Verification by translation to UML-B

Dr. Colin Snook
January 2010
Outline

• Overview of UML-B (and Event-B)
• Features of the UML model
• Translation to UML-B
• Verifying a Safety Requirement
• Conclusion
Event-B

- Systems modelling language
  - Closed Systems - No environment - No concept of ‘calling’
  - State - Represented by Typed Variables
  - Spontaneous Transitions - Guarded events that alter the state
  - Refinement - More detailed state, reveals more detailed events

- Formal modelling environment and toolset (Rodin)
  - Incremental static checker
  - Proof tools
  - Extensible
Small example of Event-B

**VARIABLES**
- flag
- count

**INVARIANTS**
- inv1: flag ∈ BOOL
- inv2: count ∈ N

**ACTION**
```
ANY
  p
WHERE
  grd1: p ∈ Z
  grd2: flag = TRUE
THEN
  act1: count = count + p
END
```
UML-B

- Diagrammatic front end for Event-B
- Close integration with Rodin platform for Event-B
  - Automatic translation to Event-B
  - Rodin static checker errors fed back to diagram
  - Provers and Model Checkers
- ‘UML-like’
  - Class Diagrams and Statemachines
  - Diagramatic notations for Refinement
- Borrows from Event-B
  - Invariants, axioms, guards, actions
Class Refinement

New data

New invariants

Refinement
State Machine Refinement

Nested state machines

Split transition (refined in nested state machines)
At each refinement

- Refine Data and Events
  - New class attributes and class events
  - New nested state machines and transitions

- Add invariants
  - specify that a hazardous state is never reached

- Add guards to events/transitions
  - to ensure invariants are obeyed

- Prove
  - That the behaviour doesn’t violate the invariants
  - That the level refines the previous level
Verification

Input material

Generis xUML
(from Euro-Interlocking)

Intermediate step

Epsilon UML
(This version of the model should be available from work by York)

Modelling/Verification itself

UML-B model

Rodin modelling platform with built in verification tools inc. prover and animator. Model is built in layers of ‘refinement’

Safety requirements are represented by invariants embedded within the UML-B model

Goal

Understandable
A layered model, showing increasing detail in gradual steps

Consistent
Proof shows that the model is ‘well-formed’

Safe
Proof shows that the model satisfies the formalised safety invariants

Valid
Animation/testing shows that the model is equivalent to the input model
**Principle**

- Can’t prove that a model is the correct one (valid)
  - The prover doesn’t know what we want
  - Need Animation for this

- We can prove 2 specifications are consistent
  - A behavioural model satisfies an invariant
  - A model refines another one

- Why is this useful?
  - Say something in a simple way (abstraction)
    - More likely to be correct
  - Prove it remains correct when we refine it
UML model of Interlocking:
Translation to UML-B

Dr. Colin Snook
January 2010
UML Domain Model
UML Behaviour model - Points

external trigger

internal trigger

guard

timeout

ignore send to GUI & railyard
UML Behaviour model - Routes

send trigger message to points

spontaneous transition
UML Domain Model
UML-B Refinements
UML-B - Mo + Co
Modelling message passing

Co

Mo

ClassTypes (given sets)

buffer_owner= BUFFEROWNER

Attributes
buffer: ↯(MESSAGE)

Context Axiom
BUFFEROWNER ≠ ∅

Context Axiom
MESSAGE ≠ ∅

Constant
set_stop: MESSAGE

Constant
set_proceed: MESSAGE

Constant
free: MESSAGE

Constant
occupied: MESSAGE

Class

Action: buffer :∈ BUFFEROWNER → ↯(MESSAGE)
UML-B - M3 Introducing Points

subtypes BUFFEROWNER
UML Behaviour model - Points

- **external trigger**
- **internal trigger**
- **guard**
- **timeout**

ignore send to GUI & railyard
UML-B - M3 point_sm

Guard: move_left ∈ buffer(self)
Action: buffer(self) = buffer(self) \ {move_left}
UML-B - M4 Elaborating Points

From any state
UML-B - M5 Elaborating Points - Moving
UML-B Route

Axioms reqd by prover

track layout constants

FMCO Graz 29th Nov 2010
UML Behaviour model - Routes

trigger message to points
UML-B Route state-machine

Action:
buffer := (buffer ↓
{self ↦ buffer(self) \ {set_route}})
↓
(λlp · lp ∈ left_points[{self}] | buffer(lp) u {move_left})
↓
(λrp · rp ∈ right_points[{self}] | buffer(rp) u {move_right})
UML-B Route state-machine

Guard1:
\[ \forall lp \cdot lp \in \text{left\_points}\{\text{self}\} \Rightarrow
lp \in \text{dom}(\text{working\_sm}) \land \text{working\_sm}(lp) = \text{left\_st} \]

Guard2:
<same for right points>

Guard3:
\[ \forall t \cdot t \in \text{tracks}\{\text{self}\} \Rightarrow
\quad t \in \text{dom}(\text{ready\_sm}) \land \text{ready\_sm}(t) = \text{free\_st} \]
Proof stats .... so far

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Total</th>
<th>Auto</th>
<th>Manual</th>
<th>Reviewed</th>
<th>Undischarged</th>
</tr>
</thead>
<tbody>
<tr>
<td>micro2010.eventB</td>
<td>200</td>
<td>155</td>
<td>16</td>
<td>0</td>
<td>29</td>
</tr>
<tr>
<td>c0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m7_implicitContext</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m8_SR0001_implicitContext</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m1</td>
<td>11</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m2</td>
<td>18</td>
<td>18</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m3</td>
<td>15</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m4</td>
<td>36</td>
<td>31</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m5</td>
<td>30</td>
<td>26</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m6</td>
<td>26</td>
<td>23</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m7</td>
<td>30</td>
<td>22</td>
<td>2</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>m8_SR0001</td>
<td>31</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>23</td>
</tr>
</tbody>
</table>
The prover helps debug models
UML Safety Requirement 1

S-0001 A Point that is locked by an established route shall never move

INVARIANT ‘no_move’:

\[ \forall r,p \cdot \]

\[ r \in \text{route} \land \]
\[ \text{route_sm}(r) = \text{established_st} \land \]
\[ p \in (\text{left_points[\{r\}]} \cup \text{right_points[\{r\}]}) \]
\[ \Rightarrow \]
\[ (p \in \text{dom(working_sm)} \]
\[ \Rightarrow \]
\[ \text{working_sm}(p) \neq \text{moving_st} \]

r is a route
r is established
p is a point of r
if p is working
then
p is not moving
Reflection

• The original model isn’t correct, but even if it was...

• The Safety Requirements look like they will be difficult to prove

• The refinements are based on the model structure, not on abstraction of the essential properties

• A more abstract model of safe behaviour would allow us to use refinement in a more progressive way (wrt proof)
Summary

• Work in Progress

• UML-B version of UML model
  – fairly close correspondence... but not identical
  – refinement helps get the model right (consistency proofs)
  – proof is mostly quite easy (but not proving much)

• Safety Invariant
  – proofs look quite hard, maybe impossible
  – PO’s will drive the model
  – may end up with a different model
    • (careful no to deviate from UML?)
Thank you

Questions?