A Formal Framework for UML Modeling with Timed Constraints: Application to Railway Control Systems

Rafael Marcano
Samuel Colin
Georges Mariano

Specification and Validation of UML models for Real Time and Embedded Systems – SVERTS’2004
UML/OCL as formal notation

- *Standard notation for systems modeling*
- *But lack of formal semantic*

Our context: railway control systems

*Is actually UML/OCL formal enough to specify reliable software?*
...UML/OCL still informally defined
...insecure basis for software development

How to specify precisely?
How to prove rigorously? ...

Using the B Formal Method

- **Formal methods are used to:**
  - specify precisely the components of a system
  - prove rigorously the desired structural/behavioral properties (analysis and verification)
1. Objective
2. The proposed process
3. From requirements to UML
4. From UML models to B specifications
   - mapping \textit{UML diagrams} into \textit{B formal specifications}
   - translating \textit{time features} into \textit{extended B}
5. Analysis and verification
6. Conclusion and discussion
I. Objective

Problem

How to enable analysis and verification on UML/OCL models with time properties?

Solution

- By combining the complementary strengths of UML/OCL and the B formal method
- By extending B with a real time semantic
- By providing automated proof tools for the B specification language
Advantages of the B formal method

- precise and rigorous description of a system
- detection of inconsistencies and mistakes on the UML model
- proof of structural and behavioral properties
- ensuring consistency between diagrams using only one formalism
- supporting refinement

But it does not support timed characteristics
II. The Proposed Process

1. Requirements elicitation
   Create a valid UML description of the required system

2. Formal specification
   Derive a B formal specification from UML diagrams

3. Analysis and verification
   Generate proof obligations and prove them
III. From Requirements to UML

A railway level crossing system [JS’00]
Scenario of an approaching train

1. **detectTrain(Arriving)**
   - **[yellowLight.state=Off]** switchOn()
   - **[redLight.state=Off]** switchOn()

2. **ackRequest**
   - **[yellowLight.state=On]** switchOff

3. **receiveAck**
   - **[theBarrier.state=Opened]** closeBarrier()
   - **[bSensor.status<>Closed or t'-t>6sec]** setMode(Unsafe)
   - **[bSensor.status=Closed and t'-t<=6sec]** setMode(Safe)

4. **getBarrierStatus**
   - **{t .. t+3sec}**
   - **{t=now}**
   - **{t .. t+12sec}**
State diagram of the LCC system

- **Activated**
  - **Yellow LightOn**
    - Transition: `trainDetectionEntry / yellowLightOn()`
    - Transition: `timeOut_1 / redLightOn()`
    - Transition: `[bSensor.status=Closed] / setMode(Safe)`
    - Transition: `timeOut_2 / closeBarrier()`
  - **Red LightOn**
    - Transition: `timeOut_3`
  - **Closed**
    - Transition: `[bSensor.status=Opened] / setMode(Unsafe)`
  - **Opening Barrier**
    - Transition: `trainDetectionRear / openBarrier()`
  - **Closing Barrier**

- **Deactivated**
  - Transition: `deactivate`

- **Failure**
  - Transition: `repair`
  - Transition: `failure`
OCL invariants and pre/post-conditions

context LCC_System
inv 1:
  self.theBarrier.state=Closed implies self.redLight.state=On and
  self.theBarrier.state=Closing implies self.yellowLight.state=On and
  self.yellowLight.state=Off and self.redLight.state=Off implies self.theBarrier.state=Open

inv 2:
  self.state=Activated and self.bSensor.state=Open implies self.mode=Unsafe

inv 4:
  self.bSensor.state=Open and self.theBarrier.state=Closed implies self.mode=Unsafe

context LCC_System :: openBarrier
pre:
  self.theBarrier.state=Closed and self.mode=Safe
post:
  self.theBarrier.state=Open
IV. From UML to B

... mapping UML diagrams into B formal specifications

Formal semantics for UML/OCL

Why B?

UML: collections of objects ↔ B: sets

How?

Transformation rules of UML/OCL into B

context Class_i::operation(arg:Type)
pre: predicate (class_i, arg)
post: predicate (class_j, arg)
The diagram represents a formalization of class diagrams using UML. It includes classes such as `BarrierSensor`, `Light`, `LevelCrossingControl`, `VehicleSensor`, `Barrier`, and `OperationsCenter`. Each class is connected by associations and operations. For example, `Barrier` has operations `close()`, `open()`, and attributes `state : bSTATE`, `theBarrier1`. The diagram also shows interactions among these classes, such as `sendSignal()` in `VehicleSensor` and `operateBarrier()` in `OperationsCenter`.

The text accompanying the diagram explains the formalization process, including the use of state machines and operations for each class.
Translation of a class:

**MACHINE** Barrier

**USES** BarrierSensor

**SETS** BARRIER

**VARIABLES** barrier, bState

**INVARIANT**

\[
\text{barrier} \subseteq \text{BARRIER} \land \text{bState} \in \text{barrier} \rightarrow \text{bSTATE}
\]

**INITIALISATION**

\[
\text{barrier}, \text{bState} := \emptyset, \emptyset
\]

**OPERATIONS**

openBarrier(obj) =

**PRE** \( obj \in \text{barrier} \land \text{bState}(obj) = \text{Closed} \)

**THEN**

\( \text{bState}(obj) := \text{Opened} \)

**END**;

closeBarrier(obj) =

**PRE** \( obj \in \text{barrier} \land \text{bState}(obj) = \text{Opened} \)

**THEN**

\( \text{bState}(obj) := \text{Closed} \)

**END**;
Part of the B specification – Machine LCC_System:

```
MACHINE LCC_System
SETS LCC;
    STATE={Deactivated, ShowingYlight, ShowingRlight, ClosingB, OpeningB, ClosedB, Failure};
    MODE={Safe, Unsafe}
INCLUDES Barrier, BarrierSensor, Red.Light, TrainborneCS, Yellow.Light
VARIABLES
    lcc, lcc_barrier, lcc_sensor, state, redLight, yellowLight, lcc_train, mode
INVARIANT
    lcc ⊆ LCC ∧
    lcc_barrier ∈ lcc → barrier ∧
    lcc_sensor ∈ lcc → bSensor ∧
    state ∈ lcc → STATE ∧
    redLight ∈ lcc → Red.light ∧
    yellowLight ∈ lcc → Yellow.light ∧
    lcc_train ∈ lcc → train ∧
    mode ∈ lcc → MODE ∧ ...
```
Machine LCC_System – operations:

```plaintext
OPERATIONS
timeOut_1_showRlight(obj) =
  PRE
    obj ∈ lcc ∧
    state(obj) = ShowingYlight ∧
    bStatus(lcc_sensor(obj)) = Opened ∧
    bState(lcc_barrier(obj)) = Opened ∧
    Red.lState(redLight(obj)) = Off ∧
    Yellow.lState(yellowLight(obj)) = On
  THEN
    state(obj) := ShowingRlight ||
    Yellow.switchOff(yellowLight(obj)) ||
    Red.switchOn(redLight(obj))
  END;

timeOut_2_closeBarrier(obj) =
  PRE
    obj ∈ lcc ∧
    state(obj) = ShowingRlight ∧
    Red.lState(redLight(obj)) = On ∧
    Yellow.lState(yellowLight(obj)) = Off
  THEN
    state(obj) := ClosingB ||
    closeBarrier(lcc_barrier(obj))
  END;

timeOut_3_setMode(obj) =
  PRE
    obj ∈ lcc ∧
    state(obj) = ClosingB ∧
    bState(lcc_barrier(obj)) = Closed ∧
    Red.lState(redLight(obj)) = On ∧
    Yellow.lState(yellowLight(obj)) = Off
  THEN
    SELECT
      bStatus(lcc_sensor(obj)) = Closed THEN
        state(obj) := ClosedB ||
        mode(obj) := Safe
    WHEN
      bStatus(lcc_sensor(obj)) = Opened THEN
        state(obj) := Failure
    ELSE skip
  END
END;
```

...
Machine LCC_System – invariant:

\[ \forall \text{obj}. (\text{obj} \in \text{lcc} \land \text{bState(lcc\_barrier(obj))}=\text{Closed} \Rightarrow \text{Red.IState(redLight(obj))}=\text{On}) \land \]

\[ \forall \text{obj}. (\text{obj} \in \text{lcc} \land \text{bState(lcc\_barrier(obj))}=\text{Closing} \Rightarrow \text{Red.IState(yellowLight(obj))}=\text{On}) \land \]

\[ \forall \text{obj}. (\text{obj} \in \text{lcc} \land \text{Yellow.IState(yellowLight(obj))}=\text{Off} \land \text{Red.IState(redLight(obj))}=\text{Off} \Rightarrow \text{bState(lcc\_barrier(obj))}=\text{Opened}) \land \]

\[ \forall \text{obj}. (\text{obj} \in \text{lcc} \land \text{state(obj)}\in \text{Activated} \land \text{bStatus(lcc\_sensor(obj))}=\text{Opened} \Rightarrow \text{mode(obj)}=\text{Unsafe}) \land \]

\[ \forall \text{obj}. (\text{obj} \in \text{lcc} \land \text{bStatus(lcc\_sensor(obj))}=\text{Opened} \land \text{bState(lcc\_barrier(obj))}=\text{Closed} \Rightarrow \text{mode(obj)}=\text{Unsafe}) \]
Classical B

- Specifying a clock abstract machine
- Defining abstract variables holding the times

Time extended B with timed substitutions

- Semantic extension of B substitutions:
  \[ \rightarrow \text{Derivative of } \text{Duration Calculus} \ (WDC^*) \]
Time extended B
operations including time constraints ...

Machine LCC_System – classical B with time variables:

closeBarrier(obj) =
PRE obj ∈ barrier ∧ bState(obj)==Opened
THEN bState(obj):=Closed || ANY newtime WHERE
   newtime ∈ N ∧
   newtime ≥ time ∧
   newtime ≤ time+ClosingDelay
THEN
   setTime(newtime)
END;

timeout_2_closeBarrier(obj) =
PRE obj ∈ lcc ∧ state(obj)==ShowingRlight ∧
   Red.lState(redLight(obj))=On ∧
   Yellow.lState(yellowLight(obj))=Off ∧
   time – trainDetectionEntry_Time ≤ 12
THEN
   state(obj):=ClosingB ||
   closeBarrier(lcc_barrier(obj)) ||
   closeBarrierTime:=time
END;

timeOut_3_setMode(obj) =
PRE obj ∈ lcc ∧ state(obj)==ClosingB ∧
   bState(lcc_barrier(obj))=Closed ∧
   Red.lState(redLight(obj))=On ∧
   Yellow.lState(yellowLight(obj))=Off
THEN
   SELECT bStatus(lcc_sensor(obj))=Closed ∧
   time – closeBarrierTime ≤ 6
   state(obj):= ClosedB ||
   mode(obj):=Safe
   WHEN bStatus(lcc_sensor(obj))=Opened ∧
   time – closeBarrierTime > 6
   state(obj):= Failure
   ELSE skip
END;
### Time extended B

#### timed substitutions

**Classic B substitutions**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GSL</td>
<td>[GSL]P</td>
<td>dur([GSL]P)</td>
</tr>
<tr>
<td><code>skip</code></td>
<td><code>P</code></td>
<td><code>[[ ] ]</code></td>
</tr>
<tr>
<td><code>x := E</code></td>
<td><code>P [E/x]</code></td>
<td><code>[[ ] ]</code></td>
</tr>
<tr>
<td>`g</td>
<td>S`</td>
<td><code>g ∧ [S]P</code></td>
</tr>
<tr>
<td><code>g ⇒ S</code></td>
<td><code>g ⇒ [S]P</code></td>
<td></td>
</tr>
<tr>
<td><code>S ; T</code></td>
<td><code>[S] ([T]P)</code></td>
<td></td>
</tr>
<tr>
<td><code>delay d</code></td>
<td><code>l = d ∧ [[P ]]</code></td>
<td></td>
</tr>
</tbody>
</table>

**Extended Hoare Triplets**

\[
\begin{align*}
\{[S]I\} [S,dur([S], I)] \{I\} & \quad I ∧ C ⇒ [S]I \quad I ∧ ¬ C ⇒ P \\
\{I\} \begin{cases}
\text{WHILE } C \\
\text{DO } S
\end{cases} & \quad , \text{dur}([S], I)^* \quad \{P\} \\
\text{INVARIANT } I
\end{align*}
\]
Timed substitutions

\[ x \gets \text{Example1} = \]

\begin{verbatim}
TIMING
  PRE
    x \geq 1
  THEN
    delay 1; x:=x-1;
  END
  POST
    x \geq 0
  REQUIRES
    \square ( \square x \geq 0 )
END
\end{verbatim}
V. Analysis and verification

Generating Proof Obligations (POs)

Abstract Machines

MACHINE Systeme
INCLUDES
  Classe_i, Classe_j
VARIABLES
  ass_ij
INVARIANT
  ass_ij ∈ classe_i ↔ classe_j
OPERATIONS

Errors detection

Proof Obligations

Generation of Proof Obligations (POs)

Unsuccessful proofs

The prove process facilitate errors detection

- Syntax and type errors
- Incompleteness of pre-conditions
- Inconsistent post-conditions
Classical B with time variables

- *Does not allow complex timed properties* (liveness and fairness)
- *Enough to railway control systems*
- *Easiest implementation*

Extended B with duration calculus

- *Timing diagrams needed*
- *Allows to express complex timing properties*

Event B ? → CTL semantics
Advantages of using B

- Analysis, verification and simulation: coherence/consistency of UML/OCL
- Use of existing proof tools
- Refinement
- Successfully used in railway applications