EMBEDDED SYSTEMS DESIGN METHODOLOGY AT LEIDEN EMBEDDED RESEARCH CENTER

Ed F. Deprettere   Todor Stefanov   Bart Kienhuis
Leiden Embedded Research Center (LERC)
Leiden Institute of Advanced Computer Science - Leiden University
The Netherlands
e-mail: edd@liacs.nl

Abstract: The new emerging Embedded Systems are increasingly becoming more heterogeneous, i.e., systems composed of fully programmable components (microprocessors), re-configurable components (FPGAs), and dedicated hardware blocks. To design and program these systems, we present in this paper a number of concepts including the Y-chart approach and the Abstraction Pyramid. These concepts allow designers to perform system design and design space exploration in a systematic and automated way, narrowing down the design space in a number of steps. The concepts presented in this paper provide a methodology in which embedded system platforms can be obtained that satisfy a set of constraints while establishing enough flexibility to support a given set of applications.
Key words: Embedded Systems Design, Y-chart Approach, Abstraction Pyramid, Design Space Exploration

1. INTRODUCTION

Many research groups in university and industrial laboratories are currently addressing the problem of how to cope with the increasing complexity of embedded processing, and the cost of specifying, designing, and manufacturing the processors that have to do the job. In the past, embedded processing boiled down to control operations which were implemented in micro-controllers. Micro-controllers are finite state machines (FSM) that model the behavior in terms of states and state transitions. Over the time, the complexity of these controllers steadily grew and in addition to that, the behavior changed from pure control to control plus processing: FSMs became *extended* FSMs (EFSM). FSMs and EFSMs are control dominated, and are quite different from the compute dominated *Instruction Set Architectures* (ISA), such as the MIPS, the (Strong) ARM, and DSPs. Nevertheless, EFSMs and ISAs have in common that they can not deal with concurrency: they have a strict sequential behavior. However, applications, whether control dominated or compute dominated have concurrency potentials that must be exploited when applications grow more complex and/or have hard execution time constraints. This has given birth to, among others, co-design finite state machines (CFSM) and co-processor architectures and, more recently, multiprocessor (homogeneous and heterogeneous) architectures.

Next generations embedded processing will be implemented in heterogeneous processing *systems* that are compositions of processing units, some of which are *programmable*, some *re-configurable*, and some *dedicated*, all being connected to a communication, synchronization and storage infrastructure. We call them *Embedded Systems*.

In the remaining part of the paper we present an approach to overcome the problem of the growing complexity and cost of *future embedded systems design*. The focus is on systems that can execute several applications that belong to an application domain, in particular streaming date applications: Array Signal Processing, Multimedia, Wireless Communications, and Molecular Cell Biology.

We present here the embedded systems design approach that was devised and implemented in the *Leiden Embedded Research Center* (LERC) in a number of steps, where each step appears in a *what, why,* and *how* order. The results have not been obtained in isolation: It was partly in cooperation with others among which were colleagues at Philips Research in Eindhoven, and partly simultaneous with similar developments in the control dominated domain in particular at the University of California at Berkeley [1].

2. THE Y-CHART APPROACH

The Y-chart approach [2] obeys a *separation of concerns* principle by making application specifications, architecture specification, and mapping methods manifest to permit quantification of choices. Moreover, the Y-chart approach is *abstraction-level invariant*. Thus the three entities – application models, architecture model, and mapping methods – are strictly separated whatever level of abstraction is considered: system-level, component-level, etc.

What. The Y-chart approach is visualized in Figure 1. An architecture template (a parameterized architecture, that is) is specified in terms of some Model(s) of Architecture (MoA) [3], and is accompanied by, typically, a set of (domain-specific) applications that are specified in terms of some Model(s) of Computation [4].

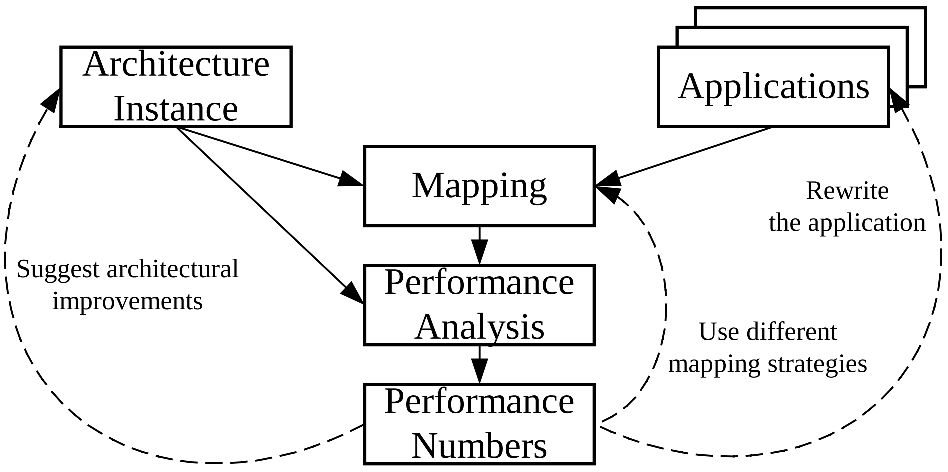

Figure 1. The Y-chart approach. Architecture, applications and mapping/performance analysis are strictly separated.

Both specifications are assumed to match (parallel architectures require applications to be specified using parallel languages, say) and are mapped onto each other (i.e., applications are mapped onto an instance of the architecture template - an architecture, that is) by means of well defined mapping methods. Performance and cost numbers are then obtained through analysis (possibly by simulation) of the applications-architecture pair. These numbers are available to the designer to revisit certain decisions through a feedback to any one of the three blocks in Figure 1.

Why. The rationale behind the Y-chart approach is the need to *master complexity*. To wit, traditional (sequential) *Instruction Set Architectures* (ISA) are not scalable and, hence, can not catch up with the exponentially growing density of transistors (Moore's law). This observation has led to a radically new approach in the design of embedded systems. Firstly, architectures are evolving from single ISAs, to ISA plus co-processor architectures, to multi-processor architectures, and beyond. These architectures are *parallel* architectures by nature. Moreover, the traditional way of designing such architectures is leading more and more to excessive non-recurrent engineering costs which can not be recovered from expected product sales. This problem can be overcome by introducing the notions of *system-level design* [5], *platform-based design* [6], and *separation of concerns* [7]. The Y-chart approach addresses all of these new trends. It advocates a separation of applications, architecture, and mapping concerns. It can also deal with platform-based design and, finally, it supports system-level design. Indeed, the Y-chart approach introduced in [2] was part of an *Abstraction Pyramid* view on emerging embedded systems design. At the top of the pyramid are the *user's requirements and constraints*. The bottom of the pyramid is the *design space*. This is visualized in Figure 2.

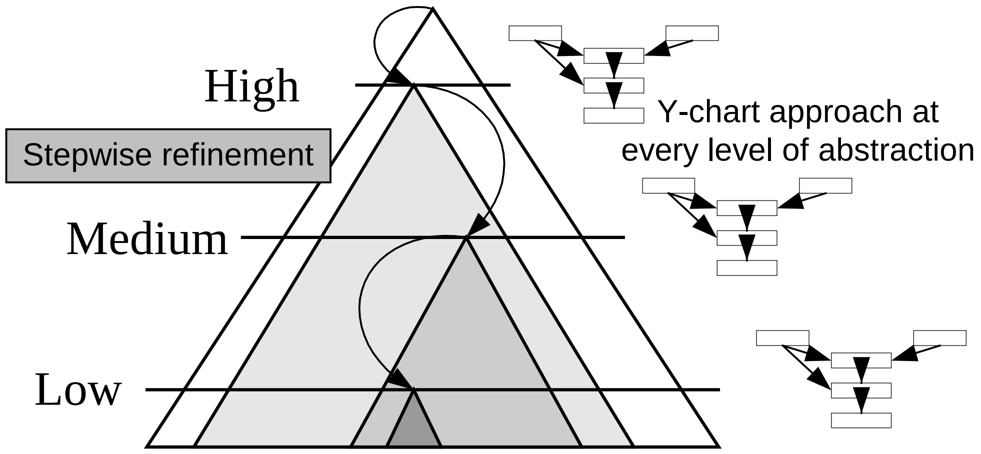

Figure 2. The Abstraction pyramid. The higher the level, the more objects there are, the faster the search and performance analysis is, but the less accurate the numbers are.

The design space can be quite large and the user's requirements can be very demanding. To master this complexity problem, the suggestion arose that one should stick to an application-domain instead of pursuing general purpose solutions. The implication of this *restriction* is that only certain applications should be implemented in only certain architectures: *domain specific applications* and *domain-specific platforms*. Roughly speaking, a platform consists of two parts: One that concerns processing elements, and one that encompasses a communication, synchronization, and storage infrastructure. This partitioning is compliant with the *communication vs. computation* separation of concerns [7]. The processing elements are taken from a library – often as intellectual property (IP) components – and the communication, synchronization, and storage infrastructure obeys certain predefined construction and operation rules. Specifying a platform is to be done through a domain analysis, and is currently still more of an art than a science. Versions of platforms are called *Architecture Templates* (parameterized architectures, that is), and instances of an architecture template are architectures. Thus the user's requirements and constraints in a particular application domain have to be brought to implementations in a platform-based design approach. The user's requirements and constraints are the highest level of abstraction and the ultimate platform–based architecture is the lowest level of abstraction.

How. How can we bridge the possibly large distance between the top and the base of the abstraction pyramid? Here is the answer: At each and every level of abstraction, a Y-chart approach architecture exploration step narrows down the design space, whereafter a *refinement* takes place to go from one level of abstraction to the next one down. This is also shown in Figure 2. Narrowing down the design space implies decision making; refinement means introducing more detailed information. For example, when it is decided that a particular processing element – a high-level parameter, that is – must be of type ISA, the Y-chart approach requires that we provide models for this ISA and for the part of the application that is to be mapped onto it as well as mapping and performance analysis methods. This is not too difficult because this is the well known case shown in Figure 3: a shared memory architecture, C language specification, and a standard compiler.

However, at higher levels of abstraction, these models and methods are not adequate. Indeed, recall that top-level domain-specific platforms define a set of *parallel architectures*, and that the Y-chart approach requires that architecture specifications and application specifications must match,

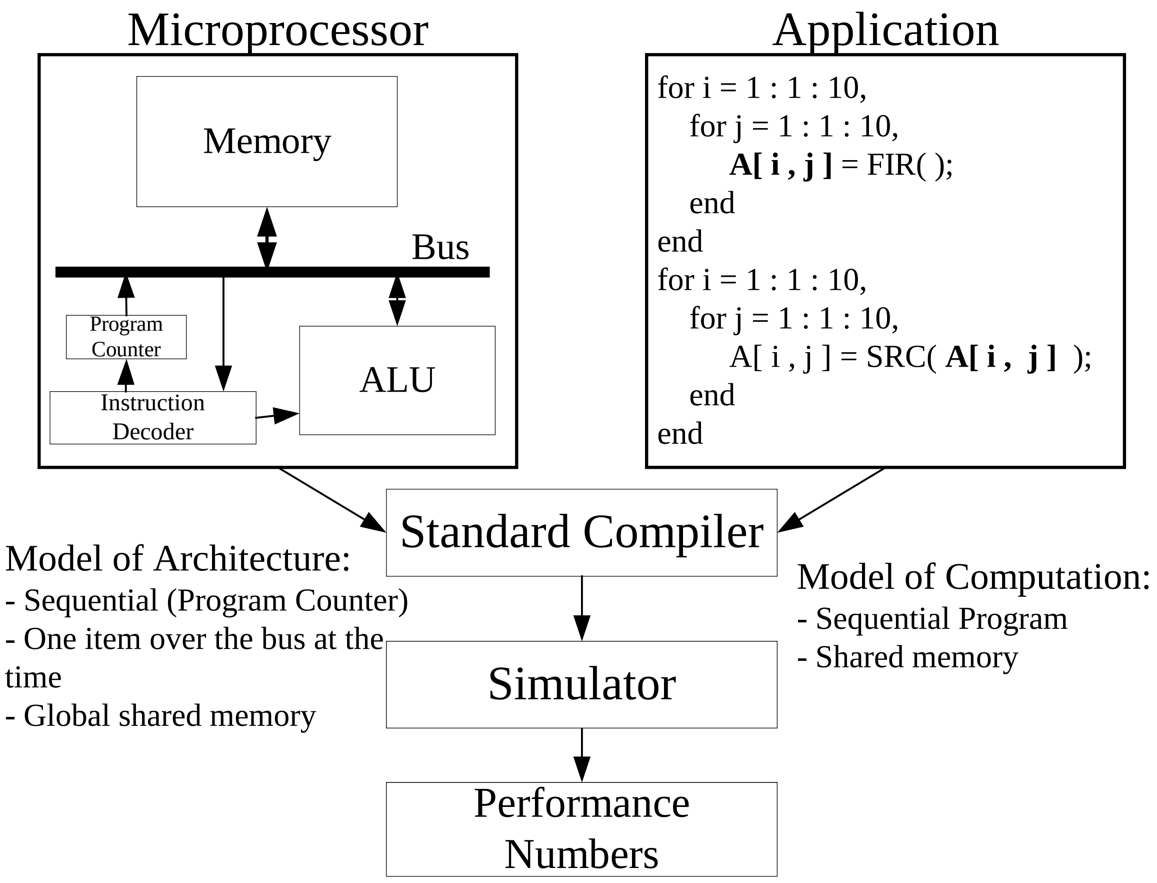

Figure 3. Traditional specification and implementation models.

implying that application specifications must be expressed in terms of *parallel programs.* There are at least three ways to provide such specifications. The first one is to use parallel languages. The second one is to rely on so-called *Models of Computation* (MoC) [4]. The third one is to translate imperative language specifications to parallel specifications. However, the compelling practice tells us that application developers invariantly use imperative languages to specify applications and that the most powerful specifications are those that are expressed in terms of MoCs. Moreover, parallel architectures can also be conveniently specified in terms of MoCs, which is an appealing way to obey the matching rule. Roughly speaking, models of computation are operational models that differ in the way communication and synchronization primitives and protocols are defined. Because all this is domain-specific, we propose to continue our discussion by taking up the particular application-domain and platform that the *Leiden Embedded Research Center* (LERC) has been working out so far. Thus we turn to the *streaming data* application domain (in particular the static, affine nested-loop programs [8])[1] and the relatively simple platform consisting of a single re-configurable processor, possibly embedded with soft programmable core.

3. THE LERC *COMPAAN* AND *LAURA* (COLA) TOOL CHAIN

What. The LERC *Compaan* and *Laura* tool sets are visualized in Figure 4. They constitute a design flow process that takes off at the top with (typically) a set of domain-specific application specifications written in an imperative language like Matlab, C/C++, or Java. Automated code-to-code transformations can be applied to these specifications by invoking the *Algorithmic*

[1] Static, affine nested-loop programs are sequential nested-loop programs in which all loop boundaries and conditions are affine functions of the loop iterators and parameters. See further down for an example.

Transformations tool. Original or code-transformed specifications are then automatically translated to functionally equivalent *Kahn Process Network* (KPN) [9] specifications[2] by invoking a set of models, methods, and tools encapsulated in the *Parallel Compiler*.

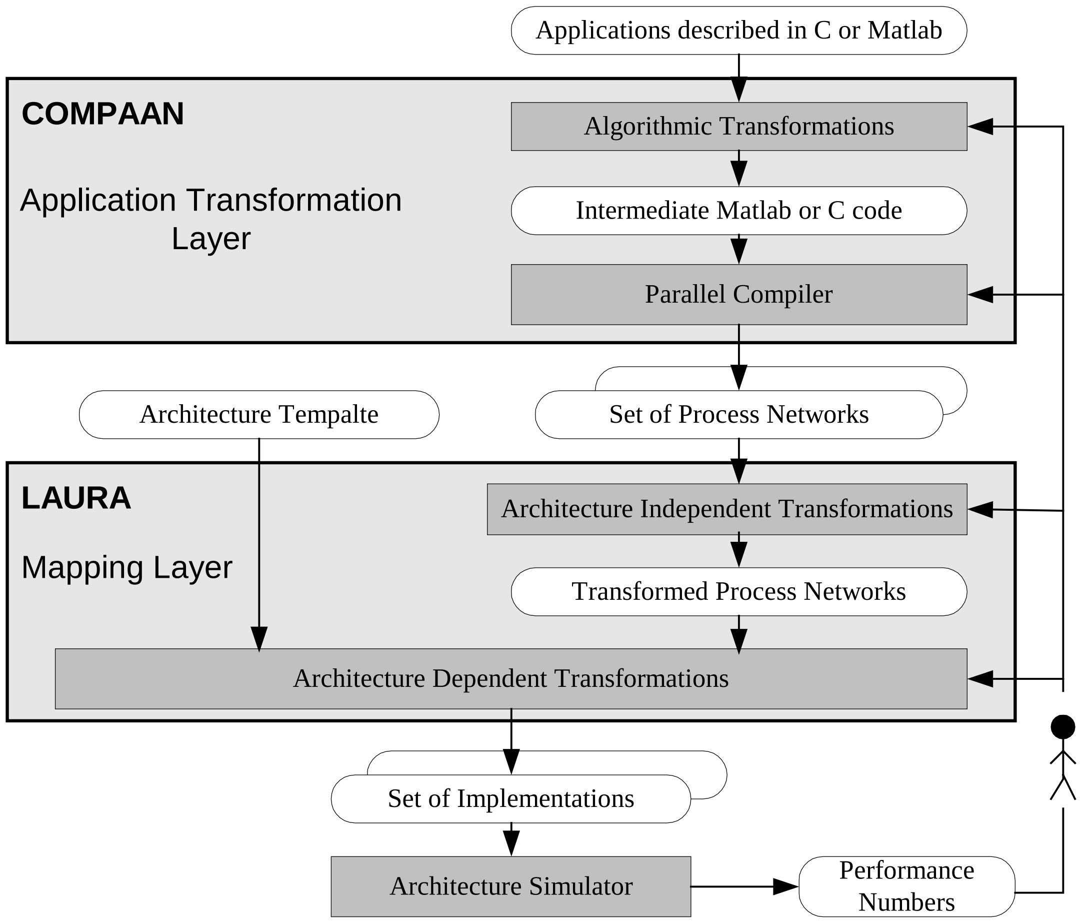
Applications described in C or Matlab
COMPAAN
Application Transformation Layer
Algorithmic Transformations
Intermediate Matlab or C code
Parallel Compiler
Architecture Tempalte
Set of Process Networks
LAURA
Mapping Layer
Architecture Independent Transformations
Transformed Process Networks
Architecture Dependent Transformations
Set of Implementations
Architecture Simulator
Performance Numbers

Figure 4. The *COMPAAN* and *LAURA* Design Flow Process.

The *Algorithmic Transformations* tool and the *Parallel Compiler* tool set constitute together the *Applications Layer* (upper gray box) which is the *Compaan* part of the flow. KPN specifications generated by *Compaan* together with a specification of a target architecture template are then taken to the *Mapping Layer* – the gray box below the *Applications Layer*. The mapping layer provides a number of models, methods, and tools to transform KPN specifications (or representations of these [3]) to architecture specifications (or representations of these [3]), partly in an *Architecture Independent Transformations* number of steps, and partly in an *Architecture Dependent Transformations* number of steps. By applying these transformations in a stepwise fashion the KPN specifications are refined into implementations. Transformation tools in the mapping layer form the *Laura* part of the flow. The architecture simulator at the bottom of Figure 4 can be used to evaluate the performance and cost of implementations generated through the mapping layer. The performance and cost numbers delivered by the simulator are available to the designer to revisit certain decisions through a feedback to the application layer and/or the mapping layer, as shown in

[2] The KPN is one of various MoCs. It has several appealing properties. It is defined as a set of autonomous processes that communicate point-to-point over unbounded FIFO channels and synchronize through simple blocking reads. There is neither a global memory nor a global scheduler. It is quite impartial to implementations. Moreover, it is deterministic, meaning that for given source data, the results are independent of the chosen operational schedule.

Figures 1 and Figure 4 by the feedback arrows to these layers. In the present *Compaan/Laura* design system, the platform is a single re-configurable processor with possibly an embedded soft core.

Why. The *LERC Compaan and Laura* tool sets have been developed to evaluate the system-level and platform-based design methodology advocated by the Y-chart approach for the streaming applications, which include Multimedia, Array Signal Processing, and Molecular Biology. The *Compaan/Laur* design tool sets constitute also one of the few system-level and platform-based approaches in the large *Matlab-to-FPGA* research community. See e.g., [10,11]. To ensure provable success, some restrictions have been built in so far. One restriction is that the feasible applications must have *static, affine nested-loop* specifications (see footnote 2). An illustrative example is the following program:

```
for k = 1 : 1 : K,
  for j = 1 : 1 : N,
     [ r( j,j) ] = F( r(j,j) , x(k,j) );
     for i = j+1 : 1 : N,
        [ r(j,i) ] = G( r(j,i) , x(k,i) );
     end
  end
end
```

The *Compaan* and *Laura* tool sets can be used to refine (parallelize) a process in a KPN application specification and to calibrate the mapping of this refinement into system architecture component. This calibration provides system level parameter values for system-level exploration. The *functions* **F()** and **G()** are not specified at this level of abstraction. They are assumed to be defined as IP cores at the level of implementation. Loop bounds and conditional constructs are affine functions of the loop iterators and/or parameters. Another restriction is that the platform is a simple re-configurable processor, possibly with embedded soft programmable core. The class of applications that satisfy the static, affine nested-loop specification restriction is still quite large. Also, the condition that the platform must be a single re-configurable processor is not severe, because such processors are predicted to become dominant in on-chip networked multi-processor architectures: Exploring the re-configurable components in such architectures is as important as exploring the complete architectures themselves, and will most likely be part of it as shown in Figure 5.

How. The *Compaan* translator takes in sequential programs written in Matlab, C/C++, or Java. An aggressive array analysis tool converts this program (or a code-to-code transformed version of it) to a so-called *single-assignment program.* A single-assignment program is a program in which the variables are assigned only once with a value. For example, the program given in Figure 6-a) is converted to the single assignment program shown in Figure 6-b).

The single-assignment program is then translated to a behavior equivalent KPN. The KPN that goes with the single-assignment program in Figure 6-b) is depicted in Figure 7. It consists of three source nodes (Nx, Ny, Nh), a computation node N_F, and a sink node Ny'. The code-to-code transformation tool can be used to transform the given program in such a way that the resulting KPN contains more nodes (more parallelism, that is) or fewer nodes (less parallelism, that is).

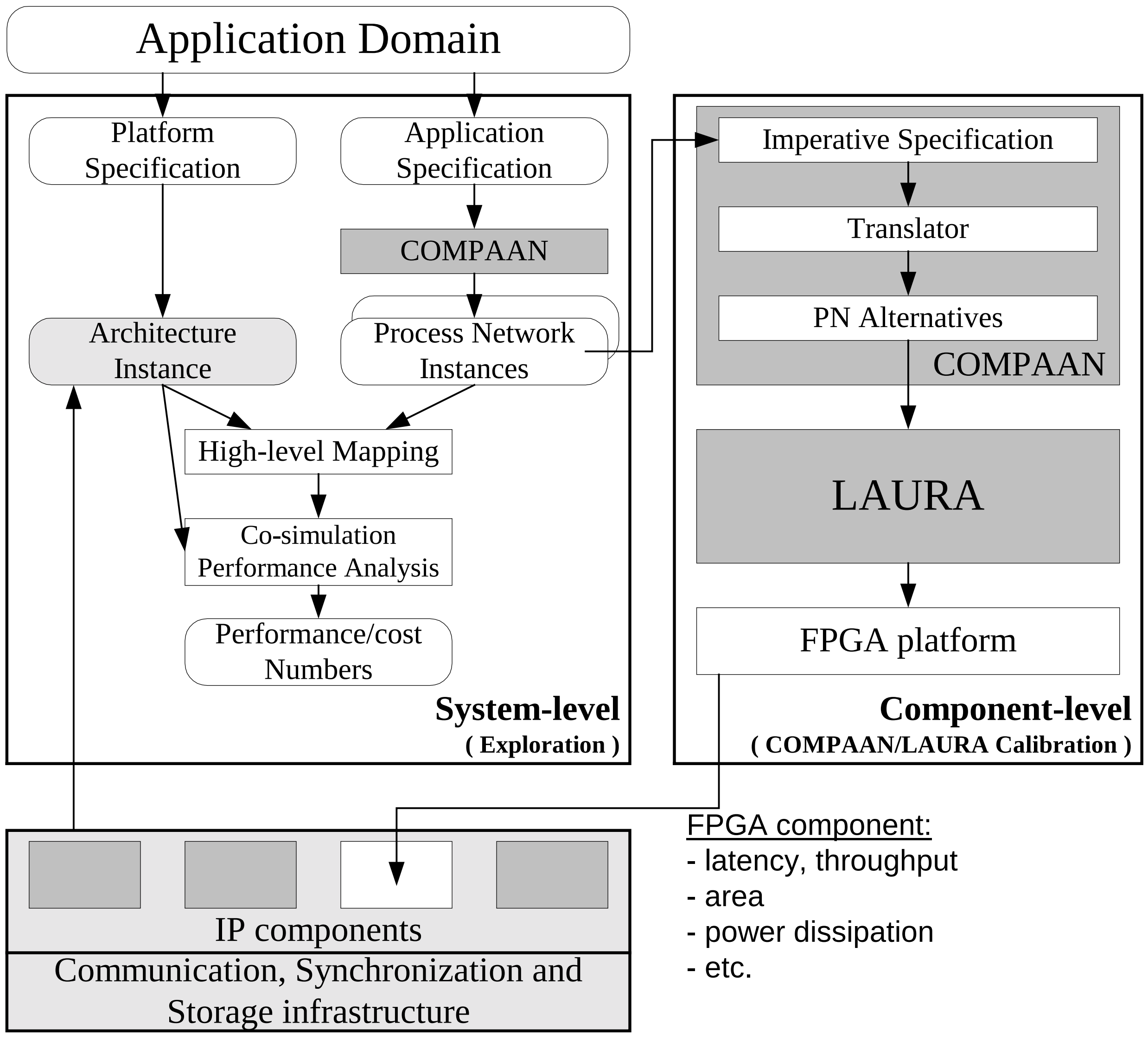

Figure 5. Using the *Compaan* and *Laura* tool sets in a system-level architecture exploration for calibrating the component models of the architecture.

```
for n = 1 : 1 : N,
  for k = 1 : 1 : K,
     y(n) = y(n) + h(k) * x(n-k);
  end
end
```

a)

```
for n = 1 : 1 : N,
  for k = 1 : 1 : K,
     y(n,k+1) = F( y(n,k) , h(n,k) , x(n,k) );
     h(n+1,k) = h(n,k);
     x(n+1,k+1) = x(n,k);
  end
end
```

b)

Figure 6. A simple example program **(a),** and its single assignment version **(b)**.

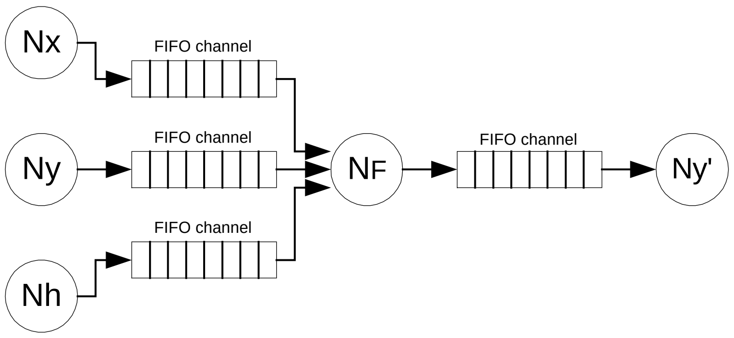

Figure 7. A Kahn Process Network corresponding to the program shown in Figure 6-b).

Now, the *Laura* tool set has to take the KPN to the re-configurable platform for implementation. Because this has to be automated, the processes (nodes) in the KPN must be further structured. Thus in *Compaan*/*Laura*, processes themselves are modeled in terms of the *Stream-based Function* (SBF) [12] model of computation. The SBF model is visualized in Figure 8.

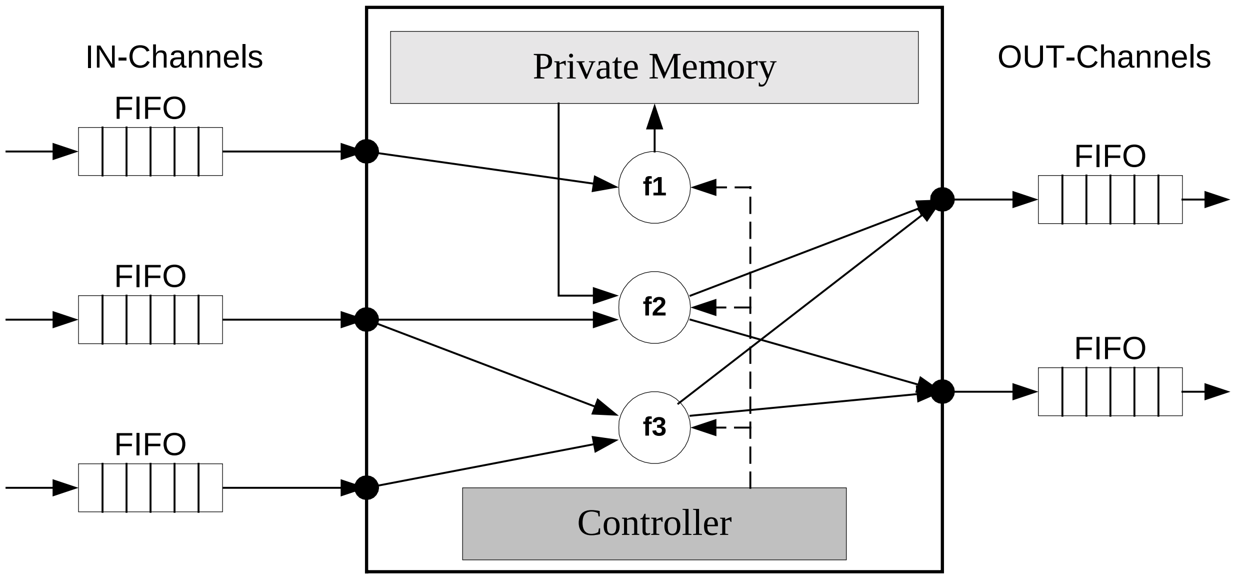

Figure 8. The Stream-based Function model of a KPN Process.

It consists of a private memory, a function repertoire, a controller, and a set of channel selectors. The function repertoire can be either a single function **f()** that takes its arguments from a varying set of channels, or a set of functions that can be invoked sequentially. The private memory stores data that is read from the FIFO channel and is not to be consumed at the time of reading. Whether such a private memory and accompanying control is needed is a compile time task [13]. The private memory, the function repertoire and the channel selection components are pre-designed and available in a library. FIFO channels are likewise pre-defined and available in the library. Thus the process networks – including the process SBF models – provided by *Compaan* can be easily mapped onto re-configurable platforms using the *Laura* tools that convert the *Compaan* models into implementations by invoking the library components. The *Laura* library does not contain implementations of the functions in the function repertoire. These are details that reside on an abstraction level that is below the *Compaan* and *Laura* levels. Typically, detailed implementations of functions are borrowed from partners dealing with IP cores. These cores are imported by *Laura* and integrated in the SBF model.

4. CONCLUSIONS

In this paper, we presented the methodology, devised at the *Leiden Embedded Research Center* (LERC), for designing the new emerging embedded systems. The presented methodology addresses the problem of mastering the complexity of designing such systems by employing the *system-level design*, *platform-based design*, and *separation of concerns* concepts.

The LERC methodology is supported by the *Compaan* and *Laura* tool sets that have been developed and implemented at LERC as well. Currently, these tool sets are application and platform domain-specific, i.e., *streaming data* applications (in particular static, affine nested-loop programs) and a relatively simple platform consisting of a single re-configurable processor, possibly embedded with soft programmable core. As a future work we plan to extend the application and platform domains of *Compaan* and *Laura* in order to support *dynamics* in streaming data applications as well as to support *networked on-chip multiprocessor* platforms.

REFERENCES

1. F. Balarin et al, *Hardware-Software Co-design of Embedded Systems – The POLIS Approach*. Kluwer Academic Publishers, 1997.
2. Bart Kienhuis, Ed Deprettere, Pieter van der Wolf, and Kees Vissers, "A Methodology to Design Programmable Embedded Systems: The Y-Chart Approach," in *Embedded Processor Design Challenges, LNCS 2268* (Ed F. Deprettere, Jurgen Teich, and Stamatis Vassiliadis, eds.), pp. 18–37, Berlin: Springer, 2002.
3. Vladimir Zivkovic, Ed Deprettere, and Pieter van derWolf, "Design Space Exploration of Streaming Multiprocessor Architectures," in *Proc. IEEE Int. Workshop on Signal Processing Systems (SiPS'02)*, (San Diego, CA, USA), pp. 228–234, October 2002.
4. Stephan Edwards, Luciano Lavagno, Edward Lee, and Alberto Sangiovanni-Vincentelli, "Design of Embedded Systems: Formal Models, Validation, and Synthesis," *Proceedings of the IEEE*, vol. 85, pp. 366–390, March 1997.
5. Daniel D. Gajski, Jianwen Zhu, Rainer Domer, Andreas Gerstlauer, and Shuqing Zhao, *SpecC: Specification Language and Methodology*. Kluwer Academic Publishers, 2000.
6. Alberto Sangiovanni-Vincentelli and Grant Martin, "A Vision for Embedded Systems: Platform-Based Design and Software Methodology," *IEEE Design and Test of Computers*, vol. 18, no. 6, pp. 23–33, 2001.
7. Kurt Keutzer, Sharad Malik, Richard Newton, Jan Rabaey, and Alberto Sangiovanni-Vincentelli, "System-Level Design: Orthogonalization of Concerns and Platform-Based Design," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 19, no. 12, pp. 1523–1543, 2000.
8. Edwin Rijpkema, *Modeling Task Level Parallelism in Piece-wise Regular Programs*. PhD thesis, Leiden Embedded Research Center, Leiden Institute of Advanced Computer Science, Leiden University, 2002.
9. Gilles Kahn, "The Semantics of a Simple Language for Parallel Programming," in *Proc. of the IFIP Congress 74*, North-Holland Publishing Co., 1974.
10. Malay Haldar, Anshuman Nayak, Alok Choudhary, and Prith Banerjee, "A System for Synthesizing Optimized FPGA Hardware from Matlab," in *Proc. IEEE/ACM Int. Conf. on Computer Aided Design (ICCAD'01)*, (San Jose, CA, USA), pp. 314–319, November 2001.
11. Jan Frigo, Maya Goghale, and Dominique Lavenier, "Evaluation of the Stream-C C-to-FPGA Compiler: An Applications Perspective," in *Proc. Int. FPGA Conference (FPGA-2001)*, (Monterey, CA, USA), February 2001.
12. l Bart Kienhuis and Ed Deprettere, "Modeling Stream-Based Applications using the SBF Model of Computation," *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technlogy*, vol. 34, no. 3, pp. 291–300, 2003.
13. Alexandru Turjan, Bart Kienhuis, and Ed F. Deprettere, "A Compile-time based Approach for Solving Out-of-Order Communication in Kahn Process Networks," in *Proc. IEEE International Conference on Application-specific Systems, Architectures, and Processors (ASAP'2002)*, (San Jose, USA), July 17-19 2002.