needed for the total manager STD. The total manager STD of the process fragment ‘writing project management documents’ prescribes in its ‘phase 1’ and ‘phase 4’ states still the same subprocesses for its phase-employees as was modeled in the phase 1- and phase 4-sub-model. These may be called ‘old prescribed subprocesses’. Subprocess prescriptions for an employee in states (of the total manager STD) that are not relevant for that employee are ‘don’t cares’.

In the integration the total manager gets new employees. For this new employees it is also prescribing subprocesses. These may be called ‘new prescribed subprocesses’. The sum total of the prescribed subprocesses by the total manager STD is therefore (per state) : {old prescribed subprocesses} $\cup$ {don’t cares} $\cup$ {new prescribed subprocesses}.

The new CPSs (Consolidated Prescribed Subprocesses), CCs (Caller_Callee combinations) and TLFs (Transition Logical Formulas) for this total manager STD are the same as the CPSs, CCs and TLFs for the total manager STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : manager-STD).

With the difference that all the subprocesses and traps are with respect to ‘head personnel section’ instead of with respect to ‘requirements document’.

6.3.2.9.5 Head personnel section (integration) : employee-STDs

The employees of the manager STD of the class ‘head personnel section’ are in the first place the employees as modeled in the phase 1 and 4. During the integration the following 7 employees are added: ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and their caller ‘pr_project_life_cycle’ and ‘phase_ended’ and its caller ‘pr_phase_ended’.

The employee ‘change_to_phase_1’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head personnel section’), according to the caller-callee construct.

The employee ‘change_to_phase_2’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head personnel section’), according to the caller-callee construct.

The employee ‘change_to_phase_3’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head personnel section’), according to the caller-callee construct.

The employee ‘change_to_phase_4’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head personnel section’), according to the caller-callee construct.

The employee ‘pr_project_life_cycle’ is the caller of the four callees ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’ and ‘change_to_phase_4’ of the class ‘head personnel section’. Consequently it has five subprocesses and traps with respect to the manager STD of the class ‘head personnel section’ : S1, S2, S3, S4 and S5 and T-1, T-2, T-3, T-4 and T-5. The next figure shows these five subprocesses of the internal operation ‘pr_project_ life_cycle’.

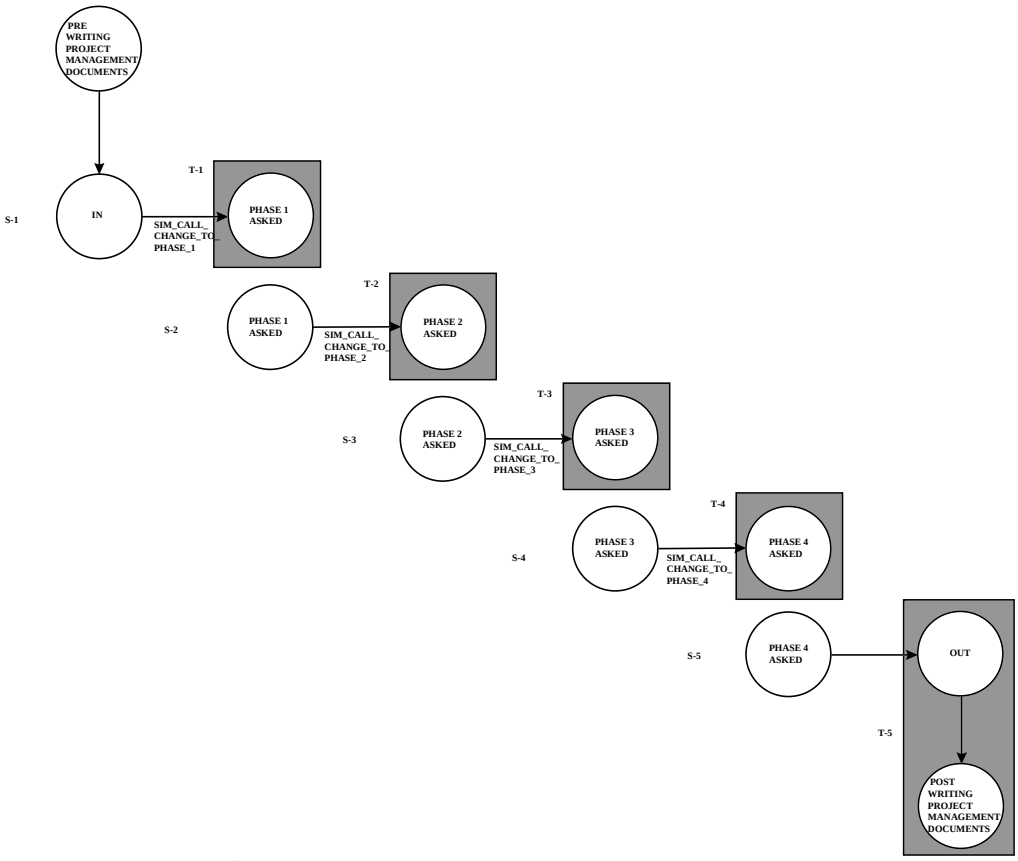


figure 6.108 employee int-pr_project_life_cycle : subprocesses S1, S2, S3, S4, S5 wrt_head_personnel_section

This operation ‘pr_project_life_cycle’ is not only an employee of ‘head personnel section’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1, S2, S3, S4 and S5. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses. The actual subprocesses of this

operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

The employee ‘phase_ended’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head personnel section’), according to the caller-callee construct.

The employee ‘pr_phase_ended’ is the caller of the callee ‘phase_ended’ of the class ‘head personnel section’. It has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head personnel section’) according to the caller_callee-construct.

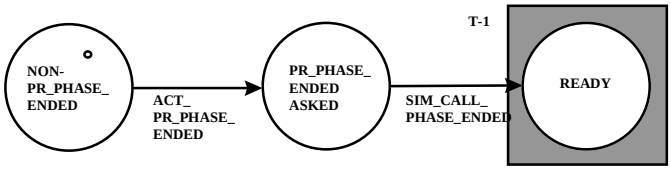


figure 6.109 employee int-pr_phase_ended : subprocess S1_wrt_head_personnel_section

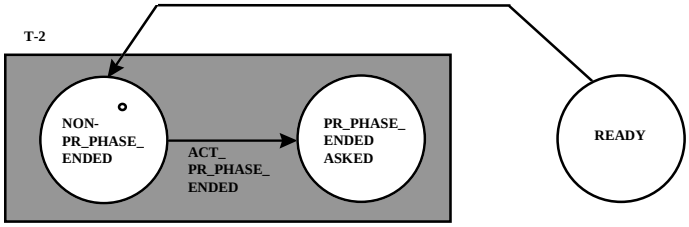


figure 6.110 employee int-pr_phase_ended : subprocess S2_wrt_head_personnel_section

This operation ‘pr_phase_ended’ is not only an employee of ‘head personnel section’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1 and S2. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses.

The actual subprocesses of this operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

6.3.2.10 Project form (integration)

6.3.2.10.1 Project form (integration) : extended phase-external behavior-STDs

The first step in the construction of the total external STD for the process fragment ‘writing project management documents’ is the extension of the constituent phase-external STDs with a start state and an end (final) state. The start state has a transition leaving it and entering the phase-external STD. The final state has a transition coming into it from the phase-external STD. This construction is performed in the next figures for the phase 1-external STD, the phase 2-external STD, the phase 3-external STD and the phase 4-external STD of the class ‘project form’.

The class ‘project form’ does not participate in phase 2 or 3. Therefore the phase 2- and phase 3-external STDs consist of a dummy state.

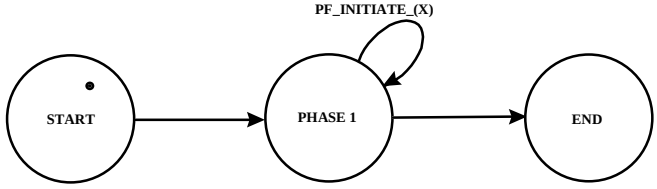


figure 6.111 project form : extended phase 1-external STD

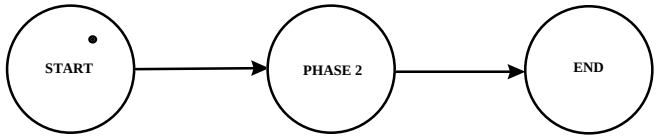


figure 6.112 project form : extended phase 2-external STD

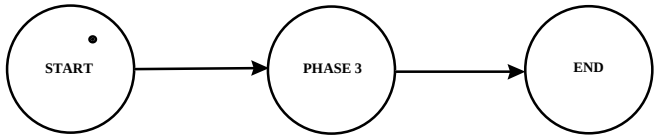


figure 6.113 project form : extended phase 3-external STD

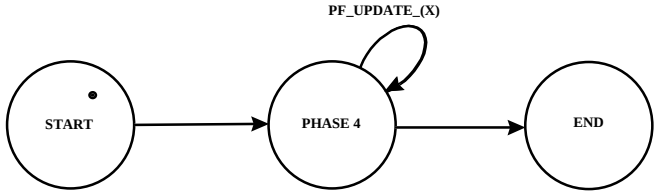


figure 6.114 project form : extended phase 4-external STD

6.3.2.10.2 Project form (integration) : total external behavior-STD

The second step in the construction of the total external STD for the process fragment ‘writing project management documents’ is to connect the extended phase-external STDs with each other in such a way that the final state of one phase-external STD coincides with the start state of the next phase-external STD. This ‘coinciding’ state between two phase-external STDs is called an ‘intermediate’ state. This models the sequential dependency between the phases.

The transitions leaving the start states get the label ‘change_to_phase_x’. These represent the ‘phase-changing’-operations. The transitions entering the final states get the label ‘pr_phase_ended’. These represent the ‘phase-ended’ operation. The construction is performed in the next figure.

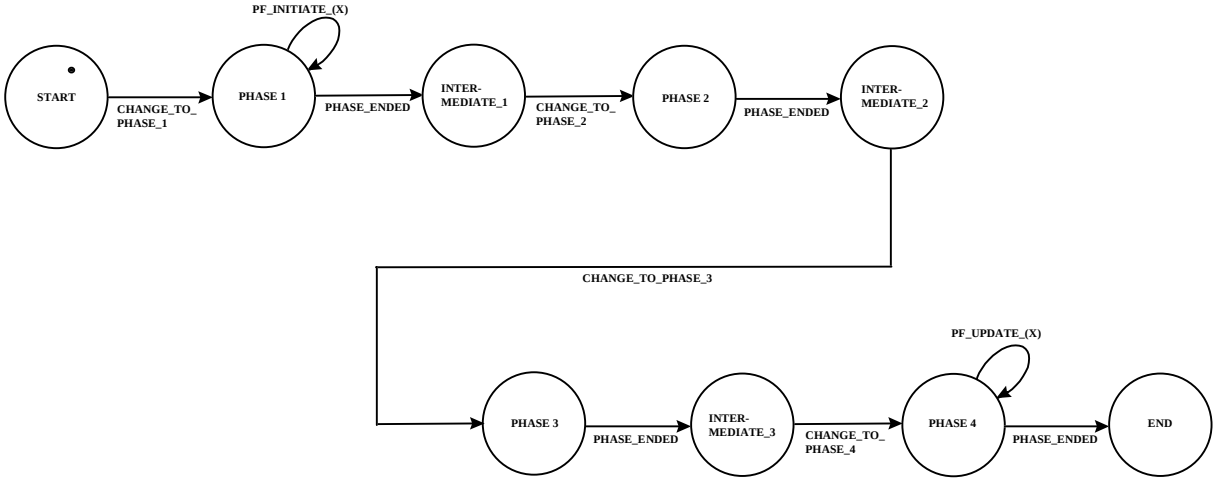


figure 6.115 project form : ext_wpmd_project_form, total external STD

The flow through this total external STD is analogous to the flow through the total external STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : total external behavior-STD).

With the difference that the class ‘project form’ does not participate in phase 2 or 3. Therefore no action of ‘project form’ will take place in these phases.

6.3.2.10.3 Project form (integration) : internal behavior-STDs

The internal operations of the class ‘project form’ are in the first place the internal operations as modeled in the phase 1 and 4. During the integration the following internal operations are added to the class: ‘change_to_phase_1’, ‘change_to_ phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and ‘phase_ended’.

These 5 additional operations are so-called ‘no-operations’ (nops). A nop is an operation that performs no function while executing. The sole purpose of such an operation is to enable the corresponding transition of the external STD within the existing SOCCA framework.

These internal operations have internal STDs equivalent to the corresponding operations of the class ‘customer’ (described in the paragraph ‘Customer (integration) : internal behavior STDs’).

6.3.2.10.4 Project form (integration) : manager-STD

The total manager STD is the same as the total external STD. Therefore no separate figure is needed for the total manager STD. The total manager STD of the process fragment ‘writing project management documents’ prescribes in its ‘phase 1’ and ‘phase 4’ states still the same subprocesses for its phase-employees as was modeled in the phase 1- and phase 4-sub-model. These may be called ‘old prescribed subprocesses’. Subprocess prescriptions for an employee in states (of the total manager STD) that are not relevant for that employee are ‘don’t cares’.

In the integration the total manager gets new employees. For this new employees it is also prescribing subprocesses. These may be called ‘new prescribed subprocesses’. The sum total of the prescribed subprocesses by the total manager STD is therefore (per state) : {old prescribed subprocesses} $\cup$ {don’t cares} $\cup$ {new prescribed subprocesses}.

The new CPSs (Consolidated Prescribed Subprocesses), CCs (Caller_Callee combinations) and TLFs (Transition Logical Formulas) for this total manager STD are the same as the CPSs, CCs and TLFs for the total manager STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : manager-STD).

With the difference that all the subprocesses and traps are with respect to ‘project form’ instead of with respect to ‘requirements document’.

6.3.2.10.5 Project form (integration) : employee-STDs

The employees of the manager STD of the class ‘project form’ are in the first place the employees as modeled in the phase 1 and 4. During the integration the following 7 employees are added: ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and their caller ‘pr_project_life_cycle’ and ‘phase_ended’ and its caller ‘pr_phase_ended’.

The employee ‘change_to_phase_1’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘project form’), according to the caller-callee construct.

The employee ‘change_to_phase_2’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘project form’), according to the caller-callee construct.

The employee ‘change_to_phase_3’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘project form’), according to the caller-callee construct.

The employee ‘change_to_phase_4’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘project form’), according to the caller-callee construct.

The employee ‘pr_project_life_cycle’ is the caller of the four callees ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’ and ‘change_to_phase_4’ of the class ‘project form’. Consequently it has five subprocesses and traps with respect to the manager STD of the class ‘project form’ : S1, S2, S3, S4 and S5 and T-1, T-2, T-3, T-4 and T-5. The next figure shows these five subprocesses of the internal operation ‘pr_project_ life_cycle’.

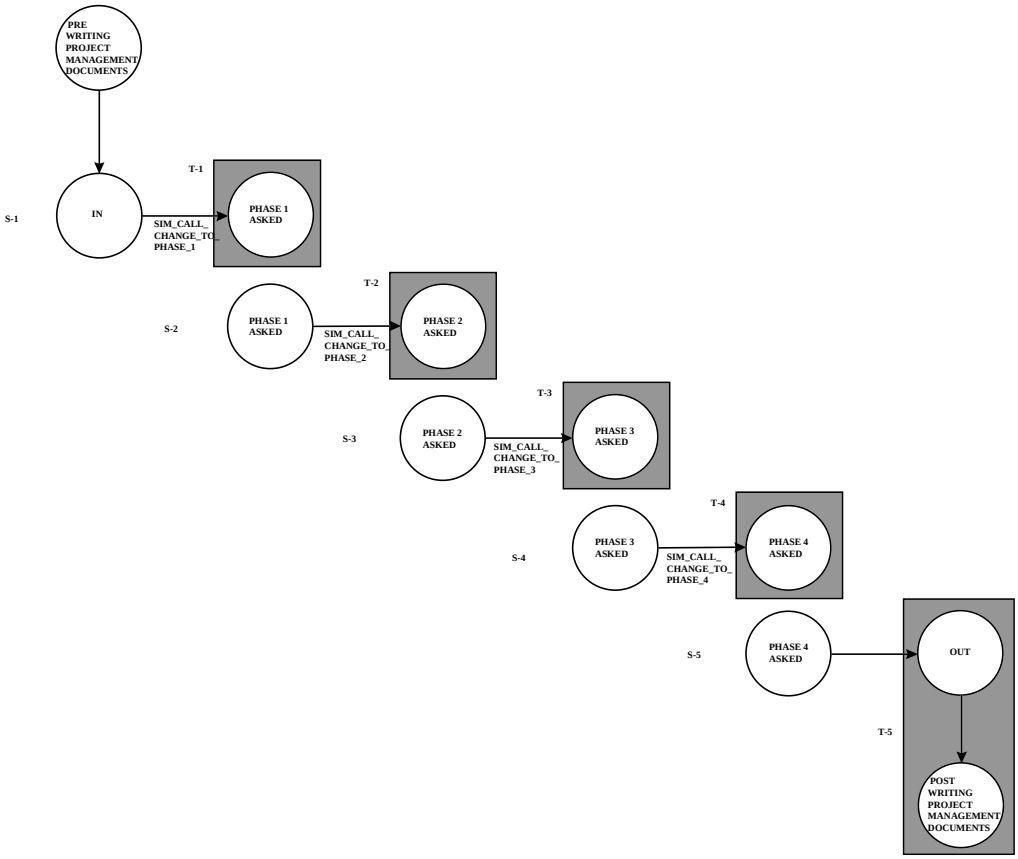


figure 6.116 employee int-pr_project_life_cycle : subprocesses S1, S2, S3, S4, S5 wrt_project_form

This operation ‘pr_project_life_cycle’ is not only an employee of ‘project form’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1, S2, S3, S4 and S5. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses. The actual subprocesses of this operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

The employee ‘phase_ended’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘project form’), according to the caller-callee construct.

The employee ‘pr_phase_ended’ is the caller of the callee ‘phase_ended’ of the class ‘project form’. It has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘project form’) according to the caller_callee-construct.

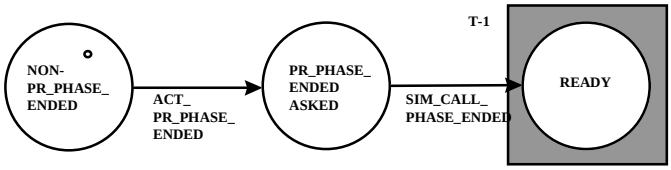


figure 6.117 employee int-pr_phase_ended : subprocess S1_wrt_project_form

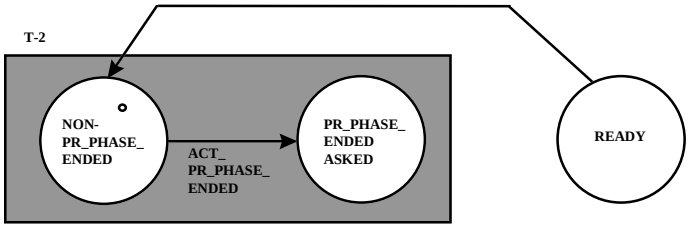


figure 6.118 employee int-pr_phase_ended : subprocess S2_wrt_project_form

This operation ‘pr_phase_ended’ is not only an employee of ‘project form’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1 and S2. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses.

The actual subprocesses of this operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

6.3.2.11 Head controller section (integration)

6.3.2.11.1 Head controller section (integration) : extended phase-external behavior-STDs

The first step in the construction of the total external STD for the process fragment ‘writing project management documents’ is the extension of the constituent phase-external STDs with a start state and an end (final) state. The start state has a transition leaving it and entering the phase-external STD. The final state has a transition coming into it from the phase-external STD. This construction is performed in the next figures for the phase 1-external STD, the phase 2-external STD, the phase 3-external STD and the phase 4-external STD of the class ‘head controller section’.

The class ‘head controller section’ does not participate in phase 2. Therefore the phase 2-external STD consists of a dummy state.

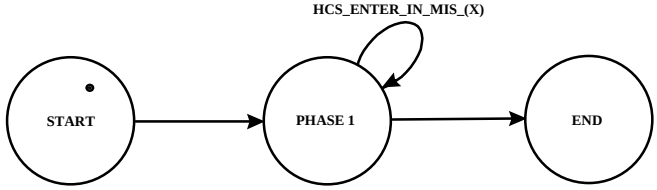


figure 6.119 head controller section : extended phase 1-external STD

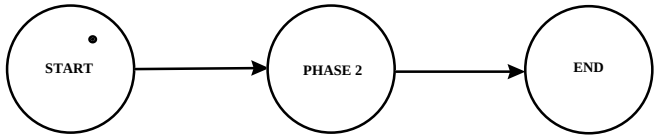


figure 6.120 head controller section : extended phase 2-external STD

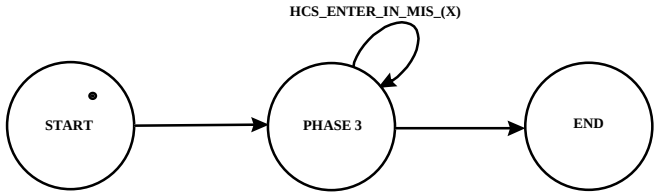


figure 6.121 head controller section : extended phase 3-external STD

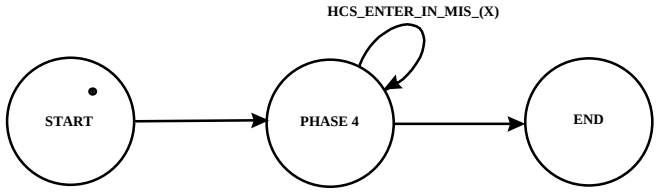


figure 6.122 head controller section : extended phase 4-external STD

6.3.2.11.2 Head controller section (integration) : total external behavior-STD

The second step in the construction of the total external STD for the process fragment ‘writing project management documents’ is to connect the extended phase-external STDs with each other in such a way that the final state of one phase-external STD coincides with the start state of the next phase-external STD. This ‘coinciding’ state between two phase-external STDs is called an ‘intermediate’ state. This models the sequential dependency between the phases.

The transitions leaving the start states get the label ‘change_to_phase_x’. These represent the ‘phase-changing’-operations. The transitions entering the final states get the label ‘pr_phase_ended’. These represent the ‘phase-ended’ operation. The construction is performed in the next figure.

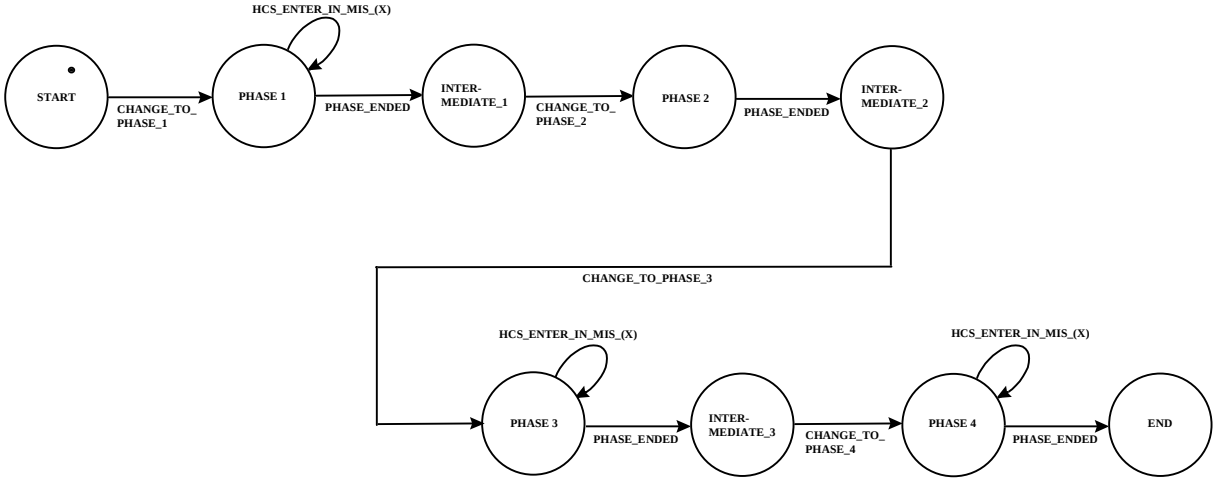


figure 6.123 head controller section : ext_wpmd_head_controller_section, total external STD

The flow through this total external STD is analogous to the flow through the total external STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : total external behavior-STD).

With the difference that the class ‘head controller section’ does not participate in phase 2. Therefore no action of ‘head controller section’ will take place in that phase.

6.3.2.11.3 Head controller section (integration) : internal behavior-STDs

The internal operations of the class ‘head controller section’ are in the first place the internal operations as modeled in the phase 1, 3 and 4. During the integration the following internal operations are added to the class: ‘change_to_phase_1’, ‘change_to_ phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and ‘phase_ended’.

These 5 additional operations are so-called ‘no-operations’ (nops). A nop is an operation that performs no function while executing. The sole purpose of such an operation is to enable the corresponding transition of the external STD within the existing SOCCA framework.

These internal operations have internal STDs equivalent to the corresponding operations of the class ‘customer’ (described in the paragraph ‘Customer (integration) : internal behavior STDs’).

6.3.2.11.4 Head controller section (integration) : manager-STD

The total manager STD is the same as the total external STD. Therefore no separate figure is needed for the total manager STD. The total manager STD of the process fragment ‘writing project management documents’ prescribes in its ‘phase 1’, phase 3’ and ‘phase 4’ states still the same subprocesses for its phase-employees as was modeled in the phase 1-, phase 3- and phase 4-sub-model. These may be called ‘old prescribed subprocesses’. Subprocess prescriptions for an employee in states (of the total manager STD) that are not relevant for that employee are ‘don’t cares’.

In the integration the total manager gets new employees. For this new employees it is also prescribing subprocesses. These may be called ‘new prescribed subprocesses’. The sum total of the prescribed subprocesses by the total manager STD is therefore (per state) : {old prescribed subprocesses} $\cup$ {don’t cares} $\cup$ {new prescribed subprocesses}.

The new CPSs (Consolidated Prescribed Subprocesses), CCs (Caller_Callee combinations) and TLFs (Transition Logical Formulas) for this total manager STD are the same as the CPSs, CCs and TLFs for the total manager STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : manager-STD).

With the difference that all the subprocesses and traps are with respect to ‘head controller section’ instead of with respect to ‘requirements document’.

6.3.2.11.5 Head controller section (integration) : employee-STDs

The employees of the manager STD of the class ‘head controller section’ are in the first place the employees as modeled in the phase 1, 3 and 4. During the integration the following 7 employees are added: ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and their caller ‘pr_project_life_cycle’ and ‘phase_ended’ and its caller ‘pr_phase_ended’.

The employee ‘change_to_phase_1’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head controller section’), according to the caller-callee construct.

The employee ‘change_to_phase_2’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head controller section’), according to the caller-callee construct.

The employee ‘change_to_phase_3’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head controller section’), according to the caller-callee construct.

The employee ‘change_to_phase_4’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head controller section’), according to the caller-callee construct.

The employee ‘pr_project_life_cycle’ is the caller of the four callees ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’ and ‘change_to_phase_4’ of the class ‘head controller section’. Consequently it has five subprocesses and traps with respect to the manager STD of the class ‘head controller section’ : S1, S2, S3, S4 and S5 and T-1, T-2, T-3, T-4 and T-5. The next figure shows these five subprocesses of the internal operation ‘pr_project_ life_cycle’.

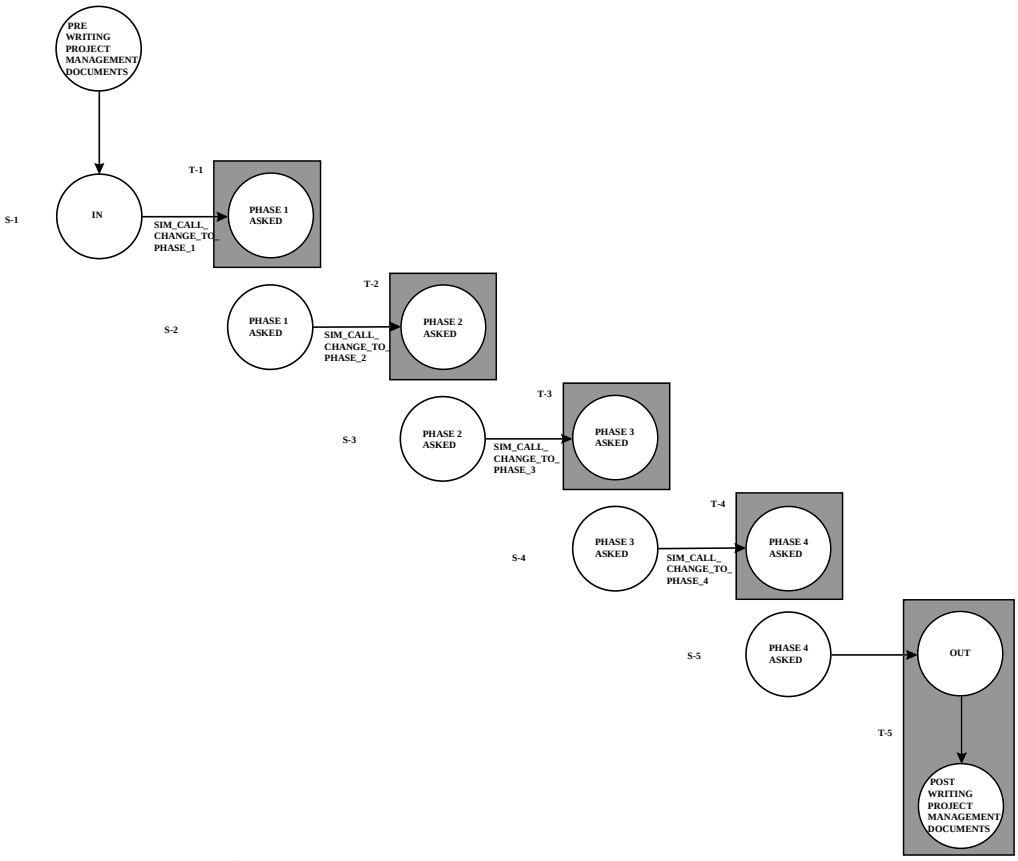


figure 6.124 employee int-pr_project_life_cycle : subprocesses S1, S2, S3, S4, S5 wrt_head_controller_section

This operation ‘pr_project_life_cycle’ is not only an employee of ‘head controller section’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1, S2, S3, S4 and S5. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses. The actual subprocesses of this

operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

The employee ‘phase_ended’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head controller section’), according to the caller-callee construct.

The employee ‘pr_phase_ended’ is the caller of the callee ‘phase_ended’ of the class ‘head_controller section’. It has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head controller section’) according to the caller_callee-construct.

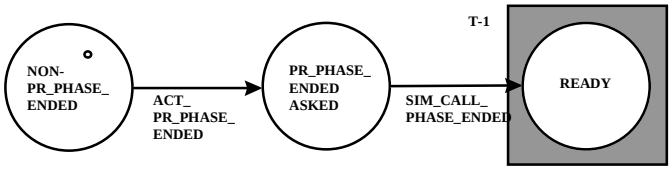


figure 6.125 employee int-pr_phase_ended : subprocess S1_wrt_head_controller_section

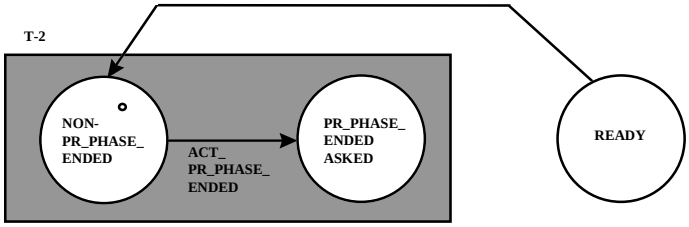


figure 6.126 employee int-pr_phase_ended : subprocess S2_wrt_head_controller_section

This operation ‘pr_phase_ended’ is not only an employee of ‘head controller section’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1 and S2. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses.

The actual subprocesses of this operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

6.3.2.12 Technical project manager (integration)

6.3.2.12.1 Technical project manager (integration) : extended phase-external behavior-STDs

The first step in the construction of the total external STD for the process fragment ‘writing project management documents’ is the extension of the constituent phase-external STDs with a start state and an end (final) state. The start state has a transition leaving it and entering the phase-external STD. The final state has a transition coming into it from the phase-external STD. This construction is performed in the next figures for the phase 1-external STD, the phase 2-external STD, the phase 3-external STD and the phase 4-external STD of the class ‘technical project manager’.

The class ‘technical project manager’ does not participate in phase 1. Therefore the phase 1-external STD consists of a dummy state.

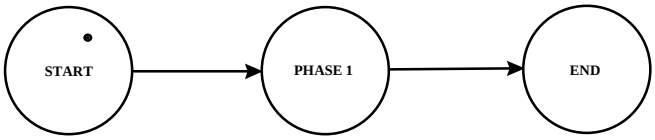


figure 6.127 technical project manager : extended phase 1-external STD

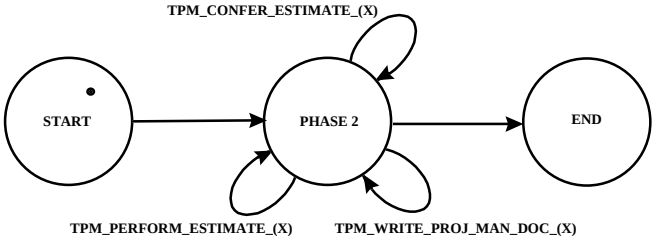


figure 6.128 technical project manager : extended phase 2-external STD

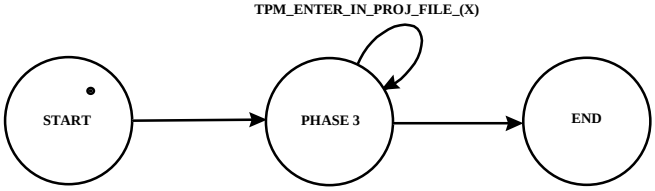


figure 6.129 technical project manager : extended phase 3-external STD

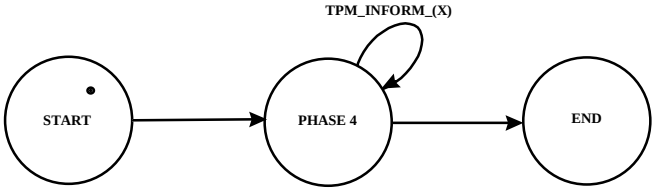


figure 6.130 technical project manager : extended phase 4-external STD

6.3.2.12.2 Technical project manager (integration) : total external behavior-STD

The second step in the construction of the total external STD for the process fragment ‘writing project management documents’ is to connect the extended phase-external STDs with each other in such a way that the final state of one phase-external STD coincides with the start state of the next phase-external STD. This ‘coinciding’ state between two phase-external STDs is called an ‘intermediate’ state. This models the sequential dependency between the phases.

The transitions leaving the start states get the label ‘change_to_phase_x’. These represent the ‘phase-changing’-operations. The transitions entering the final states get the label ‘pr_phase_ended’. These represent the ‘phase-ended’ operation. The construction is performed in the next figure.

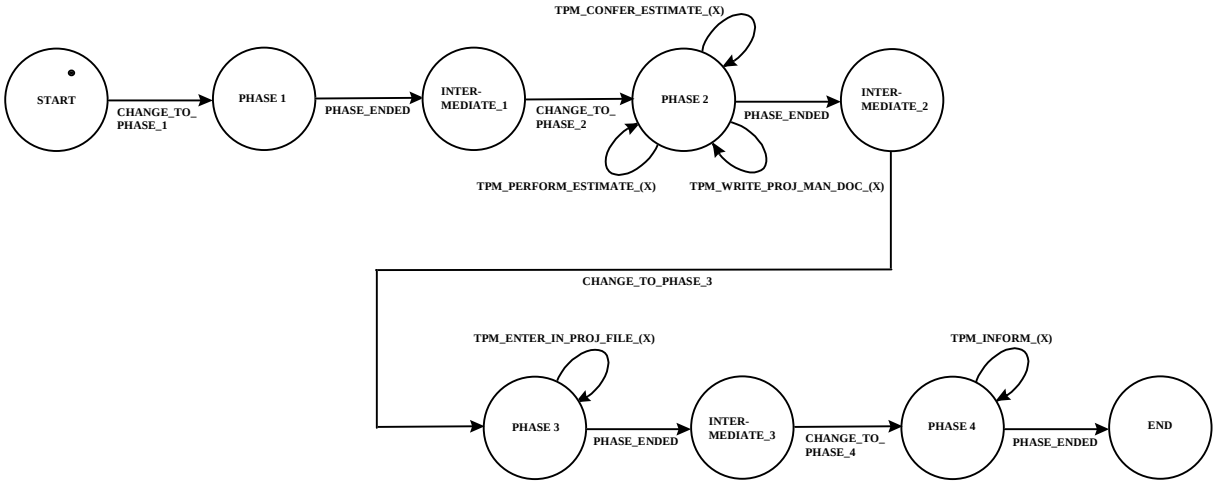


figure 6.131 technical project manager : ext_wpmd_technical_project_manager, total external STD

The flow through this total external STD is analogous to the flow through the total external STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : total external behavior-STD).

With the difference that the class ‘technical project manager’ does not participate in phase 1. Therefore no action of ‘technical project manager’ will take place in that phase.

6.3.2.12.3 Technical project manager (integration) : internal behavior-STDs

The internal operations of the class ‘technical project manager’ are in the first place the internal operations as modeled in the phase 2, 3 and 4. During the integration the following internal operations are added to the class: ‘change_to_phase_1’, ‘change_to_ phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and ‘phase_ended’.

These 5 additional operations are so-called ‘no-operations’ (nops). A nop is an operation that performs no function while executing. The sole purpose of such an operation is to enable the corresponding transition of the external STD within the existing SOCCA framework.

These internal operations have internal STDs equivalent to the corresponding operations of the class ‘customer’ (described in the paragraph ‘Customer (integration) : internal behavior STDs’).

6.3.2.12.4 Technical project manager (integration) : manager-STD

The total manager STD is the same as the total external STD. Therefore no separate figure is needed for the total manager STD. The total manager STD of the process fragment ‘writing project management documents’ prescribes in its ‘phase 2’, phase 3’ and ‘phase 4’ states still the same subprocesses for its phase-employees as was modeled in the phase 2-, phase 3- and phase 4-sub-models. These may be called ‘old prescribed subprocesses’. Subprocess prescriptions for an employee in states (of the total manager STD) that are not relevant for that employee are ‘don’t cares’.

In the integration the total manager gets new employees. For this new employees it is also prescribing subprocesses. These may be called ‘new prescribed subprocesses’. The sum total of the prescribed subprocesses by the total manager STD is therefore (per state) : {old prescribed subprocesses} $\cup$ {don’t cares} $\cup$ {new prescribed subprocesses}.

The new CPSs (Consolidated Prescribed Subprocesses), CCs (Caller_Callee combinations) and TLFs (Transition Logical Formulas) for this total manager STD are the same as the CPSs, CCs and TLFs for the total manager STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : manager-STD). With the difference that all the subprocesses and traps are with respect to ‘technical project manager’ instead of with respect to ‘requirements document’.

6.3.2.12.5 Technical project manager (integration) : employee-STDs

The employees of the manager STD of the class ‘technical project manager’ are in the first place the employees as modeled in the phase 2, 3 and 4. During the integration the following 7 employees are added: ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and their caller ‘pr_project_life_cycle’ and ‘phase_ended’ and its caller ‘pr_phase_ended’.

The employee ‘change_to_phase_1’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘technical project manager’), according to the caller-callee construct.

The employee ‘change_to_phase_2’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘technical project manager’), according to the caller-callee construct.

The employee ‘change_to_phase_3’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘technical project manager’), according to the caller-callee construct.

The employee ‘change_to_phase_4’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘technical project manager’), according to the caller-callee construct.

The employee ‘pr_project_life_cycle’ is the caller of the four callees ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’ and ‘change_to_phase_4’ of the class ‘technical project manager’. Consequently it has five subprocesses and traps with respect to the manager STD of the class ‘technical project manager’ : S1, S2, S3, S4 and S5 and T-1, T-2, T-3, T-4 and T-5. The next figure shows these five subprocesses of the internal operation ‘pr_project_ life_cycle’.

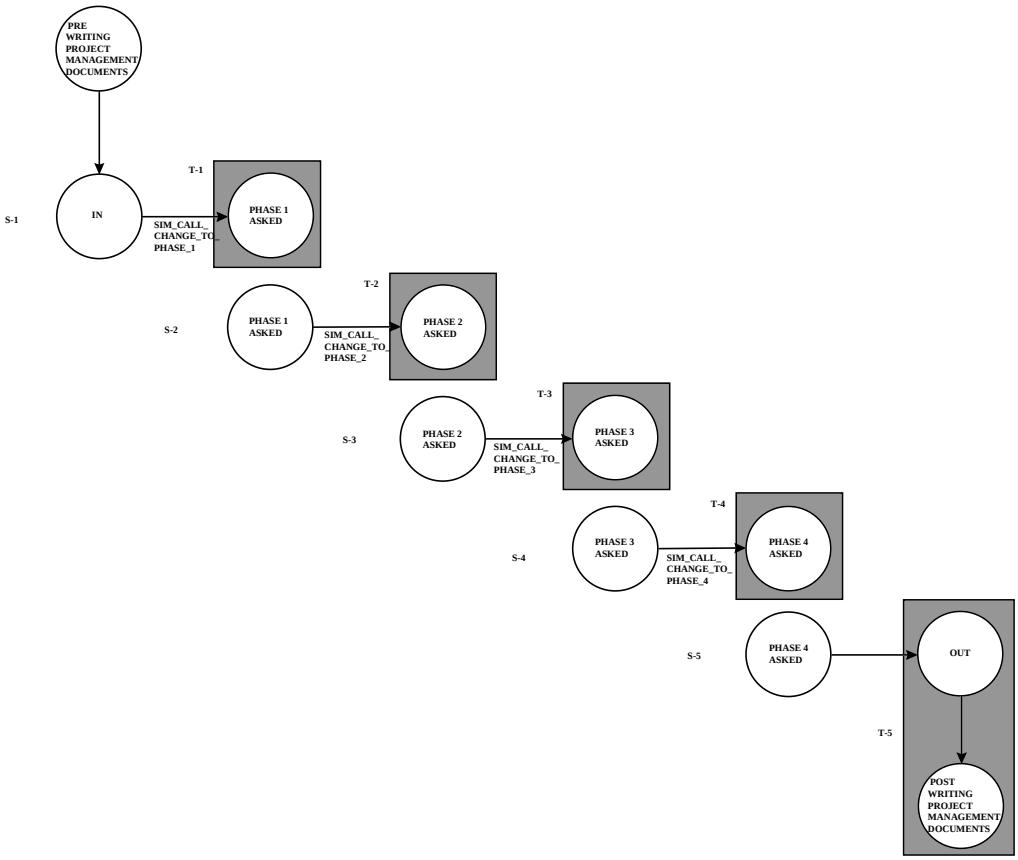


figure 6.132 employee int-pr_project_life_cycle : subprocesses S1, S2, S3, S4, S5 wrt_technical_project_manager

This operation ‘pr_project_life_cycle’ is not only an employee of ‘technical project manager’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1, S2, S3, S4 and S5. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses. The actual subprocesses of this

operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

The employee ‘phase_ended’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘technical project manager’), according to the caller-callee construct.

The employee ‘pr_phase_ended’ is the caller of the callee ‘phase_ended’ of the class ‘technical project manager’. It has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘technical project manager’) according to the caller_callee-construct.

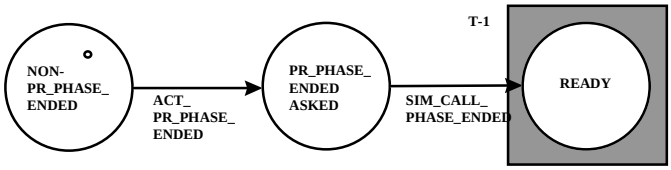


figure 6.133 employee int-pr_phase_ended : subprocess S1_wrt_technical_project_manager

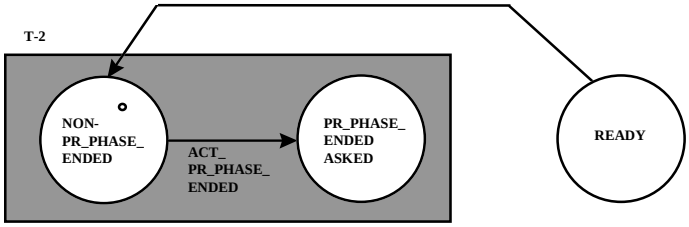


figure 6.134 employee int-pr_phase_ended : subprocess S2_wrt_technical_project_manager

This operation ‘pr_phase_ended’ is not only an employee of ‘technical project manager’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1 and S2. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses.

The actual subprocesses of this operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

6.3.2.13 Quality assurance adviser (integration)

6.3.2.13.1 Quality assurance adviser (integration) : extended phase-external behavior-STDs

The first step in the construction of the total external STD for the process fragment ‘writing project management documents’ is the extension of the constituent phase-external STDs with a start state and an end (final) state. The start state has a transition leaving it and entering the phase-external STD. The final state has a transition coming into it from the phase-external STD. This construction is performed in the next figures for the phase 1-external STD, the phase 2-external STD, the phase 3-external STD and the phase 4-external STD of the class ‘quality assurance adviser’.

The class ‘quality assurance adviser’ does not participate in phase 1 or 4. Therefore the phase 1- and phase 4-external STDs consist of a dummy state.

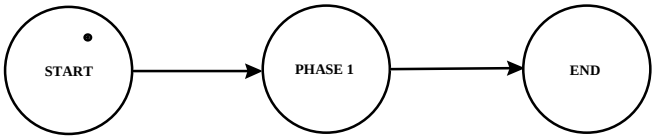


figure 6.135 quality assurance adviser : extended phase 1-external STD

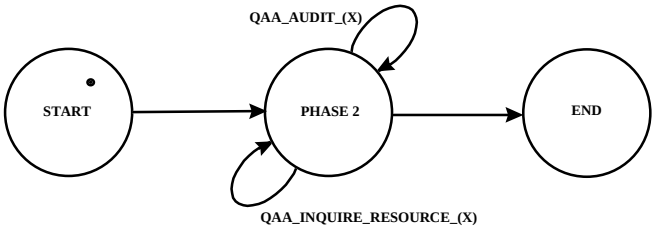


figure 6.136 quality assurance adviser : extended phase 2-external STD

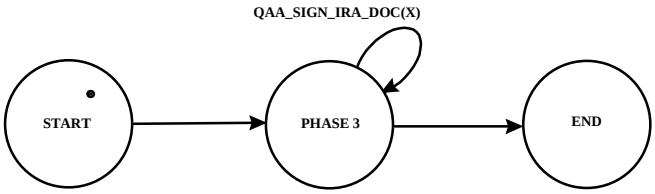


figure 6.137 quality assurance adviser : extended phase 3-external STD

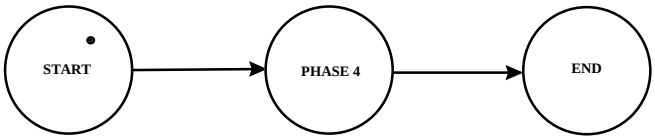


figure 6.138 quality assurance adviser : extended phase 4-external STD

6.3.2.13.2 Quality assurance adviser (integration) : total external behavior-STD

The second step in the construction of the total external STD for the process fragment ‘writing project management documents’ is to connect the extended phase-external STDs with each other in such a way that the final state of one phase-external STD coincides with the start state of the next phase-external STD. This ‘coinciding’ state between two phase-external STDs is called an ‘intermediate’ state. This models the sequential dependency between the phases.

The transitions leaving the start states get the label ‘change_to_phase_x’. These represent the ‘phase-changing’-operations. The transitions entering the final states get the label ‘pr_phase_ended’. These represent the ‘phase-ended’ operation. The construction is performed in the next figure.

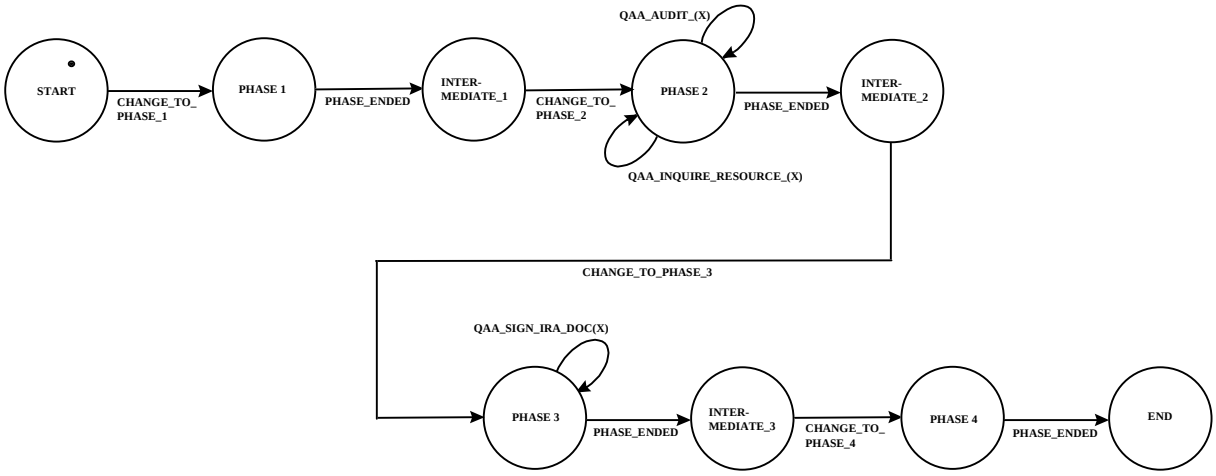


figure 6.139 quality assurance adviser : ext_wpmd_quality_assurance_adviser, total external STD

The flow through this total external STD is analogous to the flow through the total external STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : total external behavior-STD).

With the difference that the class ‘quality assurance adviser’ does not participate in phase 1 or 4. Therefore no action of ‘quality assurance adviser’ will take place in these phases.

6.3.2.13.3 Quality assurance adviser (integration) : internal behavior-STDs

The internal operations of the class ‘quality assurance adviser’ are in the first place the internal operations as modeled in the phase 2 and 3. During the integration the following internal operations are added to the class: ‘change_to_phase_1’, ‘change_to_ phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and ‘phase_ended’.

These 5 additional operations are so-called ‘no-operations’ (nops). A nop is an operation that performs no function while executing. The sole purpose of such an operation is to enable the corresponding transition of the external STD within the existing SOCCA framework.

These internal operations have internal STDs equivalent to the corresponding operations of the class ‘customer’ (described in the paragraph ‘Customer (integration) : internal behavior STDs’).

6.3.2.13.4 Quality assurance adviser (integration) : manager-STD

The total manager STD is the same as the total external STD. Therefore no separate figure is needed for the total manager STD. The total manager STD of the process fragment ‘writing project management documents’ prescribes in its ‘phase 2’ and ‘phase 3’ states still the same subprocesses for its phase-employees as was modeled in the phase 2-, and phase 3-sub-models. These may be called ‘old prescribed subprocesses’. Subprocess prescriptions for an employee in states (of the total manager STD) that are not relevant for that employee are ‘don’t cares’.

In the integration the total manager gets new employees. For this new employees it is also prescribing subprocesses. These may be called ‘new prescribed subprocesses’. The sum total of the prescribed subprocesses by the total manager STD is therefore (per state) : {old prescribed subprocesses} $\cup$ {don’t cares} $\cup$ {new prescribed subprocesses}.

The new CPSs (Consolidated Prescribed Subprocesses), CCs (Caller_Callee combinations) and TLFs (Transition Logical Formulas) for this total manager STD are the same as the CPSs, CCs and TLFs for the total manager STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : manager-STD).

With the difference that all the subprocesses and traps are with respect to ‘quality assurance adviser’ instead of with respect to ‘requirements document’.

6.3.2.13.5 Quality assurance adviser (integration) : employee-STDs

The employees of the manager STD of the class ‘quality assurance adviser’ are in the first place the employees as modeled in the phase 2 and 3. During the integration the following 7 employees are added: ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and their caller ‘pr_project_life_cycle’ and ‘phase_ended’ and its caller ‘pr_phase_ended’.

The employee ‘change_to_phase_1’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘quality assuance adviser’), according to the caller-callee construct.

The employee ‘change_to_phase_2’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘quality assurance adviser’), according to the caller-callee construct.

The employee ‘change_to_phase_3’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘quality assurance adviser’), according to the caller-callee construct.

The employee ‘change_to_phase_4’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘quality assurance adviser’), according to the caller-callee construct.

The employee ‘pr_project_life_cycle’ is the caller of the four callees ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’ and ‘change_to_phase_4’ of the class ‘quality assurance adviser’. Consequently it has five subprocesses and traps with respect to the manager STD of the class ‘quality assurance adviser’ : S1, S2, S3, S4 and S5 and T-1, T-2, T-3, T-4 and T-5. The next figure shows these five subprocesses of the internal operation ‘pr_project_ life_cycle’.

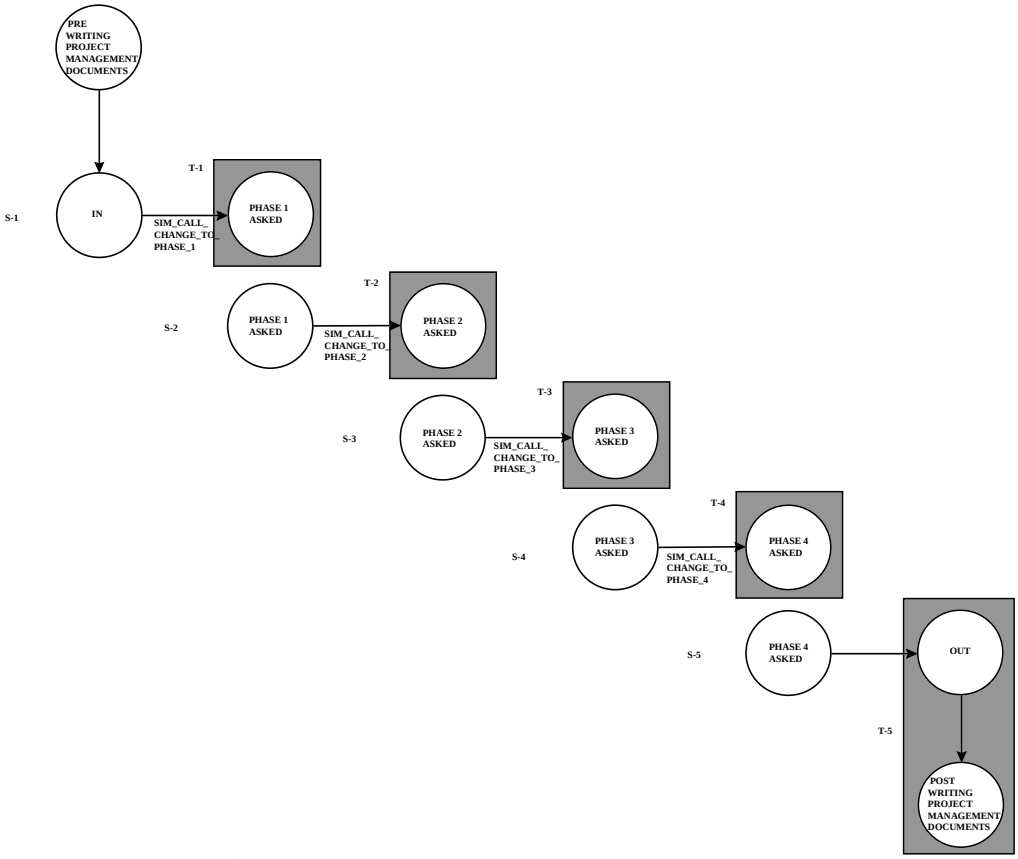


figure 6.140 employee int-pr_project_life_cycle : subprocesses S1, S2, S3, S4, S5 wrt_quality_assurance_adviser

This operation ‘pr_project_life_cycle’ is not only an employee of ‘quality assurance adviser’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1, S2, S3, S4 and S5. The operation has therefore 19 subprocesses prescribed to it at any

one time. The actual subprocess is the intersection of these 19 prescribed subprocesses. The actual subprocesses of this operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

The employee ‘phase_ended’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘quality assurance adviser’), according to the caller-callee construct.

The employee ‘pr_phase_ended’ is the caller of the callee ‘phase_ended’ of the class ‘quality assurance adviser’. It has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘quality assurance adviser’) according to the caller_callee-construct.

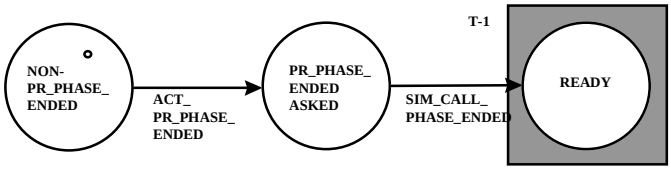


figure 6.141 employee int-pr_phase_ended : subprocess S1_wrt_quality_assurance_adviser

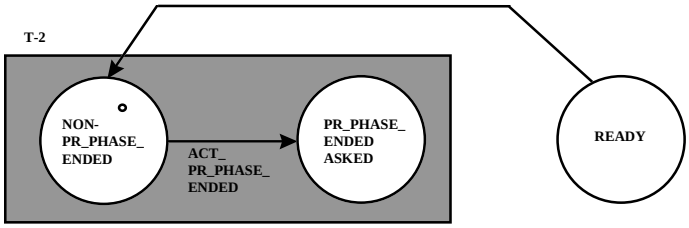


figure 6.142 employee int-pr_phase_ended : subprocess S2_wrt_quality_assurance_adviser

This operation ‘pr_phase_ended’ is not only an employee of ‘quality assurance adviser’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1 and S2. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses.

The actual subprocesses of this operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

6.3.2.14 Head production section (integration)

6.3.2.14.1 Head production section (integration) : extended phase-external behavior-STDs

The first step in the construction of the total external STD for the process fragment ‘writing project management documents’ is the extension of the constituent phase-external STDs with a start state and an end (final) state. The start state has a transition leaving it and entering the phase-external STD. The final state has a transition coming into it from the phase-external STD. This construction is performed in the next figures for the phase 1-external STD, the phase 2-external STD, the phase 3-external STD and the phase 4-external STD of the class ‘head production section’.

The class ‘head production section’ does not participate in phase 1 or 3. Therefore the phase 1- and phase 3-external STDs consist of a dummy state.

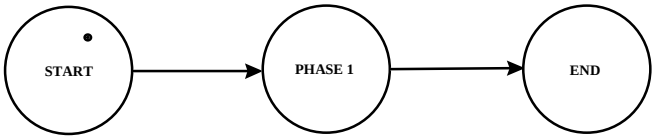


figure 6.143 head production section : extended phase 1-external STD

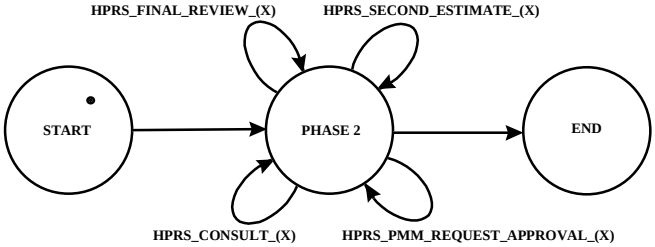


figure 6.144 head production section : extended phase 2-external STD

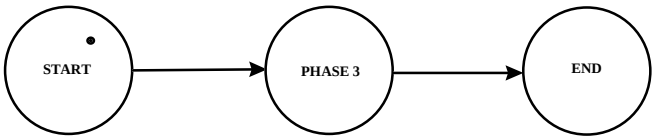


figure 6.145 head production section : extended phase 3-external STD

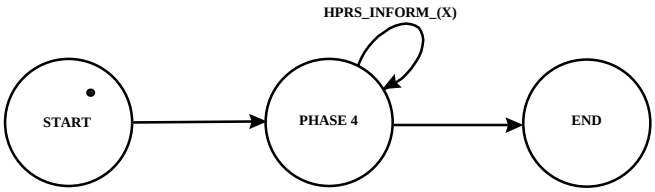


figure 6.146 head production section : extended phase 4-external STD

6.3.2.14.2 Head production section (integration) : total external behavior-STD

The second step in the construction of the total external STD for the process fragment ‘writing project management documents’ is to connect the extended phase-external STDs with each other in such a way that the final state of one phase-external STD coincides with the start state of the next phase-external STD. This ‘coinciding’ state between two phase-external STDs is called an ‘intermediate’ state. This models the sequential dependency between the phases.

The transitions leaving the start states get the label ‘change_to_phase_x’. These represent the ‘phase-changing’-operations. The transitions entering the final states get the label ‘pr_phase_ended’. These represent the ‘phase-ended’ operation. The construction is performed in the next figure.

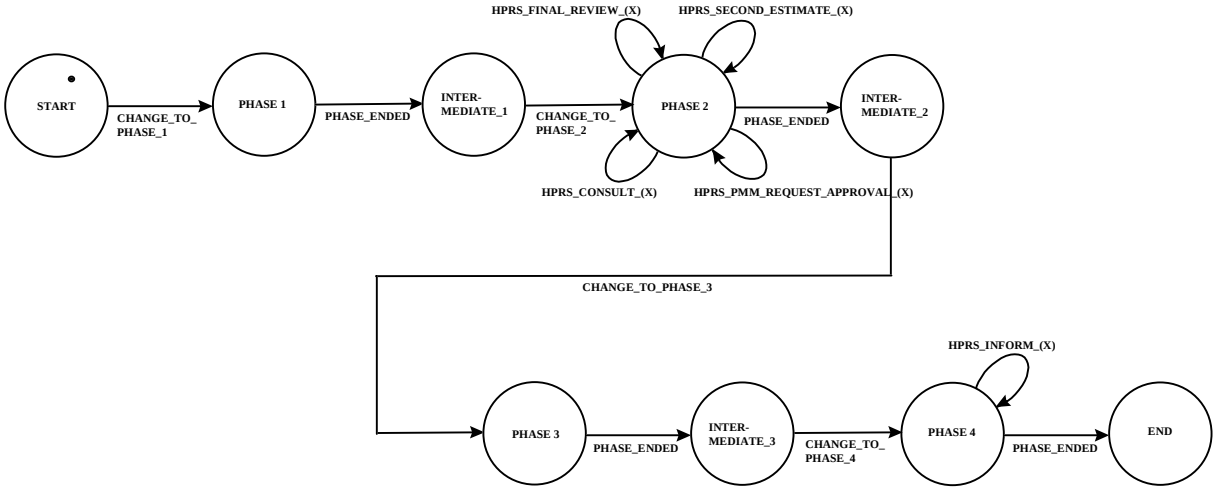


figure 6.147 head production section : ext_wpmd_head_production_section, total external STD

The flow through this total external STD is analogous to the flow through the total external STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : total external behavior-STD).

With the difference that the class ‘head production section’ does not participate in phase 1 or 3. Therefore no action of ‘head production section’ will take place in these phases.

6.3.2.14.3 Head production section (integration) : internal behavior-STDs

The internal operations of the class ‘head production section’ are in the first place the internal operations as modeled in the phase 2 and 4. During the integration the following internal operations are added to the class: ‘change_to_phase_1’, ‘change_to_ phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and ‘phase_ended’.

These 5 additional operations are so-called ‘no-operations’ (nops). A nop is an operation that performs no function while executing. The sole purpose of such an operation is to enable the corresponding transition of the external STD within the existing SOCCA framework.

These internal operations have internal STDs equivalent to the corresponding operations of the class ‘customer’ (described in the paragraph ‘Customer (integration) : internal behavior STDs’).

6.3.2.14.4 Head production section (integration) : manager-STD

The total manager STD is the same as the total external STD. Therefore no separate figure is needed for the total manager STD. The total manager STD of the process fragment ‘writing project management documents’ prescribes in its ‘phase 2’ and ‘phase 4’ states still the same subprocesses for its phase-employees as was modeled in the phase 2-, and phase 4-sub-models. These may be called ‘old prescribed subprocesses’. Subprocess prescriptions for an employee in states (of the total manager STD) that are not relevant for that employee are ‘don’t cares’.

In the integration the total manager gets new employees. For this new employees it is also prescribing subprocesses. These may be called ‘new prescribed subprocesses’. The sum total of the prescribed subprocesses by the total manager STD is therefore (per state) : {old prescribed subprocesses} $\cup$ {don’t cares} $\cup$ {new prescribed subprocesses}.

The new CPSs (Consolidated Prescribed Subprocesses), CCs (Caller_Callee combinations) and TLFs (Transition Logical Formulas) for this total manager STD are the same as the CPSs, CCs and TLFs for the total manager STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : manager-STD). With the difference that all the subprocesses and traps are with respect to ‘head production section’ instead of with respect to ‘requirements document’.

6.3.2.14.5 Head production section (integration) : employee-STDs

The employees of the manager STD of the class ‘head production section’ are in the first place the employees as modeled in the phase 2 and 4. During the integration the following 7 employees are added: ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and their caller ‘pr_project_life_cycle’ and ‘phase_ended’ and its caller ‘pr_phase_ended’.

The employee ‘change_to_phase_1’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head production section’), according to the caller-callee construct.

The employee ‘change_to_phase_2’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head production section’), according to the caller-callee construct.

The employee ‘change_to_phase_3’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head production section’), according to the caller-callee construct.

The employee ‘change_to_phase_4’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head production section’), according to the caller-callee construct.

The employee ‘pr_project_life_cycle’ is the caller of the four callees ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’ and ‘change_to_phase_4’ of the class ‘head production section’. Consequently it has five subprocesses and traps with respect to the manager STD of the class ‘head production section’ : S1, S2, S3, S4 and S5 and T-1, T-2, T-3, T-4 and T-5. The next figure shows these five subprocesses of the internal operation ‘pr_project_ life_cycle’.

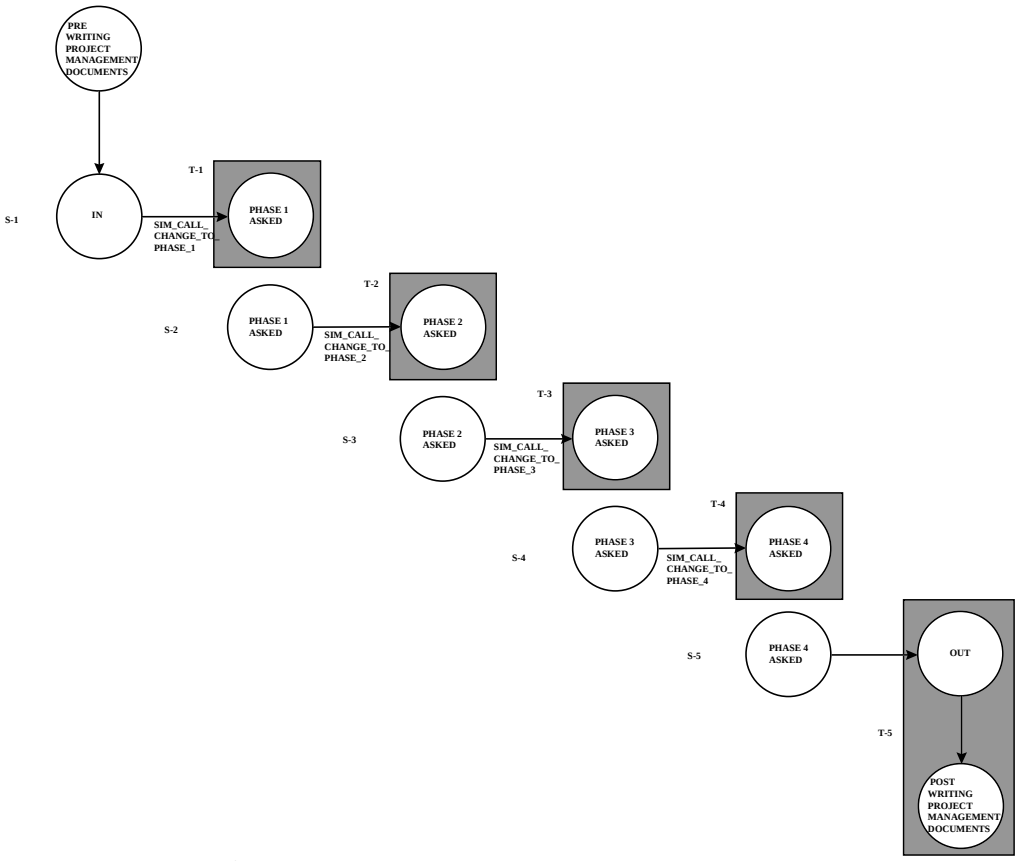


figure 6.148 employee int-pr_project_life_cycle : subprocesses S1, S2, S3, S4, S5 wrt_head_production_section

This operation ‘pr_project_life_cycle’ is not only an employee of ‘head production section’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1, S2, S3, S4 and S5. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses. The actual subprocesses of this

operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

The employee ‘phase_ended’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head production section’), according to the caller-callee construct.

The employee ‘pr_phase_ended’ is the caller of the callee ‘phase_ended’ of the class ‘head production section’. It has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head production section’) according to the caller_callee-construct.

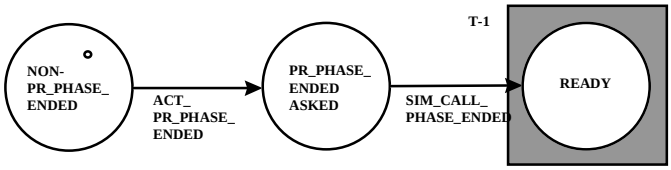


figure 6.149 employee int-pr_phase_ended : subprocess S1_wrt_head_production_section

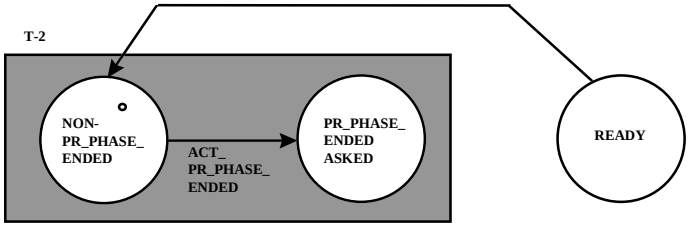


figure 6.150 employee int-pr_phase_ended : subprocess S2_wrt_head_production_section

This operation ‘pr_phase_ended’ is not only an employee of ‘head production section’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1 and S2. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses.

The actual subprocesses of this operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

6.3.2.15 Head support section (integration)

6.3.2.15.1 Head support section (integration) : extended phase-external behavior-STDs

The first step in the construction of the total external STD for the process fragment ‘writing project management documents’ is the extension of the constituent phase-external STDs with a start state and an end (final) state. The start state has a transition leaving it and entering the phase-external STD. The final state has a transition coming into it from the phase-external STD. This construction is performed in the next figures for the phase 1-external STD, the phase 2-external STD, the phase 3-external STD and the phase 4-external STD of the class ‘head support section’.

The class ‘head support section’ does not participate in phase 1 or 4. Therefore the phase 1- and phase 4-external STDs consist of a dummy state.

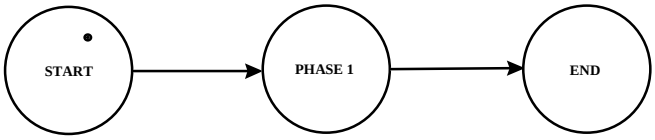


figure 6.151 head support section : extended phase 1-external STD

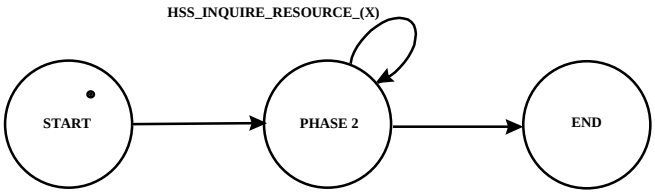


figure 6.152 head support section : extended phase 2-external STD

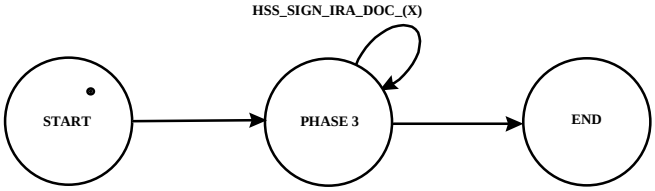


figure 6.153 head support section : extended phase 3-external STD

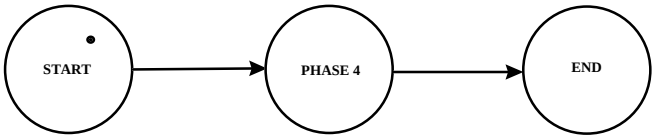


figure 6.154 head support section : extended phase 4-external STD

6.3.2.15.2 Head support section (integration) : total external behavior-STD

The second step in the construction of the total external STD for the process fragment ‘writing project management documents’ is to connect the extended phase-external STDs with each other in such a way that the final state of one phase-external STD coincides with the start state of the next phase-external STD. This ‘coinciding’ state between two phase-external STDs is called an ‘intermediate’ state. This models the sequential dependency between the phases.

The transitions leaving the start states get the label ‘change_to_phase_x’. These represent the ‘phase-changing’-operations. The transitions entering the final states get the label ‘pr_phase_ended’. These represent the ‘phase-ended’ operation. The construction is performed in the next figure.

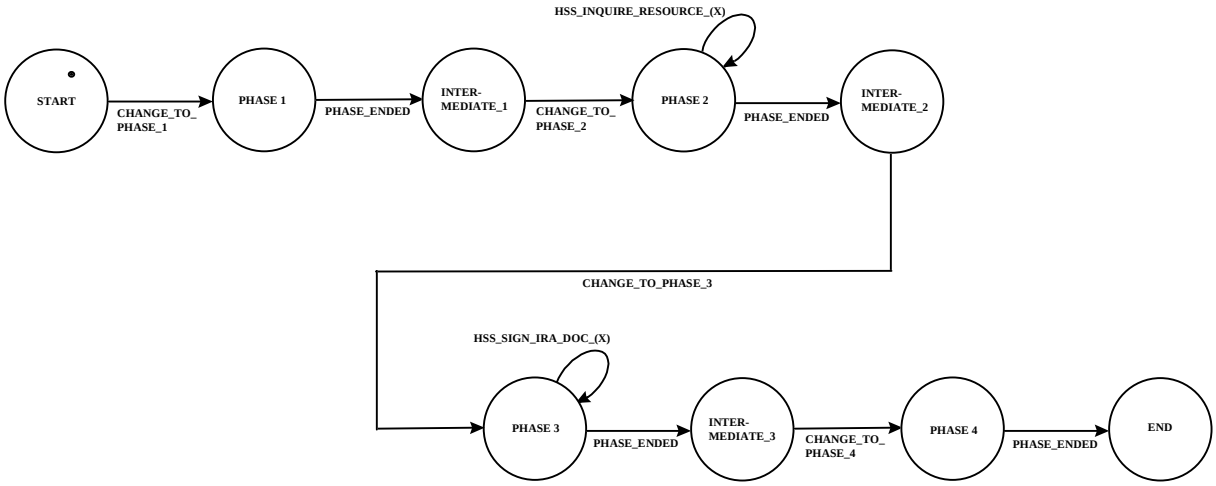


figure 6.155 head support section : ext_wpmd_head_support_section, total external STD

The flow through this total external STD is analogous to the flow through the total external STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : total external behavior-STD).

With the difference that the class ‘head support section’ does not participate in phase 1 or 4. Therefore no action of ‘head support section’ will take place in these phases.

6.3.2.15.3 Head support section (integration) : internal behavior-STDs

The internal operations of the class ‘head support section’ are in the first place the internal operations as modeled in the phase 2 and 3. During the integration the following internal operations are added to the class: ‘change_to_phase_1’, ‘change_to_ phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and ‘phase_ended’.

These 5 additional operations are so-called ‘no-operations’ (nops). A nop is an operation that performs no function while executing. The sole purpose of such an operation is to enable the corresponding transition of the external STD within the existing SOCCA framework.

These internal operations have internal STDs equivalent to the corresponding operations of the class ‘customer’ (described in the paragraph ‘Customer (integration) : internal behavior STDs’).

6.3.2.15.4 Head support section (integration) : manager-STD

The total manager STD is the same as the total external STD. Therefore no separate figure is needed for the total manager STD. The total manager STD of the process fragment ‘writing project management documents’ prescribes in its ‘phase 2’ and ‘phase 3’ states still the same subprocesses for its phase-employees as was modeled in the phase 2-, and phase 3-sub-models. These may be called ‘old prescribed subprocesses’. Subprocess prescriptions for an employee in states (of the total manager STD) that are not relevant for that employee are ‘don’t cares’.

In the integration the total manager gets new employees. For this new employees it is also prescribing subprocesses. These may be called ‘new prescribed subprocesses’. The sum total of the prescribed subprocesses by the total manager STD is therefore (per state) : {old prescribed subprocesses} $\cup$ {don’t cares} $\cup$ {new prescribed subprocesses}.

The new CPSs (Consolidated Prescribed Subprocesses), CCs (Caller_Callee combinations) and TLFs (Transition Logical Formulas) for this total manager STD are the same as the CPSs, CCs and TLFs for the total manager STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : manager-STD).

With the difference that all the subprocesses and traps are with respect to ‘head support section’ instead of with respect to ‘requirements document’.

6.3.2.15.5 Head support section (integration) : employee-STDs

The employees of the manager STD of the class ‘head support section’ are in the first place the employees as modeled in the phase 2 and 3. During the integration the following 7 employees are added: ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and their caller ‘pr_project_life_cycle’ and ‘phase_ended’ and its caller ‘pr_phase_ended’.

The employee ‘change_to_phase_1’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head support section’), according to the caller-callee construct.

The employee ‘change_to_phase_2’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head support section’), according to the caller-callee construct.

The employee ‘change_to_phase_3’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head support section’), according to the caller-callee construct.

The employee ‘change_to_phase_4’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head support section’), according to the caller-callee construct.

The employee ‘pr_project_life_cycle’ is the caller of the four callees ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’ and ‘change_to_phase_4’ of the class ‘head support section’. Consequently it has five subprocesses and traps with respect to the manager STD of the class ‘head support section’ : S1, S2, S3, S4 and S5 and T-1, T-2, T-3, T-4 and T-5. The next figure shows these five subprocesses of the internal operation ‘pr_project_ life_cycle’.

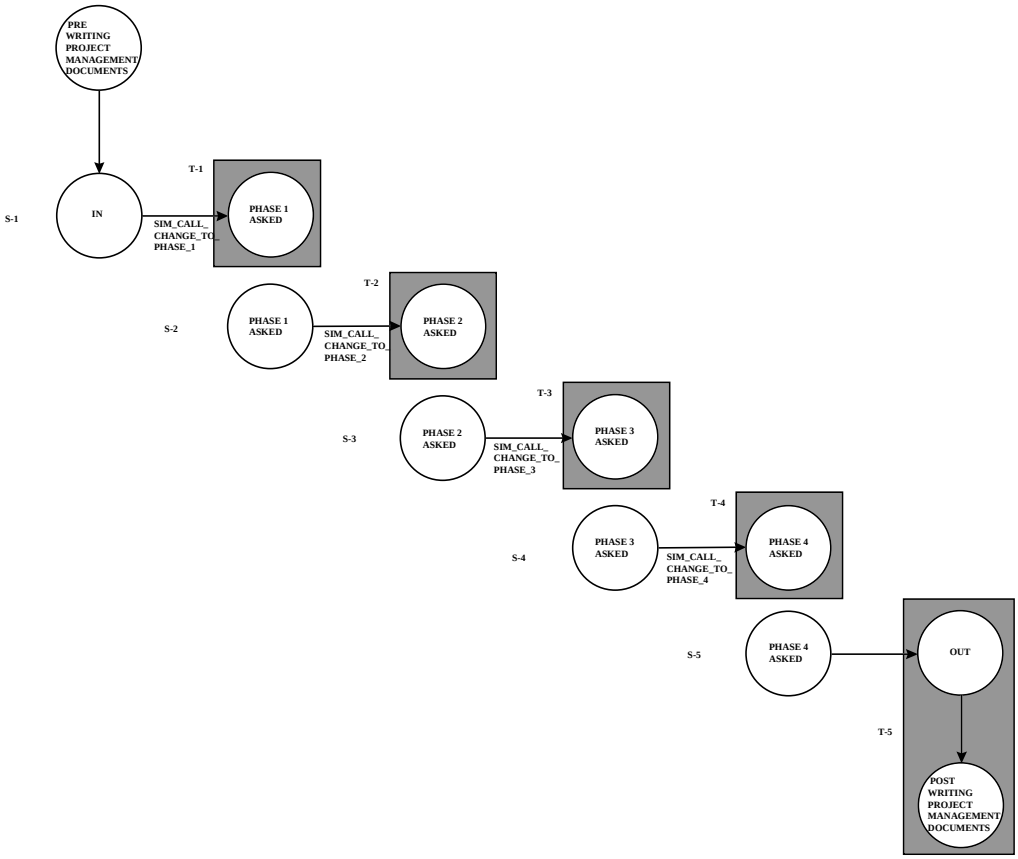


figure 6.156 employee int-pr_project_life_cycle : subprocesses S1, S2, S3, S4, S5 wrt_head_support_section

This operation ‘pr_project_life_cycle’ is not only an employee of ‘head support section’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1, S2, S3, S4 and S5. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses. The actual subprocesses of this operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

The employee ‘phase_ended’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head support section’), according to the caller-callee construct.

The employee ‘pr_phase_ended’ is the caller of the callee ‘phase_ended’ of the class ‘head support section’. It has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head support section’) according to the caller_callee-construct.

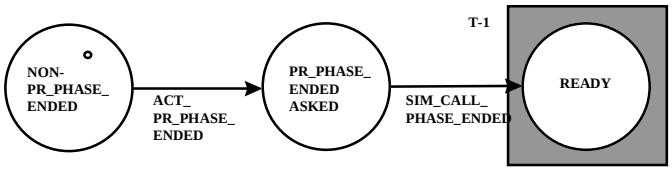


figure 6.157 employee int-pr_phase_ended : subprocess S1_wrt_head_support_section

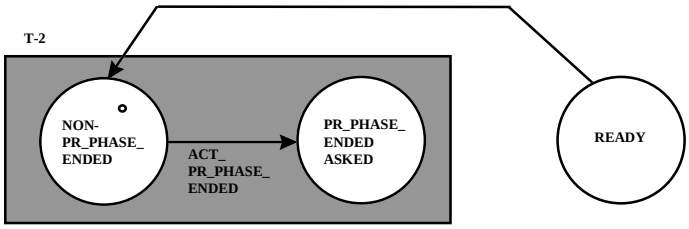


figure 6.158 employee int-pr_phase_ended : subprocess S2_wrt_head_support_section

This operation ‘pr_phase_ended’ is not only an employee of ‘head support section’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1 and S2. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses.

The actual subprocesses of this operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

6.3.2.16 Project management document (integration)

6.3.2.16.1 Project management document (integration) : extended phase-external behavior-STDs

The first step in the construction of the total external STD for the process fragment ‘writing project management documents’ is the extension of the constituent phase-external STDs with a start state and an end (final) state. The start state has a transition leaving it and entering the phase-external STD. The final state has a transition coming into it from the phase-external STD. This construction is performed in the next figures for the phase 1-external STD, the phase 2-external STD, the phase 3-external STD and the phase 4-external STD of the class ‘project management document’.

The class ‘project management document’ does not participate in phase 1, 3 or 4. Therefore the phase 1-, phase 3- and phase 4-external STDs consist of a dummy state.

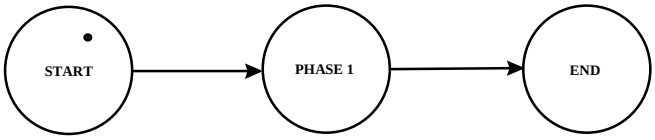


figure 6.159 project management document : extended phase 1-external STD

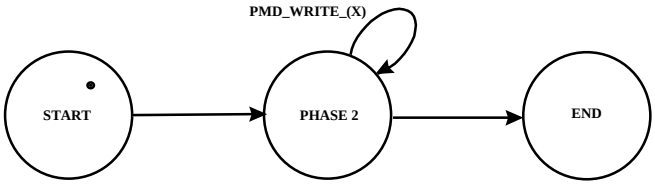


figure 6.160 project management document : extended phase 2-external STD

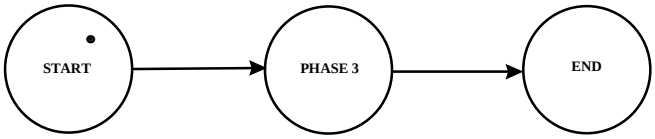


figure 6.161 project management document : extended phase 3-external STD

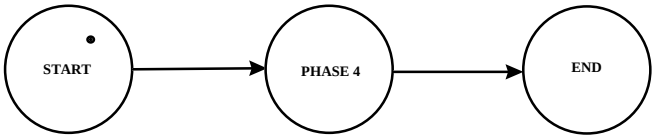


figure 6.162 project management document : extended phase 4-external STD

6.3.2.16.2 Project management document (integration) : total external behavior-STD

The second step in the construction of the total external STD for the process fragment ‘writing project management documents’ is to connect the extended phase-external STDs with each other in such a way that the final state of one phase-external STD coincides with the start state of the next phase-external STD. This ‘coinciding’ state between two phase-external STDs is called an ‘intermediate’ state. This models the sequential dependency between the phases.

The transitions leaving the start states get the label ‘change_to_phase_x’. These represent the ‘phase-changing’-operations. The transitions entering the final states get the label ‘pr_phase_ended’. These represent the ‘phase-ended’ operation. The construction is performed in the next figure.

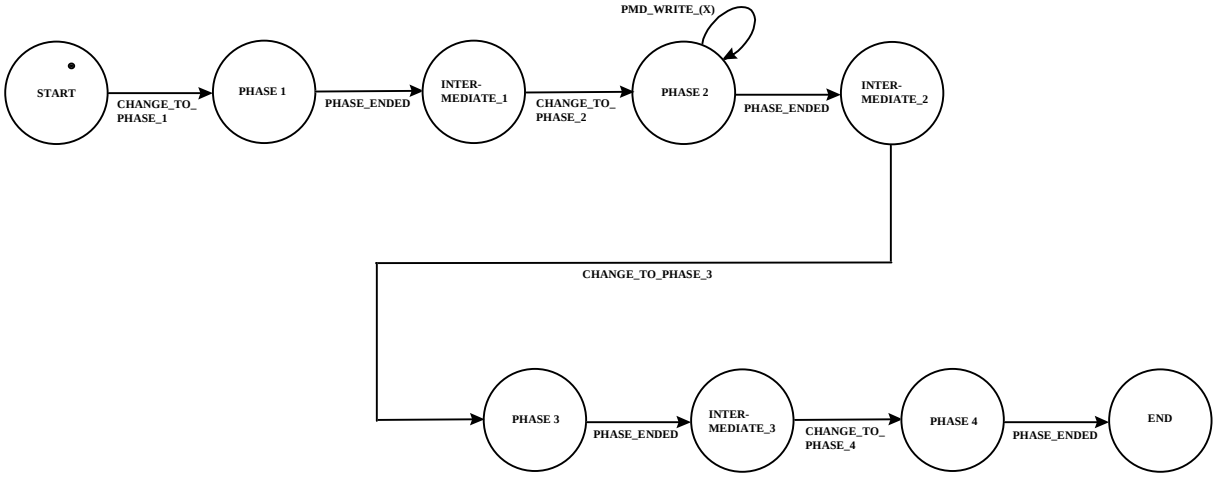


figure 6.163 project management document : ext_wpmd_project_management_document, total external STD

The flow through this total external STD is analogous to the flow through the total external STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : total external behavior-STD).

With the difference that the class ‘project management document’ does not participate in phase 1, 3 or 4. Therefore no action of ‘project management document’ will take place in these phases.

6.3.2.16.3 Project management document (integration) : internal behavior-STDs

The internal operations of the class ‘project management document’ are in the first place the internal operation as modeled in the phase 2. During the integration the following internal operations are added to the class: ‘change_to_phase_1’, ‘change_to_ phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and ‘phase_ended’.

These 5 additional operations are so-called ‘no-operations’ (nops). A nop is an operation that performs no function while executing. The sole purpose of such an operation is to enable the corresponding transition of the external STD within the existing SOCCA framework.

These internal operations have internal STDs equivalent to the corresponding operations of the class ‘customer’ (described in the paragraph ‘Customer (integration) : internal behavior STDs’).

6.3.2.16.4 Project management document (integration) : manager-STD

The total manager STD is the same as the total external STD. Therefore no separate figure is needed for the total manager STD. The total manager STD of the process fragment ‘writing project management documents’ prescribes in its ‘phase 2’ state still the same subprocesses for its phase-employees as was modeled in the phase 2-sub-model. These may be called ‘old prescribed subprocesses’. Subprocess prescriptions for an employee in states (of the total manager STD) that are not relevant for that employee are ‘don’t cares’.

In the integration the total manager gets new employees. For this new employees it is also prescribing subprocesses. These may be called ‘new prescribed subprocesses’. The sum total of the prescribed subprocesses by the total manager STD is therefore (per state) : {old prescribed subprocesses} $\cup$ {don’t cares} $\cup$ {new prescribed subprocesses}.

The new CPSs (Consolidated Prescribed Subprocesses), CCs (Caller_Callee combinations) and TLFs (Transition Logical Formulas) for this total manager STD are the same as the CPSs, CCs and TLFs for the total manager STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : manager-STD).

With the difference that all the subprocesses and traps are with respect to ‘project management document’ instead of with respect to ‘requirements document’.

6.3.2.16.5 Project management document (integration) : employee-STDs

The employees of the manager STD of the class ‘project management document’ are in the first place the employees as modeled in the phase 2. During the integration the following 7 employees are added: ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and their caller ‘pr_project_life_cycle’ and ‘phase_ended’ and its caller ‘pr_phase_ended’.

The employee ‘change_to_phase_1’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘project management document’), according to the caller-callee construct.

The employee ‘change_to_phase_2’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘project management document’), according to the caller-callee construct.

The employee ‘change_to_phase_3’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘project management document’), according to the caller-callee construct.

The employee ‘change_to_phase_4’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘project management document’), according to the caller-callee construct.

The employee ‘pr_project_life_cycle’ is the caller of the four callees ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’ and ‘change_to_phase_4’ of the class ‘project management document’. Consequently it has five subprocesses and traps with respect to the manager STD of the class ‘project management document’ : S1, S2, S3, S4 and S5 and T-1, T-2, T-3, T-4 and T-5. The next figure shows these five subprocesses of the internal operation ‘pr_project_ life_cycle’.

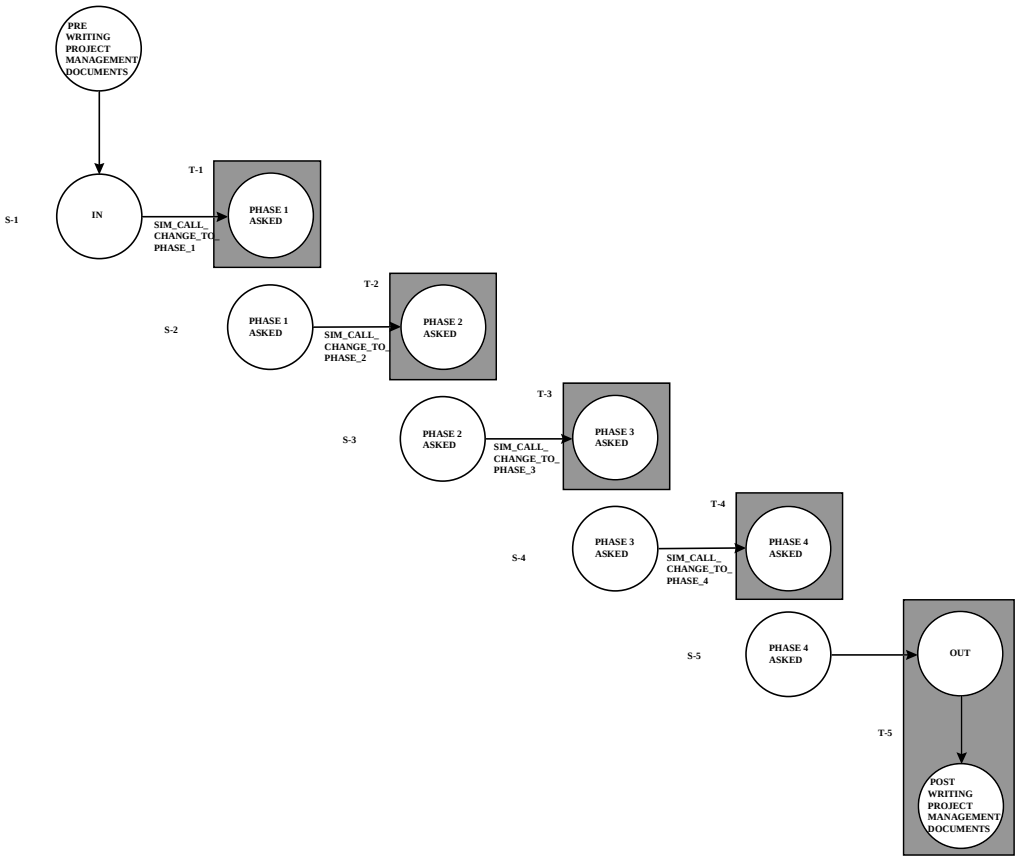


figure 6.164 employee int-pr_project_life_cycle : subprocesses S1, S2, S3, S4, S5 wrt_project_management_document

This operation ‘pr_project_life_cycle’ is not only an employee of ‘project management document’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1, S2, S3, S4 and S5. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses. The actual subprocesses of this

operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

The employee ‘phase_ended’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘project management document’), according to the caller-callee construct.

The employee ‘pr_phase_ended’ is the caller of the callee ‘phase_ended’ of the class ‘project management document’. It has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘project management document’) according to the caller_callee-construct.

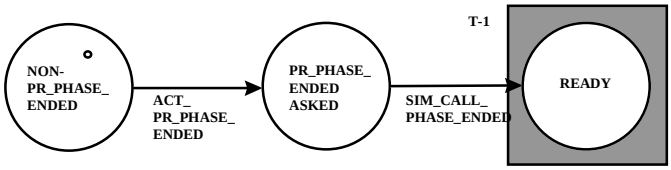


figure 6.165 employee int-pr_phase_ended : subprocess S1_wrt_project_management_document

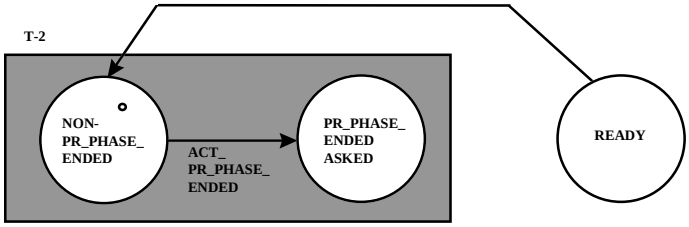


figure 6.166 employee int-pr_phase_ended : subprocess S2_wrt_project_management_document

This operation ‘pr_phase_ended’ is not only an employee of ‘project management document’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1 and S2. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses.

The actual subprocesses of this operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

6.3.2.17 Project meeting minus (integration)

6.3.2.17.1 Project meeting minus (integration) : extended phase-external behavior-STDs

The first step in the construction of the total external STD for the process fragment ‘writing project management documents’ is the extension of the constituent phase-external STDs with a start state and an end (final) state. The start state has a transition leaving it and entering the phase-external STD. The final state has a transition coming into it from the phase-external STD. This construction is performed in the next figures for the phase 1-external STD, the phase 2-external STD, the phase 3-external STD and the phase 4-external STD of the class ‘project meeting minus’.

The class ‘project meeting minus’ does not participate in phase 1, 2 or 4. Therefore the phase 1-, phase 2- and phase 4-external STDs consist of a dummy state.

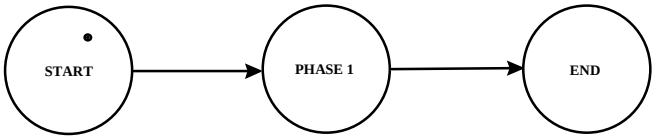


figure 6.167 project meeting minus : extended phase 1-external STD

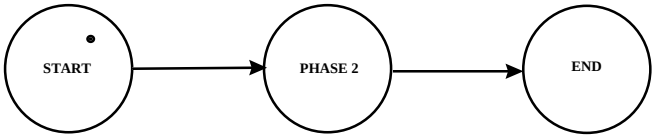


figure 6.168 project meeting minus : extended phase 2-external STD

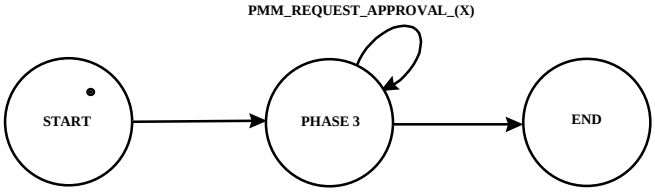


figure 6.169 project meeting minus : extended phase 3-external STD

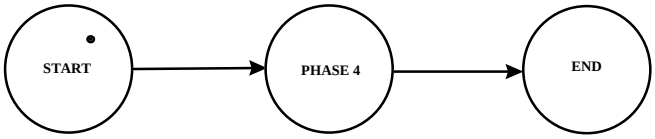


figure 6.170 project meeting minus : extended phase 4-external STD

6.3.2.17.2 Project meeting minus (integration) : total external behavior-STD

The second step in the construction of the total external STD for the process fragment ‘writing project management documents’ is to connect the extended phase-external STDs with each other in such a way that the final state of one phase-external STD coincides with the start state of the next phase-external STD. This ‘coinciding’ state between two phase-external STDs is called an ‘intermediate’ state. This models the sequential dependency between the phases.

The transitions leaving the start states get the label ‘change_to_phase_x’. These represent the ‘phase-changing’-operations. The transitions entering the final states get the label ‘pr_phase_ended’. These represent the ‘phase-ended’ operation. The construction is performed in the next figure.

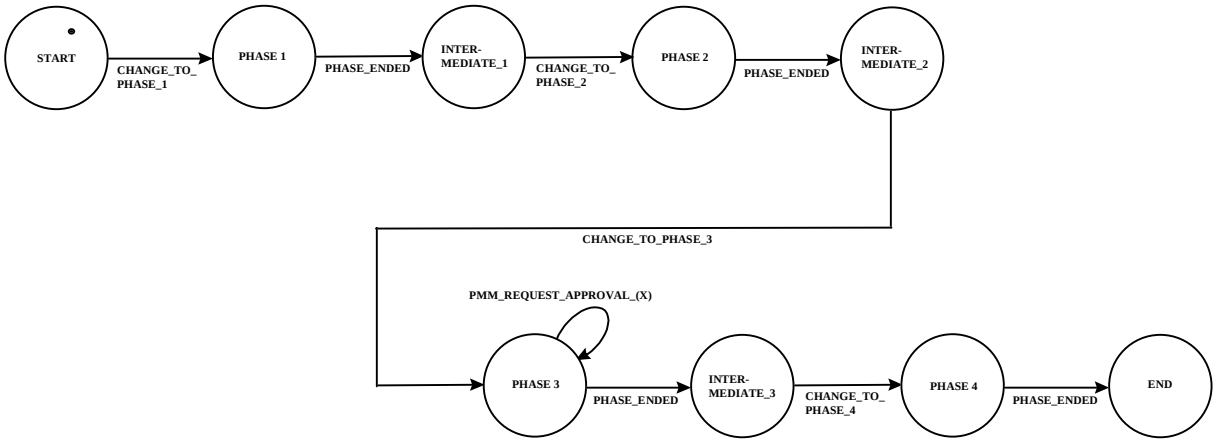


figure 6.171 project meeting minus : ext_wpmd_project_meeting_minus, total external STD

The flow through this total external STD is analogous to the flow through the total external STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : total external behavior-STD).

With the difference that the class ‘project meeting minus’ does not participate in phase 1, 2 or 4. Therefore no action of ‘project meeting minus’ will take place in these phases.

6.3.2.17.3 Project meeting minus (integration) : internal behavior-STDs

The internal operations of the class ‘project meeting minus’ are in the first place the internal operation as modeled in the phase 3. During the integration the following internal operations are added to the class: ‘change_to_phase_1’, ‘change_to_ phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and ‘phase_ended’.

These 5 additional operations are so-called ‘no-operations’ (nops). A nop is an operation that performs no function while executing. The sole purpose of such an operation is to enable the corresponding transition of the external STD within the existing SOCCA framework.

These internal operations have internal STDs equivalent to the corresponding operations of the class ‘customer’ (described in the paragraph ‘Customer (integration) : internal behavior STDs’).

6.3.2.17.4 Project meeting minus (integration) : manager-STD

The total manager STD is the same as the total external STD. Therefore no separate figure is needed for the total manager STD. The total manager STD of the process fragment ‘writing project management documents’ prescribes in its ‘phase 3’ state still the same subprocesses for its phase-employees as was modeled in the phase 3-sub-model. These may be called ‘old prescribed subprocesses’. Subprocess prescriptions for an employee in states (of the total manager STD) that are not relevant for that employee are ‘don’t cares’.

In the integration the total manager gets new employees. For this new employees it is also prescribing subprocesses. These may be called ‘new prescribed subprocesses’. The sum total of the prescribed subprocesses by the total manager STD is therefore (per state) : {old prescribed subprocesses} $\cup$ {don’t cares} $\cup$ {new prescribed subprocesses}.

The new CPSs (Consolidated Prescribed Subprocesses), CCs (Caller_Callee combinations) and TLFs (Transition Logical Formulas) for this total manager STD are the same as the CPSs, CCs and TLFs for the total manager STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : manager-STD).

With the difference that all the subprocesses and traps are with respect to ‘project meeting minus’ instead of with respect to ‘requirements document’.

6.3.2.17.5 Project meeting minus (integration) : employee-STDs

The employees of the manager STD of the class ‘project meeting minus’ are in the first place the employees as modeled in the phase 3. During the integration the following 7 employees are added: ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and their caller ‘pr_project_life_cycle’ and ‘phase_ended’ and its caller ‘pr_phase_ended’.

The employee ‘change_to_phase_1’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘project meeting minus’), according to the caller-callee construct.

The employee ‘change_to_phase_2’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘project meeting minus’), according to the caller-callee construct.

The employee ‘change_to_phase_3’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘project meeting minus’), according to the caller-callee construct.

The employee ‘change_to_phase_4’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘project meeting minus’), according to the caller-callee construct.

The employee ‘pr_project_life_cycle’ is the caller of the four callees ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’ and ‘change_to_phase_4’ of the class ‘project meeting minus’. Consequently it has five subprocesses and traps with respect to the manager STD of the class ‘project meeting minus’ : S1, S2, S3, S4 and S5 and T-1, T-2, T-3, T-4 and T-5. The next figure shows these five subprocesses of the internal operation ‘pr_project_ life_cycle’.

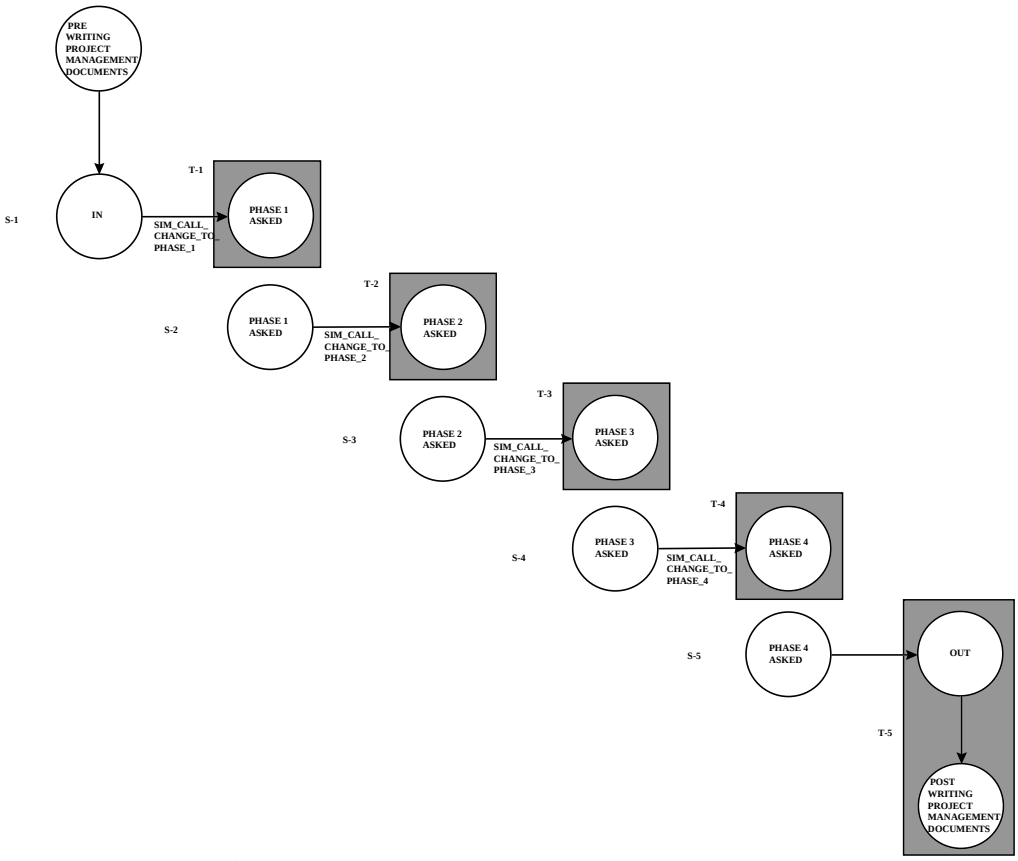


figure 6.172 employee int-pr_project_life_cycle : subprocesses S1, S2, S3, S4, S5 wrt_project_meeting_minus

This operation ‘pr_project_life_cycle’ is not only an employee of ‘project meeting minus’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1, S2, S3, S4 and S5. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses. The actual subprocesses of this

operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

The employee ‘phase_ended’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘project meeting minus’), according to the caller-callee construct.

The employee ‘pr_phase_ended’ is the caller of the callee ‘phase_ended’ of the class ‘project meeting minus’. It has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘project meeting minus’) according to the caller_callee-construct.

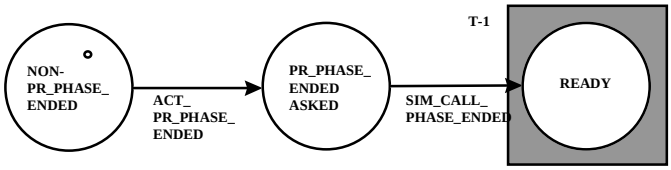


figure 6.173 employee int-pr_phase_ended : subprocess S1_wrt_project_meeting_minus

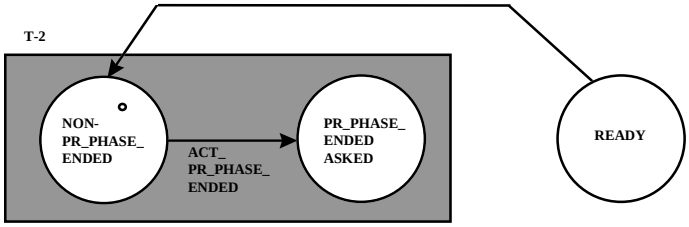


figure 6.174 employee int-pr_phase_ended : subprocess S2_wrt_project_meeting_minus

This operation ‘pr_phase_ended’ is not only an employee of ‘project meeting minus’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1 and S2. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses.

The actual subprocesses of this operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

6.3.2.18 Archive/documentation administrator (integration)

6.3.2.18.1 Archive/doc administrator (integration) : extended phase-external behavior-STDs

The first step in the construction of the total external STD for the process fragment ‘writing project management documents’ is the extension of the constituent phase-external STDs with a start state and an end (final) state. The start state has a transition leaving it and entering the phase-external STD. The final state has a transition coming into it from the phase-external STD. This construction is performed in the next figures for the phase 1-external STD, the phase 2-external STD, the phase 3-external STD and the phase 4-external STD of the class ‘archive/documentation administrator’.

The class ‘archive/documentation administrator’ does not participate in phase 1, 2 or 4. Therefore the phase 1-, phase 2- and phase 4-external STDs consist of a dummy state.

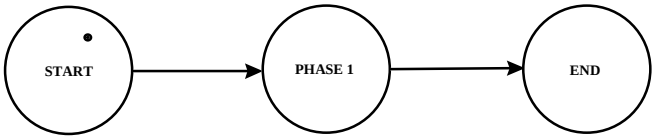


figure 6.175 archive/documentation administrator : extended phase 1-external STD

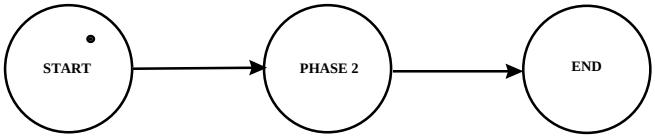


figure 6.176 archive/documentation administrator : extended phase 2-external STD

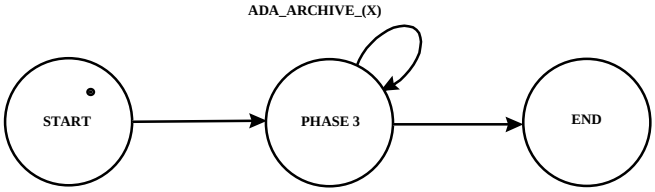


figure 6.177 archive/documentation administrator : extended phase 3-external STD

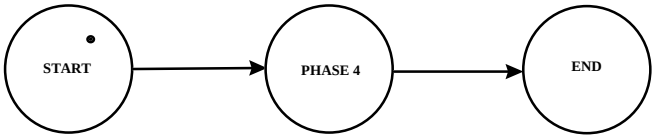


figure 6.178 archive/documentation administrator : extended phase 4-external STD

6.3.2.18.2 Archive/documentation administrator (integration) : total external behavior-STD

The second step in the construction of the total external STD for the process fragment ‘writing project management documents’ is to connect the extended phase-external STDs with each other in such a way that the final state of one phase-external STD coincides with the start state of the next phase-external STD. This ‘coinciding’ state between two phase-external STDs is called an ‘intermediate’ state. This models the sequential dependency between the phases.

The transitions leaving the start states get the label ‘change_to_phase_x’. These represent the ‘phase-changing’-operations. The transitions entering the final states get the label ‘pr_phase_ended’. These represent the ‘phase-ended’ operation. The construction is performed in the next figure.

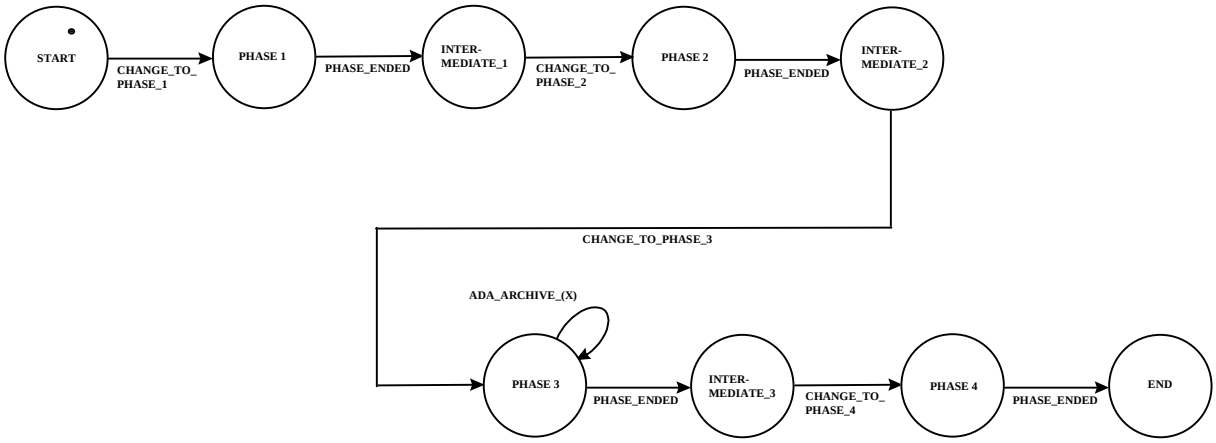


figure 6.179 archive/documentation administrator : ext_wpmd_archive/documentation_administrator, total external STD

The flow through this total external STD is analogous to the flow through the total external STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : total external behavior-STD).

With the difference that the class ‘archive/documentation administrator’ does not participate in phase 1, 2 or 4. Therefore no action of ‘archive/documentation administrator’ will take place in these phases.

6.3.2.18.3 Archive/documentation administrator (integration) : internal behavior-STDs

The internal operations of the class ‘archive/documentation administrator’ are in the first place the internal operation as modeled in the phase 3. During the integration the following internal operations are added to the class: ‘change_to_phase_1’, ‘change_to_ phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and ‘phase_ended’.

These 5 additional operations are so-called ‘no-operations’ (nops). A nop is an operation that performs no function while executing. The sole purpose of such an operation is to enable the corresponding transition of the external STD within the existing SOCCA framework.

These internal operations have internal STDs equivalent to the corresponding operations of the class ‘customer’ (described in the paragraph ‘Customer (integration) : internal behavior STDs’).

6.3.2.18.4 Archive/documentation administrator (integration) : manager-STD

The total manager STD is the same as the total external STD. Therefore no separate figure is needed for the total manager STD. The total manager STD of the process fragment ‘writing project management documents’ prescribes in its ‘phase 3’ state still the same subprocesses for its phase-employees as was modeled in the phase 3-sub-model. These may be called ‘old prescribed subprocesses’. Subprocess prescriptions for an employee in states (of the total manager STD) that are not relevant for that employee are ‘don’t cares’.

In the integration the total manager gets new employees. For this new employees it is also prescribing subprocesses. These may be called ‘new prescribed subprocesses’. The sum total of the prescribed subprocesses by the total manager STD is therefore (per state) : {old prescribed subprocesses} $\cup$ {don’t cares} $\cup$ {new prescribed subprocesses}.

The new CPSs (Consolidated Prescribed Subprocesses), CCs (Caller_Callee combinations) and TLFs (Transition Logical Formulas) for this total manager STD are the same as the CPSs, CCs and TLFs for the total manager STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : manager-STD).

With the difference that all the subprocesses and traps are with respect to ‘archive/documentation administrator’ instead of with respect to ‘requirements document’.

6.3.2.18.5 Archive/documentation administrator (integration) : employee-STDs

The employees of the manager STD of the class ‘archive/documentation administrator’ are in the first place the employees as modeled in the phase 3. During the integration the following 7 employees are added: ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and their caller ‘pr_project_life_cycle’ and ‘phase_ended’ and its caller ‘pr_phase_ended’.

The employee ‘change_to_phase_1’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘archive/documentation administrator’), according to the caller-callee construct.

The employee ‘change_to_phase_2’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘archive/documentation administrator’), according to the caller-callee construct.

The employee ‘change_to_phase_3’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘archive/documentation administrator’), according to the caller-callee construct.

The employee ‘change_to_phase_4’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘archive/documentation administrator’), according to the caller-callee construct.

The employee ‘pr_project_life_cycle’ is the caller of the four callees ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’ and ‘change_to_phase_4’ of the class ‘archive/documentation administrator’. Consequently it has five subprocesses and traps with respect to the manager STD of the class ‘archive/documentation administrator’ : S1, S2, S3, S4 and S5 and T-1, T-2, T-3, T-4 and T-5. The next figure shows these five subprocesses of the internal operation ‘pr_project_ life_cycle’.

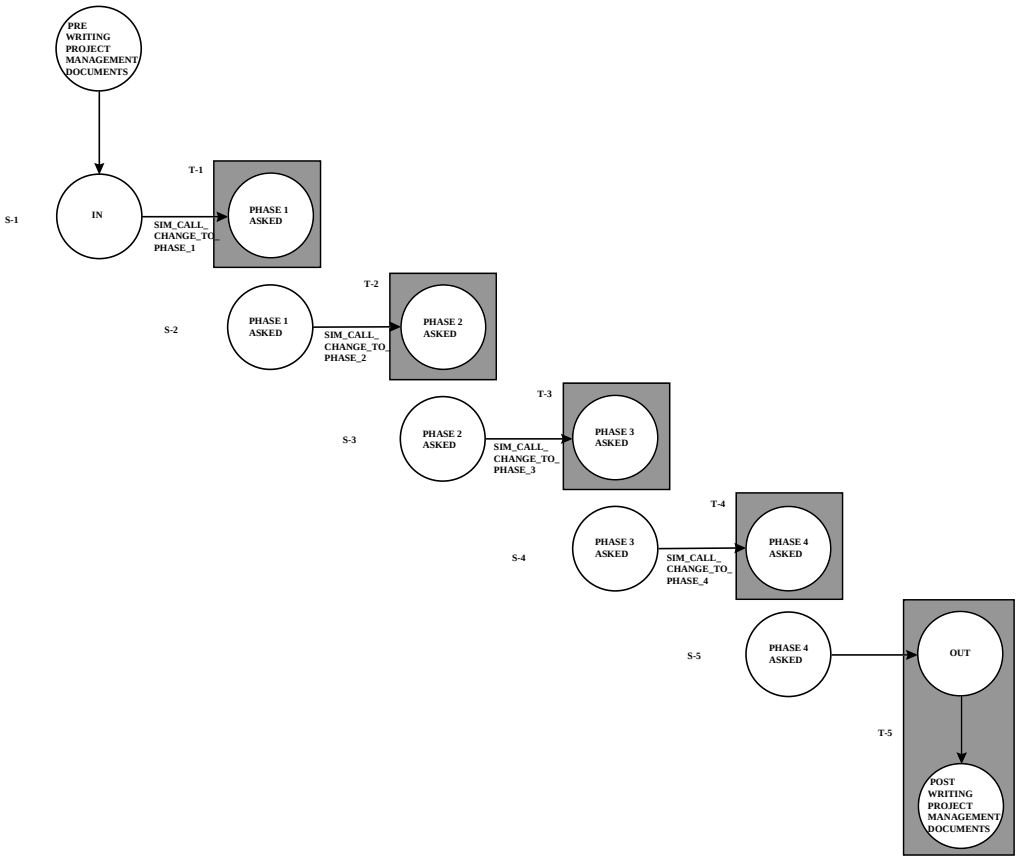


figure 6.180 employee int-pr_project_life_cycle : subprocesses S1, S2, S3, S4, S5 wrt_archive/documentation_administrator

This operation ‘pr_project_life_cycle’ is not only an employee of ‘archive/documentation administrator’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1, S2, S3, S4 and S5. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses. The actual subprocesses

of this operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

The employee ‘phase_ended’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘archive/documentation administrator’), according to the caller-callee construct.

The employee ‘pr_phase_ended’ is the caller of the callee ‘phase_ended’ of the class ‘archive/documentation administrator’. It has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘archive/ documentation administrator’) according to the caller_callee-construct.

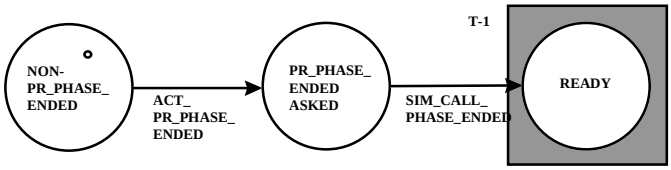


figure 6.181 employee int-pr_phase_ended : subprocess S1_wrt_archive/documentation_administrator

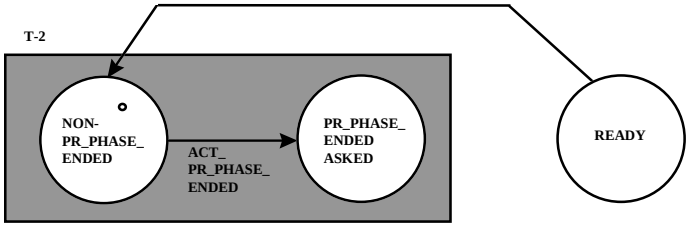


figure 6.182 employee int-pr_phase_ended : subprocess S2_wrt_archive/documentation_administrator

This operation ‘pr_phase_ended’ is not only an employee of ‘archive/documentation administrator’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1 and S2. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses.

The actual subprocesses of this operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

6.3.2.19 Head computer support section (integration)

6.3.2.19.1 Head computer support section (integration) : extended phase-external behavior-STDs

The first step in the construction of the total external STD for the process fragment ‘writing project management documents’ is the extension of the constituent phase-external STDs with a start state and an end (final) state. The start state has a transition leaving it and entering the phase-external STD. The final state has a transition coming into it from the phase-external STD. This construction is performed in the next figures for the phase 1-external STD, the phase 2-external STD, the phase 3-external STD and the phase 4-external STD of the class ‘head computer support section’.

The class ‘head computer support section’ does not participate in phase 1, 2 or 3. Therefore the phase 1-, phase 2- and phase 3-external STDs consist of a dummy state.

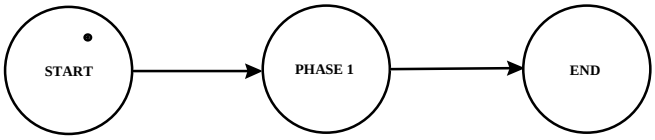


figure 6.183 head computer support section : extended phase 1-external STD

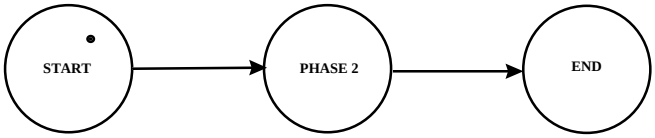


figure 6.184 head computer support section : extended phase 2-external STD

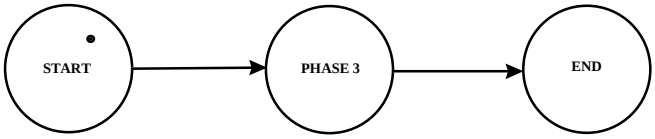


figure 6.185 head computer support section : extended phase 3-external STD

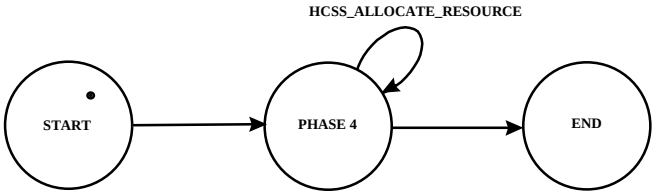


figure 6.186 head computer support section : extended phase 4-external STD

6.3.2.19.2 Head computer support section (integration) : total external behavior-STD

The second step in the construction of the total external STD for the process fragment ‘writing project management documents’ is to connect the extended phase-external STDs with each other in such a way that the final state of one phase-external STD coincides with the start state of the next phase-external STD. This ‘coinciding’ state between two phase-external STDs is called an ‘intermediate’ state. This models the sequential dependency between the phases.

The transitions leaving the start states get the label ‘change_to_phase_x’. These represent the ‘phase-changing’-operations. The transitions entering the final states get the label ‘pr_phase_ended’. These represent the ‘phase-ended’ operation. The construction is performed in the next figure.

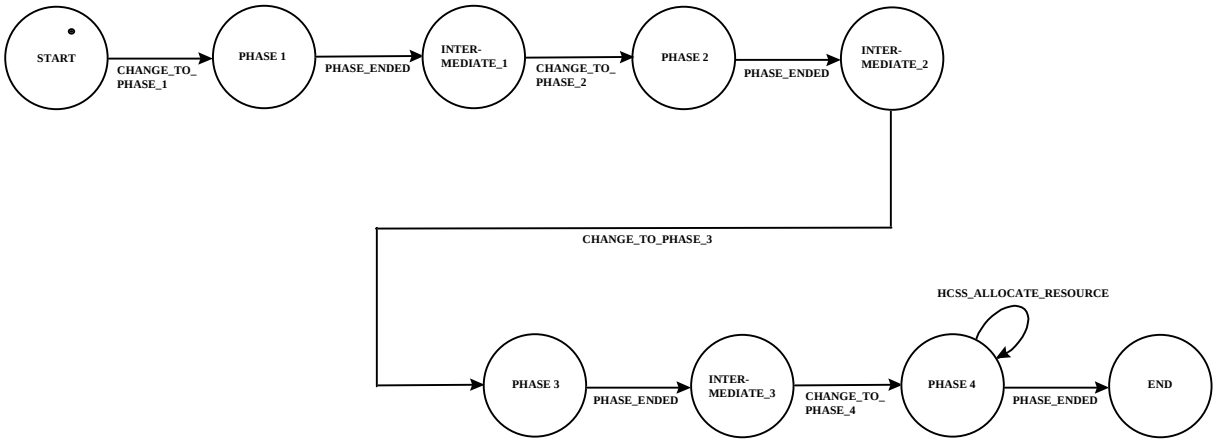


figure 6.187 head computer support section : ext_wpmd_head_computer_support_section, total external STD

The flow through this total external STD is analogous to the flow through the total external STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : total external behavior-STD).

With the difference that the class ‘head computer support section’ does not participate in phase 1, 2 or 3. Therefore no action of ‘head computer support section’ will take place in these phases.

6.3.2.19.3 Head computer support section (integration) : internal behavior-STDs

The internal operations of the class ‘head computer support section’ are in the first place the internal operation as modeled in the phase 4. During the integration the following internal operations are added to the class: ‘change_to_phase_1’, ‘change_to_ phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and ‘phase_ended’.

These 5 additional operations are so-called ‘no-operations’ (nops). A nop is an operation that performs no function while executing. The sole purpose of such an operation is to enable the corresponding transition of the external STD within the existing SOCCA framework.

These internal operations have internal STDs equivalent to the corresponding operations of the class ‘customer’ (described in the paragraph ‘Customer (integration) : internal behavior STDs’).

6.3.2.19.4 Head computer support section (integration) : manager-STD

The total manager STD is the same as the total external STD. Therefore no separate figure is needed for the total manager STD. The total manager STD of the process fragment ‘writing project management documents’ prescribes in its ‘phase 4’ state still the same subprocesses for its phase-employees as was modeled in the phase 4-sub-model. These may be called ‘old prescribed subprocesses’. Subprocess prescriptions for an employee in states (of the total manager STD) that are not relevant for that employee are ‘don’t cares’.

In the integration the total manager gets new employees. For this new employees it is also prescribing subprocesses. These may be called ‘new prescribed subprocesses’. The sum total of the prescribed subprocesses by the total manager STD is therefore (per state) : {old prescribed subprocesses} $\cup$ {don’t cares} $\cup$ {new prescribed subprocesses}.

The new CPSs (Consolidated Prescribed Subprocesses), CCs (Caller_Callee combinations) and TLFs (Transition Logical Formulas) for this total manager STD are the same as the CPSs, CCs and TLFs for the total manager STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : manager-STD).

With the difference that all the subprocesses and traps are with respect to ‘head computer support section’ instead of with respect to ‘requirements document’.

6.3.2.19.5 Head computer support section (integration) : employee-STDs

The employees of the manager STD of the class ‘head computer support section’ are in the first place the employees as modeled in the phase 4. During the integration the following 7 employees are added: ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and their caller ‘pr_project_life_cycle’ and ‘phase_ended’ and its caller ‘pr_phase_ended’.

The employee ‘change_to_phase_1’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head computer support section’), according to the caller-callee construct.

The employee ‘change_to_phase_2’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head computer support section’), according to the caller-callee construct.

The employee ‘change_to_phase_3’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head computer support section’), according to the caller-callee construct.

The employee ‘change_to_phase_4’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head computer support section’), according to the caller-callee construct.

The employee ‘pr_project_life_cycle’ is the caller of the four callees ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’ and ‘change_to_phase_4’ of the class ‘head computer support section’. Consequently it has five subprocesses and traps with respect to the manager STD of the class ‘head computer support section’ : S1, S2, S3, S4 and S5 and T-1, T-2, T-3, T-4 and T-5. The next figure shows these five subprocesses of the internal operation ‘pr_project_ life_cycle’.

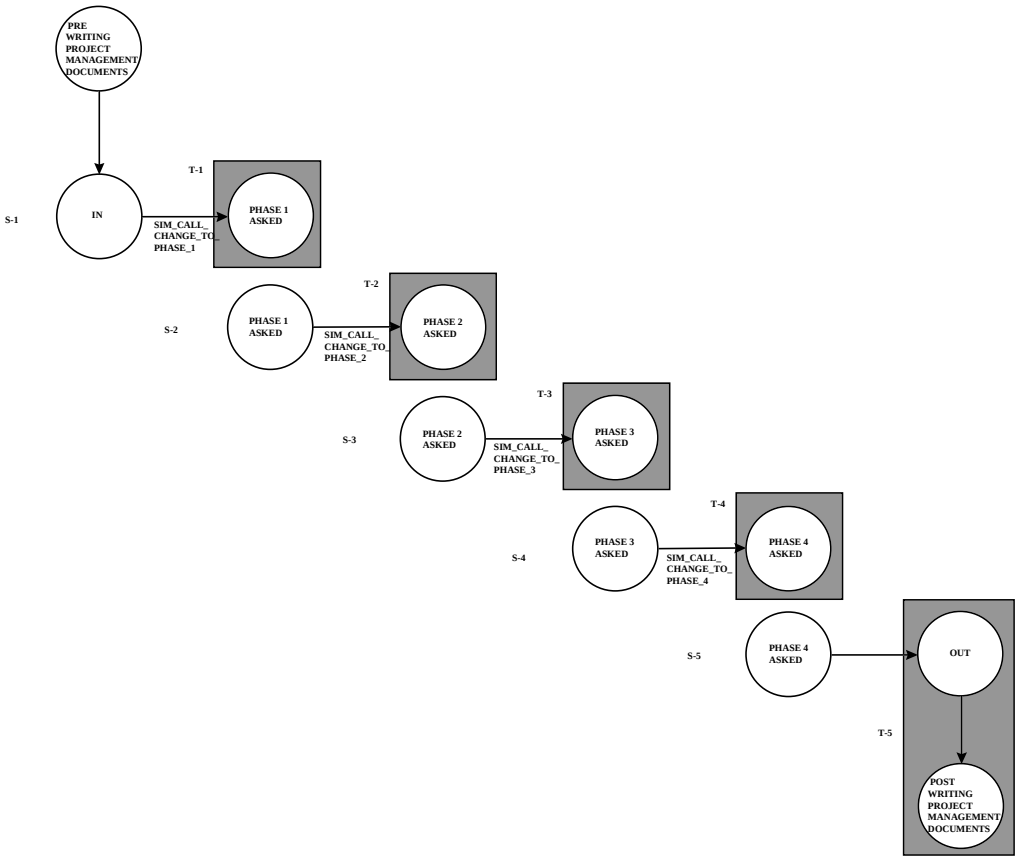


figure 6.188 employee int-pr_project_life_cycle : subprocesses S1, S2, S3, S4, S5 wrt_head_computer_support_section

This operation ‘pr_project_life_cycle’ is not only an employee of ‘head computer support section’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1, S2, S3, S4 and S5. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses. The actual subprocesses of this

operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

The employee ‘phase_ended’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head computer support section’), according to the caller-callee construct.

The employee ‘pr_phase_ended’ is the caller of the callee ‘phase_ended’ of the class ‘head computer support section’. It has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘head computer support section’) according to the caller_callee-construct.

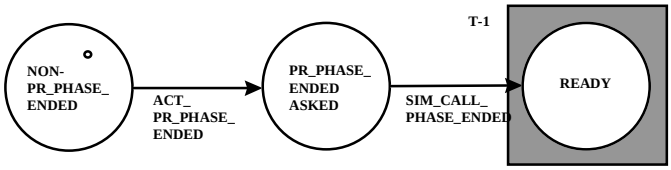


figure 6.189 employee int-pr_phase_ended : subprocess S1_wrt_head_computer_support_section

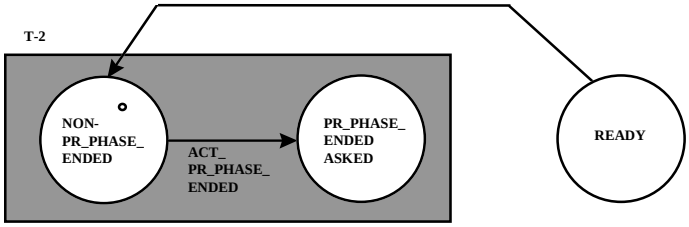


figure 6.190 employee int-pr_phase_ended : subprocess S2_wrt_head_computer_support_section

This operation ‘pr_phase_ended’ is not only an employee of ‘head computer support section’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1 and S2. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses.

The actual subprocesses of this operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

6.3.2.20 Terms of reference document (integration)

6.3.2.20.1 Terms of reference document (integration) : extended phase-external behavior-STDs

The first step in the construction of the total external STD for the process fragment ‘writing project management documents’ is the extension of the constituent phase-external STDs with a start state and an end (final) state. The start state has a transition leaving it and entering the phase-external STD. The final state has a transition coming into it from the phase-external STD. This construction is performed in the next figures for the phase 1-external STD, the phase 2-external STD, the phase 3-external STD and the phase 4-external STD of the class ‘terms of reference document’.

The class ‘terms of reference document’ does not participate in phase 1, 2 or 3. Therefore the phase 1-, phase 2- and phase 3-external STDs consist of a dummy state.

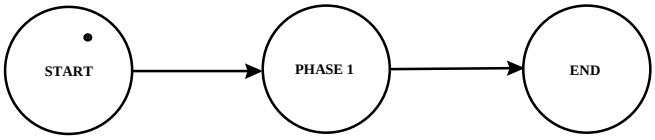


figure 6.191 terms of reference document : extended phase 1-external STD

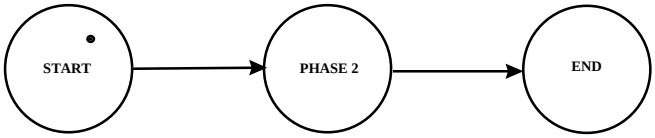


figure 6.192 terms of reference document : extended phase 2-external STD

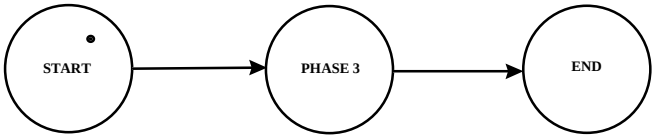


figure 6.193 terms of reference document : extended phase 3-external STD

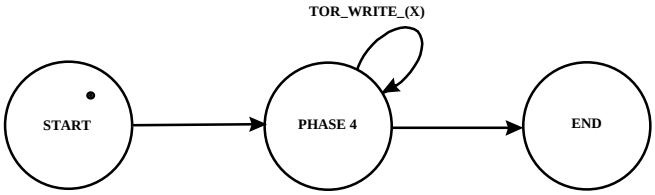


figure 6.194 terms of reference document : extended phase 4-external STD

6.3.2.20.2 Terms of reference document (integration) : total external behavior-STD

The second step in the construction of the total external STD for the process fragment ‘writing project management documents’ is to connect the extended phase-external STDs with each other in such a way that the final state of one phase-external STD coincides with the start state of the next phase-external STD. This ‘coinciding’ state between two phase-external STDs is called an ‘intermediate’ state. This models the sequential dependency between the phases.

The transitions leaving the start states get the label ‘change_to_phase_x’. These represent the ‘phase-changing’-operations. The transitions entering the final states get the label ‘pr_phase_ended’. These represent the ‘phase-ended’ operation. The construction is performed in the next figure.

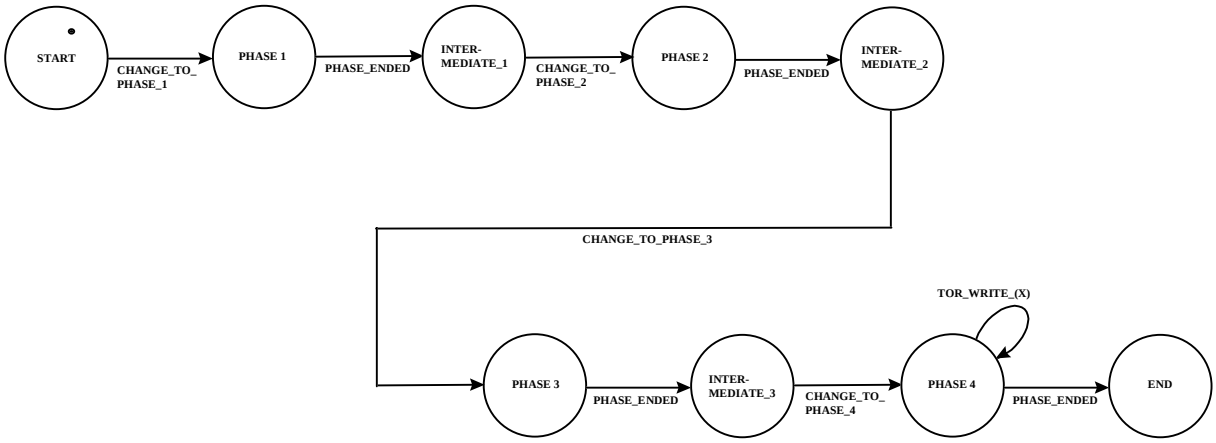


figure 6.195 terms of reference document : ext_wpmd_terms_of_reference_document, total external STD

The flow through this total external STD is analogous to the flow through the total external STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : total external behavior-STD).

With the difference that the class ‘terms of reference document’ does not participate in phase 1, 2 or 3. Therefore no action of ‘terms of reference document’ will take place in these phases.

6.3.2.20.3 Terms of reference document (integration) : internal behavior-STDs

The internal operations of the class ‘terms of reference document’ are in the first place the internal operation as modeled in the phase 4. During the integration the following internal operations are added to the class: ‘change_to_phase_1’, ‘change_to_ phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and ‘phase_ended’.

These 5 additional operations are so-called ‘no-operations’ (nops). A nop is an operation that performs no function while executing. The sole purpose of such an operation is to enable the corresponding transition of the external STD within the existing SOCCA framework.

These internal operations have internal STDs equivalent to the corresponding operations of the class ‘customer’ (described in the paragraph ‘Customer (integration) : internal behavior STDs’).

6.3.2.20.4 Terms of reference document (integration) : manager-STD

The total manager STD is the same as the total external STD. Therefore no separate figure is needed for the total manager STD. The total manager STD of the process fragment ‘writing project management documents’ prescribes in its ‘phase 4’ state still the same subprocesses for its phase-employees as was modeled in the phase 4-sub-model. These may be called ‘old prescribed subprocesses’. Subprocess prescriptions for an employee in states (of the total manager STD) that are not relevant for that employee are ‘don’t cares’.

In the integration the total manager gets new employees. For this new employees it is also prescribing subprocesses. These may be called ‘new prescribed subprocesses’. The sum total of the prescribed subprocesses by the total manager STD is therefore (per state) : {old prescribed subprocesses} $\cup$ {don’t cares} $\cup$ {new prescribed subprocesses}.

The new CPSs (Consolidated Prescribed Subprocesses), CCs (Caller_Callee combinations) and TLFs (Transition Logical Formulas) for this total manager STD are the same as the CPSs, CCs and TLFs for the total manager STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : manager-STD).

With the difference that all the subprocesses and traps are with respect to ‘terms of reference document’ instead of with respect to ‘requirements document’.

6.3.2.20.5 Terms of reference document (integration) : employee-STDs

The employees of the manager STD of the class ‘terms of reference document’ are in the first place the employees as modeled in the phase 4. During the integration the following 7 employees are added: ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and their caller ‘pr_project_life_cycle’ and ‘phase_ended’ and its caller ‘pr_phase_ended’.

The employee ‘change_to_phase_1’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘terms of reference document’), according to the caller-callee construct.

The employee ‘change_to_phase_2’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘terms of reference document’), according to the caller-callee construct.

The employee ‘change_to_phase_3’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘terms of reference document’), according to the caller-callee construct.

The employee ‘change_to_phase_4’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘terms of reference document’), according to the caller-callee construct.

The employee ‘pr_project_life_cycle’ is the caller of the four callees ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’ and ‘change_to_phase_4’ of the class ‘terms of reference document’. Consequently it has five subprocesses and traps with respect to the manager STD of the class ‘terms of reference document’ : S1, S2, S3, S4 and S5 and T-1, T-2, T-3, T-4 and T-5. The next figure shows these five subprocesses of the internal operation ‘pr_project_ life_cycle’.

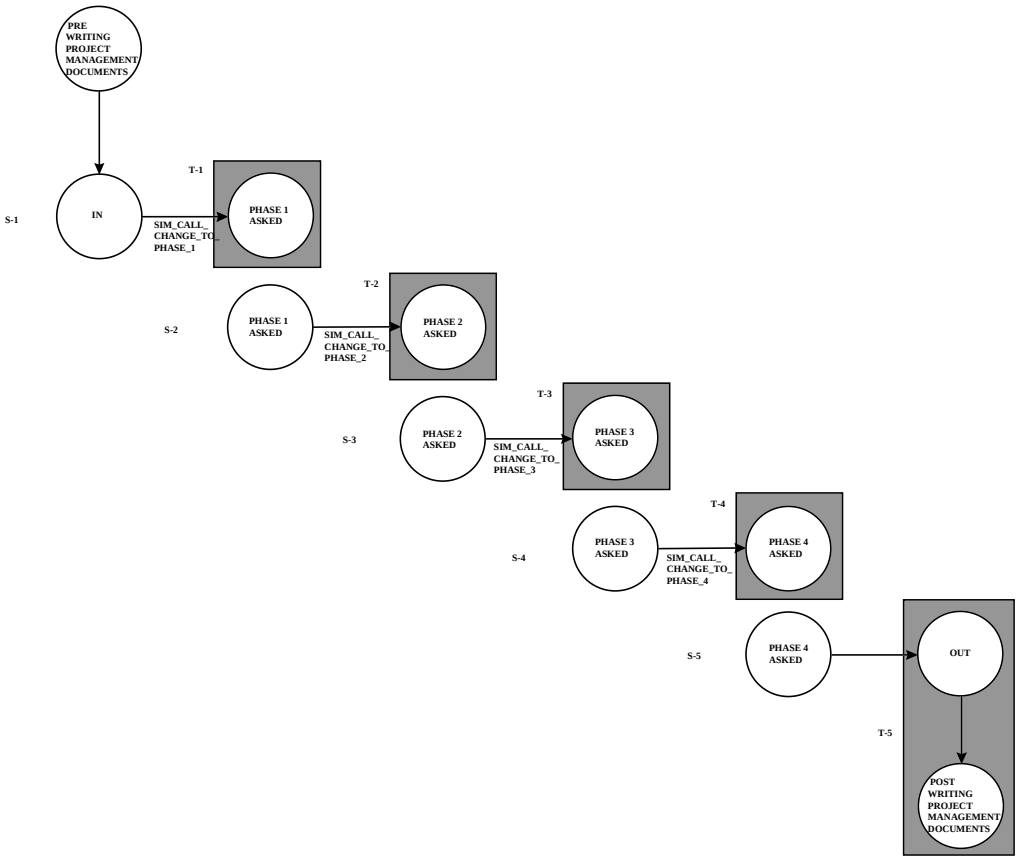


figure 6.196 employee int-pr_project_life_cycle : subprocesses S1, S2, S3, S4, S5 wrt_terms_of_reference_document

This operation ‘pr_project_life_cycle’ is not only an employee of ‘terms of reference document’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1, S2, S3, S4 and S5. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses. The actual subprocesses of this

operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

The employee ‘phase_ended’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘terms of reference document’), according to the caller-callee construct.

The employee ‘pr_phase_ended’ is the caller of the callee ‘phase_ended’ of the class ‘terms of reference document’. It has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘terms of reference document’) according to the caller_callee-construct.

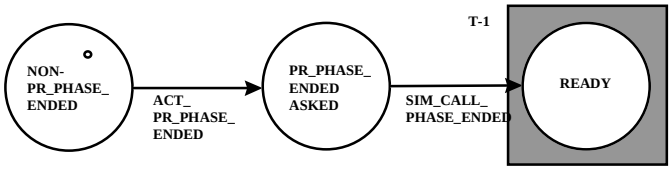


figure 6.197 employee int-pr_phase_ended : subprocess S1_wrt_terms_of_reference_document

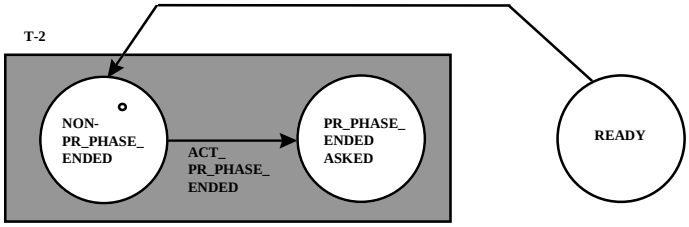


figure 6.198 employee int-pr_phase_ended : subprocess S2_wrt_terms_of_reference_document

This operation ‘pr_phase_ended’ is not only an employee of ‘terms of reference document’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1 and S2. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses.

The actual subprocesses of this operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

6.3.2.21 Internal memorandum (integration)

6.3.2.21.1 Internal memorandum (integration) : extended phase-external behavior-STDs

The first step in the construction of the total external STD for the process fragment ‘writing project management documents’ is the extension of the constituent phase-external STDs with a start state and an end (final) state. The start state has a transition leaving it and entering the phase-external STD. The final state has a transition coming into it from the phase-external STD. This construction is performed in the next figures for the phase 1-external STD, the phase 2-external STD, the phase 3-external STD and the phase 4-external STD of the class ‘internal memorandum’.

The class ‘internal memorandum’ does not participate in phase 1, 2 or 3. Therefore the phase 1-, phase 2- and phase 3-external STDs consist of a dummy state.

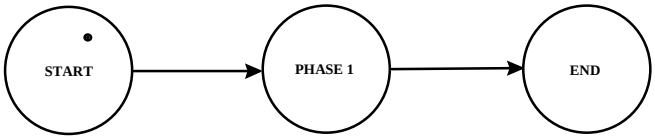


figure 6.199 internal memorandum : extended phase 1-external STD

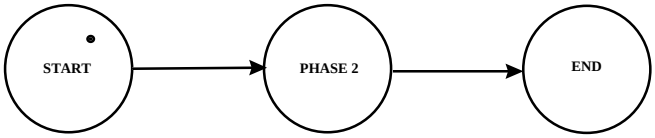


figure 6.200 internal memorandum : extended phase 2-external STD

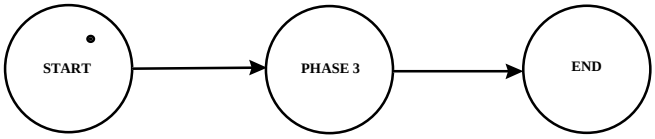


figure 6.201 internal memorandum : extended phase 3-external STD

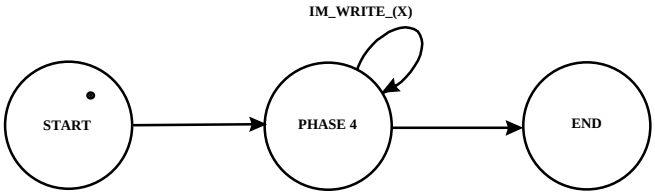


figure 6.202 internal memorandum : extended phase 4-external STD

6.3.2.21.2 Internal memorandum (integration) : total external behavior-STD

The second step in the construction of the total external STD for the process fragment ‘writing project management documents’ is to connect the extended phase-external STDs with each other in such a way that the final state of one phase-external STD coincides with the start state of the next phase-external STD. This ‘coinciding’ state between two phase-external STDs is called an ‘intermediate’ state. This models the sequential dependency between the phases.

The transitions leaving the start states get the label ‘change_to_phase_x’. These represent the ‘phase-changing’-operations. The transitions entering the final states get the label ‘pr_phase_ended’. These represent the ‘phase-ended’ operation. The construction is performed in the next figure.

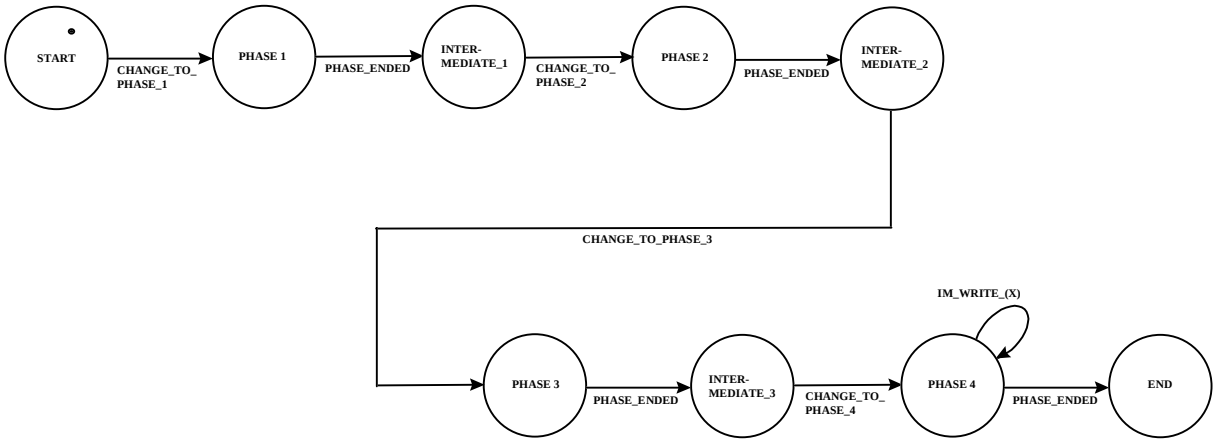


figure 6.203 internal memorandum : ext_wpmd_internal_memorandum, total external STD

The flow through this total external STD is analogous to the flow through the total external STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : total external behavior-STD).

With the difference that the class ‘internal memorandum’ does not participate in phase 1, 2 or 3. Therefore no action of ‘internal memorandum’ will take place in these phases.

6.3.2.21.3 Internal memorandum (integration) : internal behavior-STDs

The internal operations of the class ‘internal memorandum’ are in the first place the internal operation as modeled in the phase 4. During the integration the following internal operations are added to the class: ‘change_to_phase_1’, ‘change_to_ phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and ‘phase_ended’.

These 5 additional operations are so-called ‘no-operations’ (nops). A nop is an operation that performs no function while executing. The sole purpose of such an operation is to enable the corresponding transition of the external STD within the existing SOCCA framework.

These internal operations have internal STDs equivalent to the corresponding operations of the class ‘customer’ (described in the paragraph ‘Customer (integration) : internal behavior STDs’).

6.3.2.21.4 Internal memorandum (integration) : manager-STD

The total manager STD is the same as the total external STD. Therefore no separate figure is needed for the total manager STD. The total manager STD of the process fragment ‘writing project management documents’ prescribes in its ‘phase 4’ state still the same subprocesses for its phase-employees as was modeled in the phase 4-sub-model. These may be called ‘old prescribed subprocesses’. Subprocess prescriptions for an employee in states (of the total manager STD) that are not relevant for that employee are ‘don’t cares’.

In the integration the total manager gets new employees. For this new employees it is also prescribing subprocesses. These may be called ‘new prescribed subprocesses’. The sum total of the prescribed subprocesses by the total manager STD is therefore (per state) : {old prescribed subprocesses} $\cup$ {don’t cares} $\cup$ {new prescribed subprocesses}.

The new CPSs (Consolidated Prescribed Subprocesses), CCs (Caller_Callee combinations) and TLFs (Transition Logical Formulas) for this total manager STD are the same as the CPSs, CCs and TLFs for the total manager STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : manager-STD).

With the difference that all the subprocesses and traps are with respect to ‘internal memorandum’ instead of with respect to ‘requirements document’.

6.3.2.21.5 Internal memorandum (integration) : employee-STDs

The employees of the manager STD of the class ‘internal memorandum’ are in the first place the employees as modeled in the phase 4. During the integration the following 7 employees are added: ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and their caller ‘pr_project_life_cycle’ and ‘phase_ended’ and its caller ‘pr_phase_ended’.

The employee ‘change_to_phase_1’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘internal memorandum’), according to the caller-callee construct.

The employee ‘change_to_phase_2’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘internal memorandum’), according to the caller-callee construct.

The employee ‘change_to_phase_3’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘internal memorandum’), according to the caller-callee construct.

The employee ‘change_to_phase_4’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘internal memorandum’), according to the caller-callee construct.

The employee ‘pr_project_life_cycle’ is the caller of the four callees ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’ and ‘change_to_phase_4’ of the class ‘internal memorandum’. Consequently it has five subprocesses and traps with respect to the manager STD of the class ‘internal memorandum’ : S1, S2, S3, S4 and S5 and T-1, T-2, T-3, T-4 and T-5. The next figure shows these five subprocesses of the internal operation ‘pr_project_ life_cycle’.

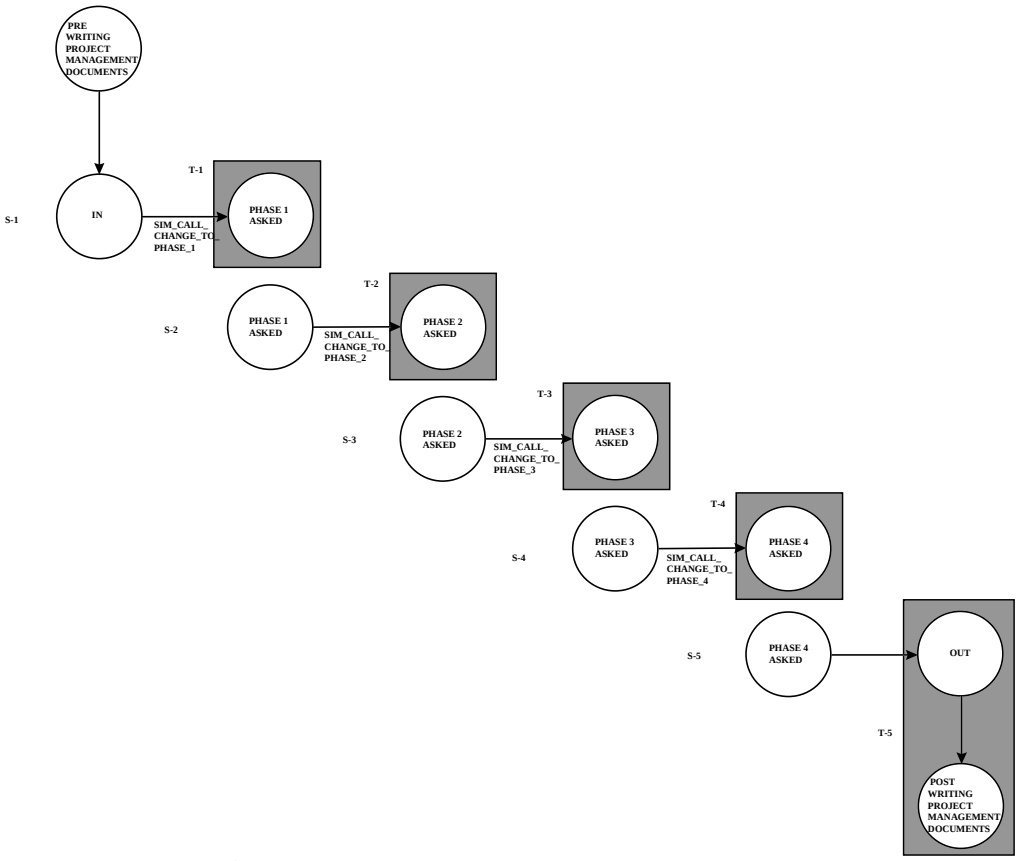


figure 6.204 employee int-pr_project_life_cycle : subprocesses S1, S2, S3, S4, S5 wrt_internal_memorandum

This operation ‘pr_project_life_cycle’ is not only an employee of ‘internal memorandum’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1, S2, S3, S4 and S5. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses. The actual subprocesses of this

operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

The employee ‘phase_ended’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘internal memorandum’), according to the caller-callee construct.

The employee ‘pr_phase_ended’ is the caller of the callee ‘phase_ended’ of the class ‘internal memorandum’. It has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘internal memorandum’) according to the caller_callee-construct.

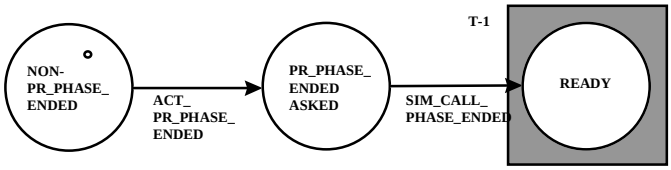


figure 6.205 employee int-pr_phase_ended : subprocess S1_wrt_internal_memorandum

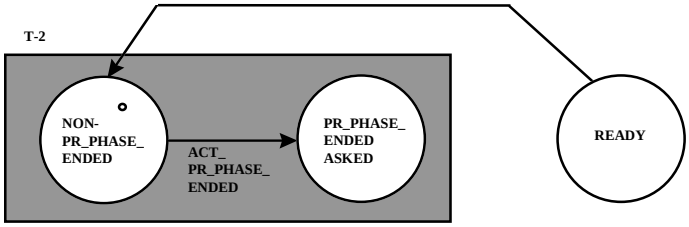


figure 6.206 employee int-pr_phase_ended : subprocess S2_wrt_internal_memorandum

This operation ‘pr_phase_ended’ is not only an employee of ‘internal memorandum’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1 and S2. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses.

The actual subprocesses of this operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

6.3.2.22 Engineer (integration)

6.3.2.22.1 Engineer (integration) : extended phase-external behavior-STDs

The first step in the construction of the total external STD for the process fragment ‘writing project management documents’ is the extension of the constituent phase-external STDs with a start state and an end (final) state. The start state has a transition leaving it and entering the phase-external STD. The final state has a transition coming into it from the phase-external STD. This construction is performed in the next figures for the phase 1-external STD, the phase 2-external STD, the phase 3-external STD and the phase 4-external STD of the class ‘engineer’.

The class ‘engineer’ does not participate in phase 1, 2 or 3. Therefore the phase 1-, phase 2- and phase 3-external STDs consist of a dummy state.

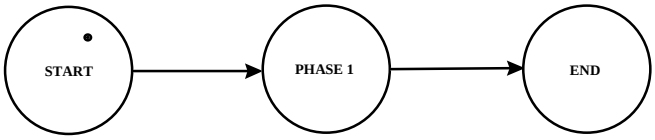


figure 6.207 engineer : extended phase 1-external STD

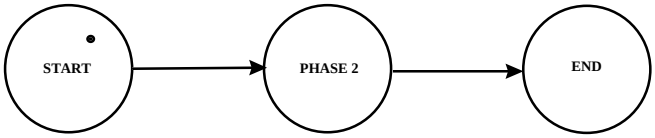


figure 6.208 engineer : extended phase 2-external STD

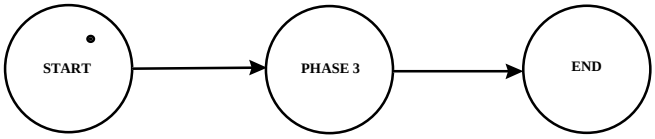


figure 6.209 engineer : extended phase 3-external STD

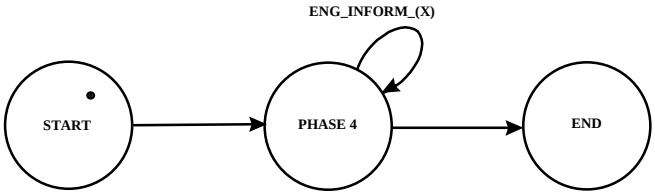


figure 6.210 engineer : extended phase 4-external STD

6.3.2.22.2 Engineer (integration) : total external behavior-STD

The second step in the construction of the total external STD for the process fragment ‘writing project management documents’ is to connect the extended phase-external STDs with each other in such a way that the final state of one phase-external STD coincides with the start state of the next phase-external STD. This ‘coinciding’ state between two phase-external STDs is called an ‘intermediate’ state. This models the sequential dependency between the phases.

The transitions leaving the start states get the label ‘change_to_phase_x’. These represent the ‘phase-changing’-operations. The transitions entering the final states get the label ‘pr_phase_ended’. These represent the ‘phase-ended’ operation. The construction is performed in the next figure.

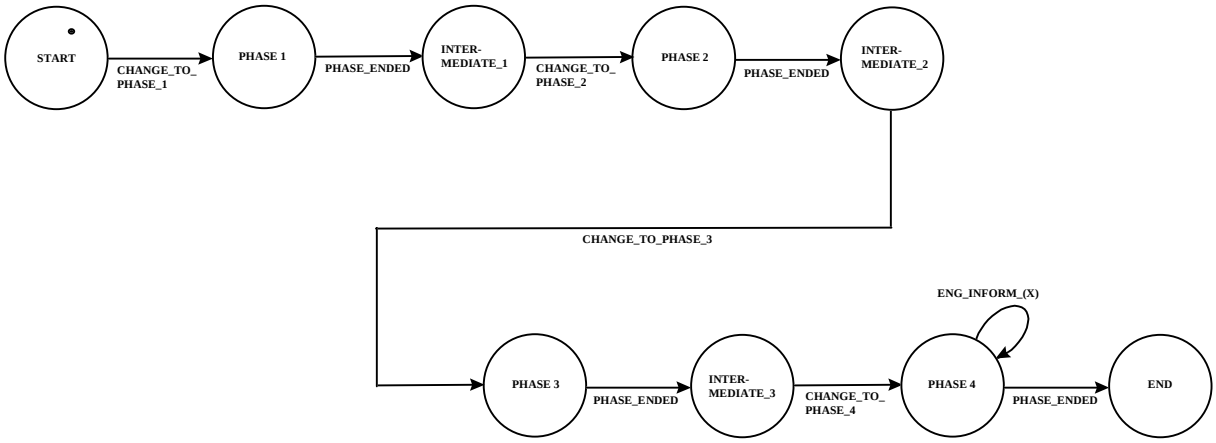


figure 6.211 engineer : ext_wpmd_engineer, total external STD

The flow through this total external STD is analogous to the flow through the total external STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : total external behavior-STD).

With the difference that the class ‘engineer’ does not participate in phase 1, 2 or 3. Therefore no action of ‘engineer’ will take place in these phases.

6.3.2.22.3 Engineer (integration) : internal behavior-STDs

The internal operations of the class ‘engineer’ are in the first place the internal operation as modeled in the phase 4. During the integration the following internal operations are added to the class: ‘change_to_phase_1’, ‘change_to_ phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and ‘phase_ended’.

These 5 additional operations are so-called ‘no-operations’ (nops). A nop is an operation that performs no function while executing. The sole purpose of such an operation is to enable the corresponding transition of the external STD within the existing SOCCA framework.

These internal operations have internal STDs equivalent to the corresponding operations of the class ‘customer’ (described in the paragraph ‘Customer (integration) : internal behavior STDs’).

6.3.2.22.4 Engineer (integration) : manager-STD

The total manager STD is the same as the total external STD. Therefore no separate figure is needed for the total manager STD. The total manager STD of the process fragment ‘writing project management documents’ prescribes in its ‘phase 4’ state still the same subprocesses for its phase-employees as was modeled in the phase 4-sub-model. These may be called ‘old prescribed subprocesses’. Subprocess prescriptions for an employee in states (of the total manager STD) that are not relevant for that employee are ‘don’t cares’.

In the integration the total manager gets new employees. For this new employees it is also prescribing subprocesses. These may be called ‘new prescribed subprocesses’. The sum total of the prescribed subprocesses by the total manager STD is therefore (per state) : {old prescribed subprocesses} $\cup$ {don’t cares} $\cup$ {new prescribed subprocesses}.

The new CPSs (Consolidated Prescribed Subprocesses), CCs (Caller_Callee combinations) and TLFs (Transition Logical Formulas) for this total manager STD are the same as the CPSs, CCs and TLFs for the total manager STD of the class ‘requirements document’ (as described in the paragraph ‘Requirements document (integration) : manager-STD).

With the difference that all the subprocesses and traps are with respect to ‘engineer’ instead of with respect to ‘requirements document’.

6.3.2.22.5 Engineer (integration) : employee-STDs

The employees of the manager STD of the class ‘engineer’ are in the first place the employees as modeled in the phase 4. During the integration the following 7 employees are added: ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’, ‘change_to_phase_4’ and their caller ‘pr_project_life_cycle’ and ‘phase_ended’ and its caller ‘pr_phase_ended’.

The employee ‘change_to_phase_1’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘engineer’), according to the caller-callee construct.

The employee ‘change_to_phase_2’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘engineer’), according to the caller-callee construct.

The employee ‘change_to_phase_3’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘engineer’), according to the caller-callee construct.

The employee ‘change_to_phase_4’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘engineer’), according to the caller-callee construct.

The employee ‘pr_project_life_cycle’ is the caller of the four callees ‘change_to_phase_1’, ‘change_to_phase_2’, ‘change_to_phase_3’ and ‘change_to_phase_4’ of the class ‘engineer’. Consequently it has five subprocesses and traps with respect to the manager STD of the class ‘engineer’ : S1, S2, S3, S4 and S5 and T-1, T-2, T-3, T-4 and T-5. The next figure shows these five subprocesses of the internal operation ‘pr_project_ life_cycle’.

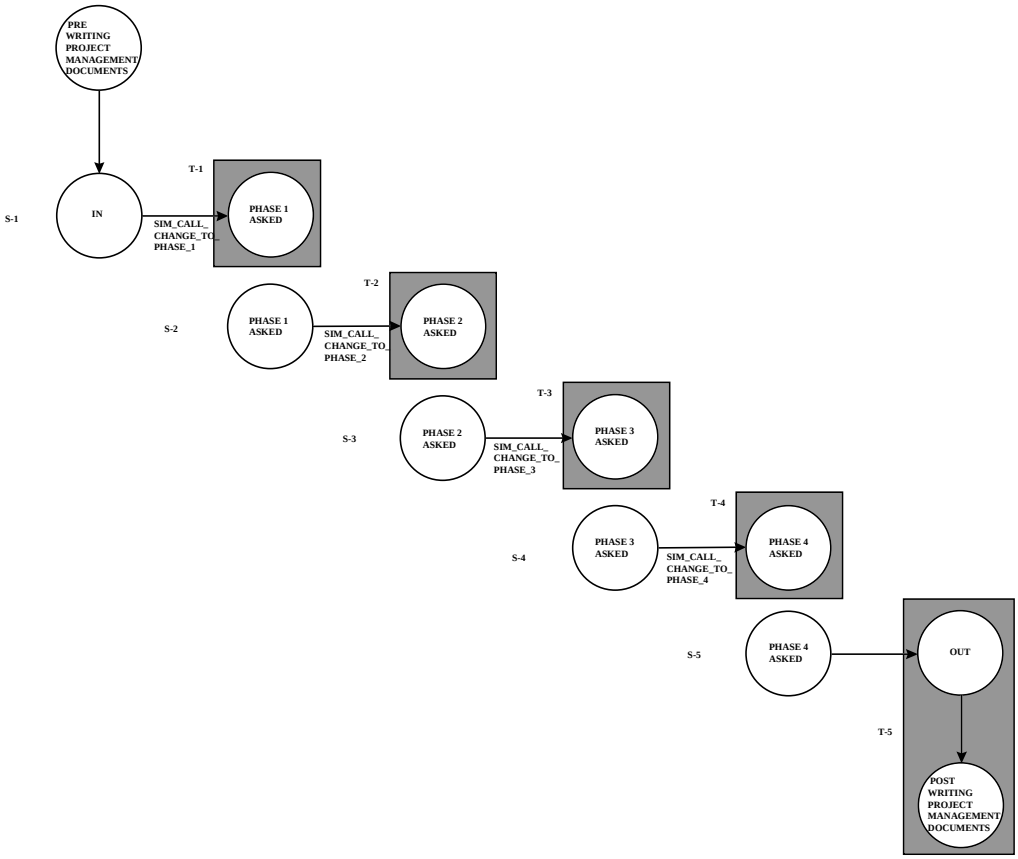


figure 6.212 employee int-pr_project_life_cycle : subprocesses S1, S2, S3, S4, S5 wrt_engineer

This operation ‘pr_project_life_cycle’ is not only an employee of ‘engineer’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1, S2, S3, S4 and S5. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses. The actual subprocesses of this operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

The employee ‘phase_ended’ has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘engineer’), according to the caller-callee construct.

The employee ‘pr_phase_ended’ is the caller of the callee ‘phase_ended’ of the class ‘engineer’. It has two subprocesses S1 and S2, and two traps T-1 and T-2 (all with respect to ‘engineer’) according to the caller_callee-construct.

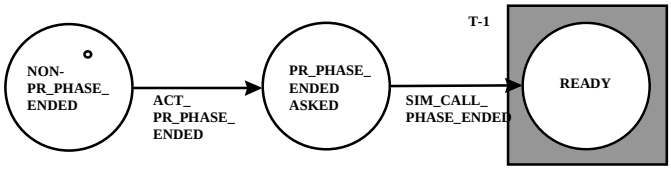


figure 6.213 employee int-pr_phase_ended : subprocess S1_wrt_engineer

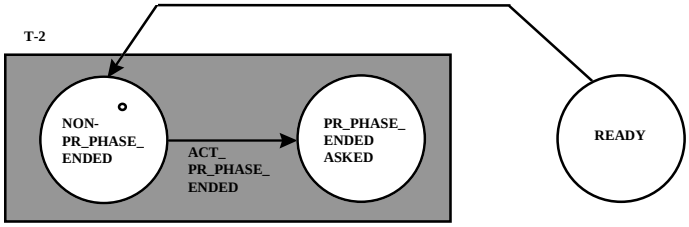


figure 6.214 employee int-pr_phase_ended : subprocess S2_wrt_engineer

This operation ‘pr_phase_ended’ is not only an employee of ‘engineer’, but also of all other participating classes of the process fragment ‘writing project management documents’. All these participating classes prescribe the same subprocesses S1 and S2. The operation has therefore 19 subprocesses prescribed to it at any one time. The actual subprocess is the intersection of these 19 prescribed subprocesses.

The actual subprocesses of this operation due to the intersection mechanism, are explained in the paragraph ‘Requirements document (integration) : employee-STDs’.

SOFTWARE PROCESS MODELING in SOCCA

student : H.G.Brugman
studentnr : 9105506
version : 0.10

7. Summary and Conclusions

This thesis presents the investigation of two already existing ideas within the Software Engineering and Information Systems (SEIS) group of the Department of Computer Science of the University of Leiden, to achieve very large SOCCA models.

This thesis also looks at the issue of the use of a SOCCA model as a process description instead of the usual textual description.

The three topics addressed in this thesis are :

- investigate the use of often recurring SOCCA 'constructs' to limit the size of the model
- investigate if the SOCCA modeling language can be scaled up (i.e. can a larger SOCCA model be constructed from indepently developed sub-models)
- investigate the usefullness of a SOCCA model as a process description

Use of 'constructs' to limit the size of the SOCCA model

The first question was handled as follows. A very large SOCCA model was developed. The size of this model is 29 classes, 86 operations and 1041 state transition diagrams (of wich only 310 state transition diagrams had to be explicitily shown as a result of using 'constructs').

The modeled process is the software process of the software development organization 'WBU' of the Dutch Ministry of Defense. Two processes of this organization were modeled using the SOCCA modeling language. These processes are the 'Software Configuration Management'-process and part of the 'Software Project Planning'-process (i.e. the process fragment 'writing project management documents'). The basis for the models were the process descriptions as found in the 'Manual for Technical Project Management' of the 'WBU' organization.

It was found that these processes, SCM and part of SPP, could be accurately modeled in SOCCA. The participants (agents) in a process, be they humans, documents, software items or abstract concepts (e.g. a meeting), can all be modeled as objects. Their actions while performing the process, are modeled as the operations of these objects. The communication between the participants is modeled by using the concepts of a 'parallel processes modeling' formalism which is part of SOCCA.

This combination within SOCCA of object-oriented concepts and 'parallel processes' concepts allows for the development of an accurate model of the real world process. In the real world a lot of parallel actions take place. The ability of SOCCA to model this parallellism is something which gives it an 'added value' with respect to other object oriented modeling languages. The object orientation of SOCCA is in line with the current direction of object oriented analysis and design in the software engineering field.

During the modeling often recurring 'constructs' were identified, named and used. These 'constructs' are :

- Caller_Callee-construct
- Waiting_caller_proceed-construct
- Only_internal_action-template
- No-operation (nop)
- Autonomous behavior
- Simultaneous_call-construct
- Discriminator-construct
- Counting-construct
- Consolidated Prescribed Subprocesses and Traps Logical Formula

The size of the developed model is 29 classes, 86 operations and, without the use of 'constructs', 1041 state transition diagrams (STDs). As a result of using 'constructs' only 310 state transition diagrams (STDs) had to be explicitily shown in the model.

This is a reduction in the size of the SOCCA model of 731 state transition diagrams, i.e. a reduction of 70 %. The 'Caller_Callee-construct' with its variant 'Waiting_caller_proceed-construct', was by far the most important in terms of limiting the size of the SOCCA model. This 'construct' specifically applies to the so-called 'subprocess'-STDs of a SOCCA model. In the original SOCCA model 688 of the 1041 STDs are of this type. When using the 'construct' only 118 'subprocess'-STDs had to be shown. This is a saving of 570 STDs, or 55 %.

The 'only_internal_action template' with its variant 'No-operation', also made a significant contribution to the reduction of the size of the SOCCA model. This 'construct' specifically applies to the so-called 'internal'-STDs of a SOCCA model. In the original SOCCA model 184 of the 1041 STDs are of this type. When using the 'construct' only 89 'internal'-STDs had to be shown. This is a saving of 95 STDs, or 9 %.

The 'External STD = Manager STD' variant of the 'Consolidated Prescribed Subprocesses and Traps Logical Formula' also made a contribution. It accounted for a saving of 66 STDs, or 6 %.

A more detailed analysis of the saving in the different parts of the SOCCA model is given in the tables below.

- Software Configuration Management, 11 classes (without constructs)

class	extern STD organ. view	extern STD com. view	manager STD	internal STD	employee subpr. STD	Total
tpm	-	1	1	6	12	20
cm	-	1	1	6	48	56
ci	1	1	1	13	46	62
pcr	-	1	1	8	30	40
se	-	1	1	1	4	7
re	-	1	1	1	4	7
scb	-	1	1	1	4	7
ccb	-	1	1	1	4	7
te	-	1	1	1	4	7
cu	-	1	1	2	6	10
rn	-	1	1	1	4	7
	------	------	------	------	------	------
Total	1	11	11	41	166	230

- Software Configuration Management, 11 classes, Saving 80 %
(with caller_callee construct, only_internal_action template and manager STD = external STD)

class	extern STD organ. view	extern STD com. view	manager STD	internal STD	employee subpr. STD	Total
tpm	-	1	-	6	-	7
cm	-	1	-	6	-	7
ci	1	1	-	1	6	9
pcr	-	1	-	1	6	8
se	-	1	-	1	-	2
re	-	1	-	1	-	2
scb	-	1	-	1	-	2
ccb	-	1	-	1	-	2
te	-	1	-	1	-	2
cu	-	1	-	2	-	3
rn	-	1	-	1	-	2
	------	------	------	------	------	------
Total	1	11	-	22	12	46

- Software Project Planning (phase 1), 9 classes (without constructs)

class	extern STD organ. view	extern STD com. view	manager STD	internal STD	employee subpr. STD	Total
cu	1	1	1	1	2	6
rd	1	1	1	1	4	8
am	1	1	1	1	4	8
mb	1	1	1	1	4	8
ceo	1	1	1	1	4	8
hps	1	1	1	2	8	13
pf	1	1	1	1	4	8
hcs	1	1	1	1	4	8
tpm	1	1	1	1	4	8
	------	------	------	------	------	------
Total	9	9	9	10	38	75

- Software Project Planning (phase 1), 9 classes, Saving 62 %
(with caller_callee construct and manager STD = external STD)

class	extern STD organ. view	extern STD com. view	manager STD	internal STD	employee subpr. STD	Total
cu	1	1	-	1	-	3
rd	1	1	-	1	-	3
am	1	1	-	1	-	3
mb	1	1	-	1	-	3
ceo	1	1	-	1	-	3
hps	1	1	-	2	-	4
pf	1	1	-	1	-	3
hcs	1	1	-	1	-	3
tpm	1	1	-	1	-	3
	------	------	------	------	------	------
Total	9	9	-	10	-	28

- Software Project Planning (phase 2), 8 classes (without constructs)

class	extern STD organ. view	extern STD com. view	manager STD	internal STD	employee subpr. STD	Total
tpm	1	1	1	4	15	22
cu	1	1	1	2	8	13
am	1	1	1	2	8	13
qaa	1	1	1	2	8	13
hprs	1	1	1	4	16	23
hss	1	1	1	1	4	8
pmd	1	1	1	1	4	8
pmm	1	1	1	1	4	8
	------	------	------	------	------	------
Total	8	8	8	17	67	108

- Software Project Planning (phase 2), 8 classes, Saving 64 %
  (with caller_callee construct and manager STD = external STD)

class	extern STD organ. view	extern STD com. view	manager STD	internal STD	employee subpr. STD	Total
tpm	1	1	-	4	5	11
cu	1	1	-	2	-	4
am	1	1	-	2	-	4
qaa	1	1	-	2	-	4
hprs	1	1	-	4	-	6
hss	1	1	-	1	-	3
pmd	1	1	-	1	-	3
pmm	1	1	-	1	-	3
	------	------	------	------	------	------
Total	8	8	-	17	5	38

- Software Project Planning (phase 3), 9 classes (without constructs)

class	extern STD organ. view	extern STD com. view	manager STD	internal STD	employee subpr. STD	Total
pmm	1	1	1	1	4	8
ceo	1	1	1	1	4	8
hss	1	1	1	1	4	8
qaa	1	1	1	1	4	8
hcs	1	1	1	1	4	8
am	1	1	1	1	4	8
cu	1	1	1	1	4	8
ada	1	1	1	1	4	8
tpm	1	1	1	1	6	10
	------	------	------	------	------	------
Total	9	9	9	9	38	74

- Software Project Planning (phase 3), 9 classes, Saving 55 %
  (with caller_callee construct and manager STD = external STD)

class	extern STD organ. view	extern STD com. view	manager STD	internal STD	employee subpr. STD	Total
pmm	1	1	-	1	-	3
ceo	1	1	-	1	2	5
hss	1	1	-	1	2	5
qaa	1	1	-	1	2	5
hcs	1	1	-	1	-	3
am	1	1	-	1	-	3
cu	1	1	-	1	-	3
ada	1	1	-	1	-	3
tpm	1	1	-	1	-	3
	------	------	------	------	------	------
Total	9	9	-	9	6	33

- Software Project Planning (phase 4), 9 classes (without constructs)

class	extern STD organ. view	extern STD com. view	manager STD	internal STD	employee subpr. STD	Total
hps	1	1	1	1	2	6
hcss	1	1	1	1	2	6
tor	1	1	1	1	4	8
pf	1	1	1	1	4	8
im	1	1	1	1	6	10
tpm	1	1	1	1	6	10
hps	1	1	1	1	6	10
eng	1	1	1	1	4	8
hcs	1	1	1	1	4	8
	------	------	------	------	------	------
Total	9	9	9	9	38	74

- Software Project Planning (phase 4), 9 classes, Saving 63 %
  (with caller_callee construct and manager STD = external STD)

class	extern STD organ. view	extern STD com. view	manager STD	internal STD	employee subpr. STD	Total
hps	1	1	-	1	-	3
hcss	1	1	-	1	-	3
tor	1	1	-	1	-	3
pf	1	1	-	1	-	3
im	1	1	-	1	-	3
tpm	1	1	-	1	-	3
hps	1	1	-	1	-	3
eng	1	1	-	1	-	3
hcs	1	1	-	1	-	3
	------	------	------	------	------	------
Total	9	9	-	9		27

- Integration, 20 classes (without constructs)

class	extern STD organ. view	extern STD com. view	manager STD	internal STD	employee subpr. STD	Total
project	1	1	1	3	18	24
cu	-	1	1	5	17	24
rd	-	1	1	5	17	24
am	-	1	1	5	17	24
mb	-	1	1	5	17	24
ceo	-	1	1	5	17	24
hps	-	1	1	5	17	24
pf	-	1	1	5	17	24
hcs	-	1	1	5	17	24
tpm	-	1	1	5	17	24
qaa	-	1	1	5	17	24
hprs	-	1	1	5	17	24
hss	-	1	1	5	17	24
pmd	-	1	1	5	17	24
pmm	-	1	1	5	17	24

ada	-	1	1	5	17	24
hccs	-	1	1	5	17	24
tor	-	1	1	5	17	24
im	-	1	1	5	17	24
eng	-	1	1	5	17	24
	------	------	------	------	------	------
Total	1	20	20	98	341	480

- Integration, 20 classes, Saving 71 %
(with caller_callee construct, nop, manager STD = external STD and finishing_state construct)

class	extern STD organ. view	extern STD com. view	manager STD	internal STD	employee subpr. STD	Total
project	1	1	-	3	-	5
cu	-	1	-	1	5	7
rd	-	1	-	1	5	7
am	-	1	-	1	5	7
mb	-	1	-	1	5	7
ceo	-	1	-	1	5	7
hps	-	1	-	1	5	7
pf	-	1	-	1	5	7
hcs	-	1	-	1	5	7
tpm	-	1	-	1	5	7
qaa	-	1	-	1	5	7
hprs	-	1	-	1	5	7
hss	-	1	-	1	5	7
pmd	-	1	-	1	5	7
pmm	-	1	-	1	5	7
ada	-	1	-	1	5	7
hccs	-	1	-	1	5	7
tor	-	1	-	1	5	7
im	-	1	-	1	5	7
eng	-	1	-	1	5	7
	------	------	------	------	------	------
Total	1	20	-	22	95	138

- Grand Total

# classes	# STDs (not using constructs)	# STDs (using constructs)	Saving (# STDs)	Saving (%)
29	1041	310	731	70 %

Conclusion

The conclusion that can be drawn is that use of ‘constructs’ do indeed limit the size of a SOCCA model significantly. In this thesis a reduction of size of 70 % was achieved. The ‘Caller_Callee-construct’ alone contributed a saving of 55 %. It is expected that this ‘Caller_Callee-construct’ will become an imported ‘tool’ when making a very large SOCCA model.

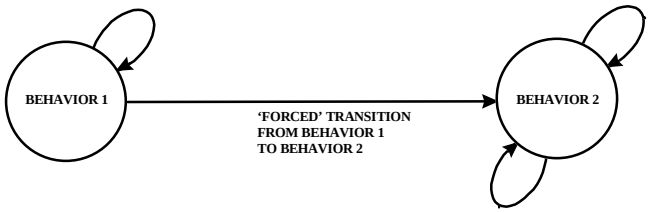
It can be noted that the ‘simultaneous_call-construct’, ‘discriminator-construct’ and ‘counting-construct’ have no influence on the number of STDs in the model. They ‘standardize’ often recurring situations in a process. Their use speeds up the modeling. They also reduce the number of states in a STD.

Scaleability of SOCCA

To look into the question of the 'scaleability' of SOCCA, the modeling of the process fragment 'writing project management documents' was done by splitting the process fragment into four smaller process fragments. These smaller process fragments were modeled indepently of each other. Then the sub-models of these four smaller process fragments were integrated into one SOCCA model of the total process fragment 'writing project management documents'.

The intention of the integration process was not to influence the four sub-models. That is to say to make the modeling of the constituent smaller process fragments independent of their integration. This allows for a separate modeling of the sub-models (by separate engineers) in a big project. It also facilitates the update of the total model if there are any changes required during the life time of the model. This is according to the accepted software engineering principle of low coupling between software modules.

During the modeling of the process fragment 'writing project management documents' an 'integration'-algorithm was developed. The basic idea of this algorithm is that of 'foreseen evolution' of a process. The behavior of the process evolves over time. This is accomplished by 'forcing' the external STDs of the classes participating in the process from one behavior (state) to the next.



This 'forcing' is done by a special object, the 'control' object. This principle of 'foreseen' process evolution is used in the integration of SOCCA sub-models. What is done, is constructing a 'total' external STD per class. The sub-model external STDs of each class are linked together to form a 'total' external STD per class. The 'total' external STD is the external STD for that class for the total integrated SOCCA model. The 'control' operation 'forces' all total external STDs in the integrated model from one sub-model behavior to the next. This is in fact the sequential integration of sub-models. With some adaptations parallel integration of sub-models is also possible.

So, three kinds of integration of SOCCA sub-models are possible :

- sequential integration of sub-models
- parallel integration of sub-models
- mixed sequential and parallel integration of sub-models

This constitutes the first step in the 'scaling up' of a SOCCA model. The next step involves the integration of sub-models that are themselves integrated models. I.e. the sub-models to be integrated are the result of the first step. They already incorporate a 'control' class. Now the integration algorithm is applied to the 'control' classes of the sub-models. The 'control' classes are then managed by a 'master' control class. (Another, less modular way to scale up, is to construct a new 'control' class that applies to all the integrated models. This new 'control' class then supersedes the existing 'control' classes of the participating integrated models. Also for all the classes in the integrated models their 'grand total' external STDs have to be constructed, using the 'total' external STDs of the integrated models.)

So, also three kinds of integration of SOCCA integrated sub-models are possible :

- sequential integration of integrated sub-models
- parallel integration of integrated sub-models
- mixed sequential and parallel integration of integrated sub-models

In this way successively bigger models can be built. To prevent the bigger models to become to unwieldy (physically to big), the external STDs must be aggregated to a higher level view before they are incorporated into their total external STD. In this way, using the integration algorithm in combination with the view concept, the model can be scaled up in a transparant manner.

Constructing a view on the external STDs before using them in a ‘total’ external STD also diminishes the amount of ‘coupling’ between the sub-models and the integration model. Changes in a sub-model are less likely to influence the integration model, because they are not ‘visible’ in the view of the external STD of the sub-model.

A view of a STD is another STD in which some parts of the original STD are shown and other parts not (those are hidden). Only the information relevant for a specific ‘user’ is shown. This thesis documents two methods for the construction of a view of a STD. These are the ‘homomorphic picture’-construction and the ‘aggregate state’-construction.

Another measure to lower the ‘coupling’ between sub-models and integration model and reduce the amount of rework needed on the sub-models when integrating, was the introduction of the ‘finishing state’-construct. This construct standardizes the communication between a sub-model and the integration model.

The integration algorithm was succesfully used in this thesis to integrate the four sub-models of the process fragment ‘writing project management documents’ of the ‘Software Project Planning’ process. The integration involved :

- sequential integration
- 4 sub-models
- 1 control class
- 19 participating classes

The extra work to perform the integration was :

- 20 external STDs
- 22 internal STDs
- 95 subprocess STDs

This is a total of 138 STDs, or 45 % of the total of 310 STD in the SOCCA model. This seems a heavy price for the possibility of integration. However, because of the experimental character of the integration as performed in this thesis, no maximum use was made of the ‘constructs’. If maximum use is made, the 22 internal STDs can be reduced to 4 internal STDs, and the 95 subprocess STDs can be reduced to 5 subprocess STDs. Then the extra work to perform the integration will be 29 STDs, or 14 % of the then total of 202 STDs in the SOCCA model. An overhead of 14 % seems a reasonable price to pay for the integration.

Conclusion

The integration algorithm allows for the construction of very large SOCCA models :

- by using a team of designers each modeling sub-model(s)
- by re-using existing SOCCA models (model-fragments)
- because of the scaleability of the integration algorithm

It also allows for :

- better control of the design process by decomposition of the process to be modeled
- higher maintainability of the resulting SOCCA model
- low coupling between sub-models and the integration model

The overhead of 14 % looks acceptable when compared to the above mentioned benefits.

It must be noted that the modeling of the integration of the process fragment ‘writing project management documents’ involved only sequential integration. No integration however was performed in this thesis involving parallel or mixed integration. Also no integration was performed using a ‘master control’ class. So it is recommended to do further research in this area to check if the integration algorithm as it is defined now, maybe needs some adaptations or extensions for parallel, mixed or ‘master control’ class integration.

The usefullness of a SOCCA model as a process description

The usefullness of a SOCCA model as a process description compared to the usual textual description was investigated. This was done by checking if the SOCCA models of the ‘Software Configuration Management’-process and the ‘Software Project Planning’-process could be used as input for a process audit. As audit method was chosen the ‘Capability Maturity Model’-assessment.

The Capability Maturity Model (CMM) is a framework that will help organizations manage and improve their software process. The CMM was developed by the Software Engineering Institute (SEI) of the Carnegie Mellon University.

CMM provides a set of generic rules for the software process. It prescribes regular measurement of the software process. Against these measurements the implementation of the rules is verified. On the basis of this feedback the software process can be better managed and improved.

The measure in which an organization controls its software process is an indication for the ‘maturity’ of that organization. A mature organization is more likely to deliver a software product according to the specifications, on time and within budget. The Capability Maturity Model distinguishes between five levels of maturity : Initial, Repeatable, Defined, Managed and Optimizing.

Each maturity level is divided into Key Process Areas (KPAs). Each Key Process Area is divided into Key Practices. These Key Practices constitute the ‘checklist’ for each Key Process Area. When all Key Practices of all the Key Process Areas of a certain maturity level have been fullfilled by an organization, the organization has reached that maturity level.

The SOCCA models in this thesis concern two of the Key Process Areas of maturity level 2. Namely the KPA ‘Software Configuration Management’ and the KPA ‘Software Project Planning’. Consequently the SOCCA models were checked against the Key Practices of these KPAs.

The result of this audit is a ‘ticked off’ CMM assessment checklist. The checklists are part of the chapters containing the SOCCA models. The entries in these lists were checked against the SOCCA process model.

Conclusion

It has been found that the process audit could readily be performed with the SOCCA models as input. In this respect a SOCCA model has the same usefullness as a textual process description. It must be noted however that a SOCCA model enforces a more accurate process description. During the modeling several inconsistencies were discovered in the textual process descriptions that were the basis of the SOCCA models. So, when a realy accurate process description is required, it is advisable to construct a SOCCA model.

FINAL CONCLUSIONS

- ‘constructs’ limit the size of a SOCCA model significantly. In this thesis a reduction of size of 70 % was achieved

- the integration algorithm developed in this thesis allows a SOCCA model to be scaled up in an easy manner

- a SOCCA model is a more accurate process description than a textual process description

The SOCCA model developed in this thesis comprises 29 classes and 310 STDs. The net modeling time is estimated at 0.5 year. It is therefore felt that constructing realy large SOCCA models will be a team effort. Some form of integration will be indispensable in this effort. The integration algorithm presented in this thesis appears to be a promising candidate for this.

SOFTWARE PROCESS MODELING in SOCCA

student : H.G.Brugman
studentnr : 9105506
version : 0.10

8. References

[CMM] Paulk M.C., Curtis B., Chrissis M.B., Weber C.V. : Capability Maturity Model for Software, Version 1.1, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, Technical Report CMU/SEI-93-TR-24 (ESC-TR-93-177), February 1993

[EBE] Ebert J., Engels G. : Structural and Behavioural Views on OMT-Classes, in Bertino E., Urban S. (eds.) : Object-Oriented Methodologies and Systems (ISOOMS), Palermo, September 1994, LNCS 858, pp. 142-157, Springer-Verlag, Berlin, Germany, 1994

[ELM] Elmasri R., Navathe S. : Fundamentals of Database Systems, second edition, pp. 457-462, pp. 530-532, Benjamin/Cummings Publishing Company Inc., Redwood City, California, 1994

[FIN] Finkelstein A., Kramer J., Nuseibeh B. (eds.) : Software Process Modelling and Technology, pp. 9-31, Research Studies Press Ltd., Taunton, England, 1994

[FOW] Fowler M., Scott K. : UML Distilled, Applying the Standard Object Modeling Language, Object Technology Series, Addison Wesley Longman Inc., Reading, Massachusetts, 1997

[GRO] Groenewegen L.P.J. : Simulation, Modelling parallel phenomena in Paradigm, Lecture Notes No. 85, University of Leiden, Department of Computer Science, Leiden, The Netherlands, 1994

[HAR] Harel D. : Statecharts: A Visual Formalism for Complex Systems, in Science of Computer Programming, Vol 8, No. 3, pp. 231-274, June 1987

[HUM] Humprey W.S. : Managing the Software Process, SEI series in Software Engineering, Addison-Wesley Publishing Company Inc., Reading, Massachusetts, 1989

[LON] Lonchamp J. : A Structured Conceptual and Terminological Framework for Software Process Engineering, in Proceedings of the 2nd International Conference on the Software Process, pages 41-53, Berlin, Germany, February 1993

[MTP] Manual for Technical Project Management, version 1.00, Waco Business Unit, Ministry of Defense, The Hague, The Netherlands (in Dutch), August 1996

[RUM] Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorensen W. : Object-Oriented Modeling and Design, Prentice-Hall International Inc., Englewood Cliffs, New Jersey, 1991

[UML] Booch G., Jacobson I., Rumbaugh J. : Unified Modeling Language, Notation Guide, version 1.0, 13 January 1997, Rational Software Corporation, Santa Clara, California, http://www.rational.com

SOFTWARE PROCESS MODELING in SOCCA

student : H.G.Brugman
studentnr : 9105506
version : 0.10

9. Abbreviations and Acronyms

ACT - Activate
ADA - Archive/Documentation Administrator
AM - Account Manager

CC - Caller(s)-Callee
CCB - Configuration Control Board
CEO - Chief Executive Officer
CI - Configuration Item
CM - Configuration Manager
CMM - Capability Maturity Model
CMS - Code Management System
CPM - Corporate Process Model
CPS - Consolidated Prescribed Subprocesses
CSCI - Computer Software Configuration Item
CU - CUstomer

DISC - Discriminator

EER - Extended Entity Relationship
ENG - Engineer
EXT - External

HCS - Head Controller Section
HCSS - Head Computer Support Section
HPS - Head Personnel Section
HPRS - Head PRoduction Section
HSS - Head Support Section

IAP - Internal Automation Projects
IM - Internal Memorandum
INT - Internal
IRA - Internal Resources Allocation

KP - Key Practice
KPA - Key Process Area

MB - Make or Buy meeting
MoD - Ministry of Defense

NOP - No OPeration

OO - Object Oriented

PC - Project Contract
PCR - Problem and Change Report
PF - Project Form
PMD - Project Management Document
PMM - Project Meeting Minus
PR - PRoject

QAA - Quality Assurance Adviser

RD - Requirements Document
RE - Reviewer
RN - Release Note

SCB - Software Configuration Board

SCM	- Software Configuration Management
SDP	- Software Development Plan
SE	- Software Engineer
SEI	- Software Engineering Institute
SEIS	- Software Engineering and Information Systems
SIM	- Simultaneous
SOCCA	- Specifications Of Coordinated and Cooperative Activities
SPP	- Software Project Planning
STD	- State Transition Diagram
TE	- Test Engineer
TLF	- Traps Logical Formula
TOR	- Terms Of Reference
TPM	- Technical Project Manager
UML	- Unified Modeling Language
Waco	- Weapon and command systems
WBU	- Weapon and command systems Business Unit