Semantic Adaptation for Models of Computation
ACSD 2011 in Newcastle

Frédéric Boulanger
Cécile Hardebolle, Christophe Jacquet, Dominique Marcadet

Supélec E3S – Computer Science Department

June 24 2011
1. Introduction

2. Languages and models of computation

3. Semantic adaptation

4. Conclusion
Context

Modeling and validation of embedded software systems

- Tight coupling with the environment
- Cost of errors
Heterogeneity

- Different specialties
- Different methods and different modeling tools

Starmac (Stanford/Berkeley)

Differential equations
Functions of a complex variable
Mechanics
Logics systems
Control science
State machines
Signal processing
Aerodynamics
Telecommunications
Discrete events
Energy
Sampled systems
Plan

1. Introduction
2. Languages and models of computation
3. Semantic adaptation
4. Conclusion
Heterogeneous modeling

- Necessity to handle several modeling languages
- Principle: language $\rightarrow$ model of computation (common syntax)

Model of computation

- Set of rules for composing the behavior of the components of a model.
- Semantics of the structure of the model
- Algorithm for solving the rules $\Rightarrow$ model of execution
Heterogeneous modeling

- Necessity to handle several modeling languages
- Principle: language $\rightarrow$ model of computation (common syntax)

Model of computation

- Set of rules for composing the behavior of the components of a model.
- Semantics of the structure of the model
- Algorithm for solving the rules $\Rightarrow$ model of execution

MoC = FSM

A

B
Heterogeneous modeling

• Necessity to handle several modeling languages
• Principle: language $\rightarrow$ model of computation (common syntax)

Model of computation

• Set of rules for composing the behavior of the components of a model.
  Semantics of the structure of the model
• Algorithm for solving the rules $\Rightarrow$ model of execution

MoC = FSM

MoC = CSP
Modeling Framework

- Define models of computation (model semantics)
- Define semantic adaptation between heterogeneous models

Design principles

- Meta-model + generic execution engine:
  - schedule: choose the next component to be observed
  - update: update the interface of a component
  - propagate: propagate information toward other components
- Behavior of a model = fixed point of $propagate \circ update \circ schedule$
- Existence, unicity, reachability by iteration of the fixed point?
Unit of behavior: Block, “black box” with an interface
Unit of behavior: Block, “black box” with an interface
Unit of interface: Pin, for sending and getting information
Unit of behavior: Block, “black box” with an interface
Unit of interface: Pin, for sending and getting information
Structure: Relations between pins, semantics is defined by the MoC
Generic Meta-model

Unit of behavior: Block, “black box” with an interface
Unit of interface: Pin, for sending and getting information
Structure: Relations between pins, semantics is defined by the MoC
Model: Model = structure + MoC
Unit of behavior: **Block**, “black box” with an interface

Unit of interface: **Pin**, for sending and getting information

Structure: **Relations** between pins, semantics is defined by the MoC

Model: **Model** = structure + MoC

Hierarchical heterogeneity

**InterfaceBlock**: behavior is described by a **Model**
Semantic Adaptation for Models of Computation

Languages and models of computation

June 24 2011
Behavior of a model = series of observations

Synchronous approach to the observation of models:
- no communication with the environment during the snapshot;
- no change in the internal state of the blocks.
Computation of a fixed point by iteration:
• sequential observation of the blocks;
• update of their interface;
• propagation of the information according to the relations between pins.

Schedule, Propagate and Done are the three operations which define a MoC.
Update represents the observable behavior of the blocks.
It is possible to reject the fixed point which has been reached.

Search for another fixed point from different initial conditions.
1 Introduction

2 Languages and models of computation

3 Semantic adaptation

4 Conclusion
Heterogeneous modeling $\Rightarrow$ definition of different MoCs

Interaction between models that use different models of computation?

How to combine

- State machines
- Block diagrams
- Process networks
- Discrete systems
- Continuous systems
Heterogeneous modeling $\Rightarrow$ definition of different MoCs

Interaction between models that use different models of computation?

How to combine

- State machines
- Block diagrams
- Process networks
- Discrete systems
- Continuous systems

How to tame the beast?
Hierarchical composition

• Approach used in Ptolemy and ModHel’X
• Each hierarchical level uses only one MoC
• MoCs are combined by pairs only
ModHel’X: semantic adaptation along three axes

- Adaptation of **data**: different MoCs may use different kinds of data
- Adaptation of **time**: different notions of time are used in different MoCs
- Adaptation of **control**: instants at which a model should be observed depend on the MoC

EDONA project

- Precise definition of MoCs from the automotive domain
- Design of configurable semantic adaptation patterns
- Example of a power window
Example: power window
Example: power window

*Semantic adaptation*

June 24 2011 14 / 22
Semantic adaptation

Example: power window
Example: power window
Semantic adaptation

Example: power window
Adaptation of data between DE and TFSM

<table>
<thead>
<tr>
<th>value</th>
<th>symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td></td>
</tr>
</tbody>
</table>
Adaptation of data between DE and TFSM

\[ \text{symbol} = f(\text{value}) \]

- **value**
- **date**
- **symbol**
Adaptation of data between DE and TFSM

\[ \text{symbol} = f(\text{value}) \]

\[ \text{value} = g(\text{symbol}) \]
\[ \text{date} = \text{DE.now} \]
Adaptation of data between DE and TFSM

\[
\text{value} = g(\text{symbol}) \quad \text{date} = \text{DE.now}
\]

\[
\text{symbol} = f(\text{value})
\]

**Cmd Interface**

- **cmd**: cmd_neutral, cmd_up, cmd_down
- **motor**: motor_stop, motor_up, motor_down
Adaptation of data between DE and SDF

Semantic adaptation (data)
Semantic adaptation (data)

Adaptation of data between DE and SDF

- Relative position of events / samples ⇒ adaptation of time
- Presence of samples ⇒ adaptation of control
Semantic adaptation (data)

Adaptation of data between DE and SDF

DE

<table>
<thead>
<tr>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
</tr>
</tbody>
</table>

SDF

<table>
<thead>
<tr>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_2$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_2$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_3$</td>
</tr>
</tbody>
</table>
Adaptation of time

Time in DE

- Time stamps in $\mathbb{R} \times \mathbb{N}$
- Synchronization, causality
- Controls when events are processed

Time in TFSM

- Measure of the elapsed time in a state
- Time stamps have no special meaning

Adaptation DE – TFSM

- Mapping from TFSM durations to differences between DE time stamps
- Consequences on control
Adaptation of time

Time in DE

- Time stamps in $\mathbb{R} \times \mathbb{N}$
- Synchronization, causality
- Controls when events are processed

Time in SDF

- Series of samples, no real notion of date
- No notion of duration

Adaptation DE – SDF

- Give a DE time stamp to SDF data samples
- Sampling period, consequences on control
Adaptation of control

TFSM → DE → SDF
Adaptation of control

Event in DE, no input for the state machine
Semantic adaptation (control)

Adaptation of control

TFSM

DE

SDF

Periodic control for SDF
Semantic adaptation (control)

Adaptation of control

- Periodic control for SDF $\Rightarrow$ control in DE
- Event in DE, no input for the state machine $\Rightarrow$ control in TFSM
- Firing of a timed transition in the state machine $\Rightarrow$ control in DE
- Periodic observation of SDF $\Rightarrow$ control in DE + event for the state machine $\Rightarrow$ control in TFSM
- Data never creates control for SDF $\Rightarrow$ data is memorized at the DE-SDF interface
Adaptation of control

Event in DE, input for the state machine
⇒ control in TFSM
Adaptation of control

Firing of a timed transition in the state machine
Adaptation of control

Firing of a timed transition in the state machine
⇒ control in DE
Adaptation of control

Periodic observation of SDF
⇒ control in DE + event for the state machine
⇒ control in TFSM
Adaptation of control

Data never creates control for SDF
Adaptation of control

Data never creates control for SDF

⇒ data is memorized at the DE-SDF interface
Today

- Adaptation of data: configurable semantic adaptation patterns
- Adaptation of time: ad hoc handling in interface blocks
- Adaptation of control: constraints on the date of the next snapshot

The future

- Adaptation of data: special MoC?
- Adaptation of time and control: clock calculus
  (Clock Constraint Specification Language)
1 Introduction
2 Languages and models of computation
3 Semantic adaptation
4 Conclusion
Conclusion

Multi-paradigm modeling

• Accept heterogeneity
• Handle heterogeneity at the modeling level
• Many special purpose tools (Simulink, System C-AMS)

Our approach

• Unifying syntax for heterogeneous models
• Definition of models of execution
• Explicit description of the semantic adaptation between models

Key points

• Well defined semantics for models and combinations of models
• Joint use of already existing tools
• Modularity (black box approach)
Thanks for your attention

Questions?
Synchronous data flow MoC

- Instantaneous propagation of data
- Consumption and production of a fixed number of data samples
- Snapshot = shortest production/consumption cycle
Example of computation of a snapshot

Synchronous data flow MoC

- Instantaneous propagation of data
- Consumption and production of a fixed number of data samples
- Snapshot = shortest production/consumption cycle

\[ x_{[2]} \]

StartOfSnapshot
Example of computation of a snapshot

Synchronous data flow MoC

- Instantaneous propagation of data
- Consumption and production of a fixed number of data samples
- Snapshot = shortest production/consumption cycle

Schedule $\rightarrow$ ramp
Example of computation of a snapshot

Synchronous data flow MoC

- Instantaneous propagation of data
- Consumption and production of a fixed number of data samples
- Snapshot = shortest production/consumption cycle

Update ramp
Example of computation of a snapshot

Synchronous data flow MoC

- Instantaneous propagation of data
- Consumption and production of a fixed number of data samples
- Snapshot = shortest production/consumption cycle

Semantic Adaptation for Models of Computation

Appendix

June 24 2011 24 / 22
Example of computation of a snapshot

Synchronous data flow MoC

- Instantaneous propagation of data
- Consumption and production of a fixed number of data samples
- Snapshot = shortest production/consumption cycle

Schedule $\rightarrow$ ramp
Example of computation of a snapshot

Synchronous data flow MoC

- Instantaneous propagation of data
- Consumption and production of a fixed number of data samples
- Snapshot = shortest production/consumption cycle

Update ramp
Example of computation of a snapshot

**Synchronous data flow MoC**

- Instantaneous propagation of data
- Consumption and production of a fixed number of data samples
- Snapshot = shortest production/consumption cycle

![Diagram](image)

Propagate
Example of computation of a snapshot

Synchronous data flow MoC

- Instantaneous propagation of data
- Consumption and production of a fixed number of data samples
- Snapshot = shortest production/consumption cycle

Schedule → average
Example of computation of a snapshot

**Synchronous data flow MoC**

- Instantaneous propagation of data
- Consumption and production of a fixed number of data samples
- Snapshot = shortest production/consumption cycle

\[
\bar{x}[2] = 1.5
\]

**Update average**
Example of computation of a snapshot

Synchronous data flow MoC

- Instantaneous propagation of data
- Consumption and production of a fixed number of data samples
- Snapshot = shortest production/consumption cycle

Propagate
Example of computation of a snapshot

Synchronous data flow MoC

- Instantaneous propagation of data
- Consumption and production of a fixed number of data samples
- Snapshot = shortest production/consumption cycle

\[
\bar{x}_{[2]} \rightarrow 1.5 \rightarrow \text{display}
\]

Schedule \(\rightarrow\) display
Example of computation of a snapshot

Synchronous data flow MoC

- Instantaneous propagation of data
- Consumption and production of a fixed number of data samples
- Snapshot = shortest production/consumption cycle

Update display
Example of computation of a snapshot

Synchronous data flow MoC

- Instantaneous propagation of data
- Consumption and production of a fixed number of data samples
- Snapshot = shortest production/consumption cycle

\[ x_2 \]

Done
Example of computation of a snapshot

Synchronous data flow MoC

- Instantaneous propagation of data
- Consumption and production of a fixed number of data samples
- Snapshot = shortest production/consumption cycle

Validate
Example of computation of a snapshot

Synchronous data flow MoC

- Instantaneous propagation of data
- Consumption and production of a fixed number of data samples
- Snapshot = shortest production/consumption cycle

EndOfSnapshot
Example of computation of a snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial inputs)
- Snapshot = all signals are known

![Diagram of a synchronous reactive MoC circuit]
Example of computation of a snapshot

**Synchronous reactive MoC**

- Instantaneous propagation of data
- Non-strict blocks (may react to partial inputs)
- Snapshot = all signals are known

![Diagram of a signal flow graph](image)
Example of computation of a snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial inputs)
- Snapshot = all signals are known

Schedule $\rightarrow$ A
Example of computation of a snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial inputs)
- Snapshot = all signals are known

Diagram:

Update A
Example of computation of a snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial inputs)
- Snapshot = all signals are known

```
A

1

0

B

Schedule → B
```
Example of computation of a snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial inputs)
- Snapshot = all signals are known

Update B
Example of computation of a snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial inputs)
- Snapshot = all signals are known

Propagate
Example of computation of a snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial inputs)
- Snapshot = all signals are known

Schedule → A
Example of computation of a snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial inputs)
- Snapshot = all signals are known

Update A
Example of computation of a snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial inputs)
- Snapshot = all signals are known

```
A
  1
  0
B
  0
  1
```

Propagate
Example of computation of a snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial inputs)
- Snapshot = all signals are known

Schedule $\rightarrow$ B
Example of computation of a snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial inputs)
- Snapshot = all signals are known

Update B
Example of computation of a snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial inputs)
- Snapshot = all signals are known

Done
Example of computation of a snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial inputs)
- Snapshot = all signals are known

Validate
Example of computation of a snapshot

Synchronous reactive MoC

- Instantaneous propagation of data
- Non-strict blocks (may react to partial inputs)
- Snapshot = all signals are known

EndOfSnapshot
Example of non validation in CT

Semantic Adaptation for Models of Computation
Appendix
June 24 2011

Continuous Director

Temperature

Guard: true
Set:
heating.Integrator.initialState = 16.0;
heating.HeatSource.value = 9.0;
heating.LevelCrossingDetector.level = 20.0

Guard: output < 19.0
Set:
heating.HeatSource.value = 9.0;
heating.LevelCrossingDetector.level = 20.0

Guard: event_isPresent
Set:
heating.HeatSource.value = 0.0;
heating.LevelCrossingDetector.level = 19.0
Example of non validation in CT

State = heating
Example of non validation in CT

State = heating

$\Delta T$ determined according to the precision on $T$
Example of non validation in CT

21°C
20°C
19°C
18°C
17°C
16°C
15°C

State = heating
Example of non validation in CT

State = heating

△T determined according to the precision on T

The upper threshold is reached, but we don't know when.

The snapshot is not valid

The date is precise enough.

State changes

State = cooling

\[ \Delta T < \varepsilon \]
Example of non validation in CT

State = heating
Example of non validation in CT

The upper threshold is reached, but we don’t know when.
The snapshot is not valid.
Example of non validation in CT

State = heating
Example of non validation in CT

\( \Delta T < \epsilon \)

State = heating

The upper threshold is reached, the date is precise enough.

State changes
Example of non validation in CT

State = cooling
Example of non-validation in CT

\[ \Delta T > \varepsilon \]

State = cooling
Example of non validation in CT

State = cooling
Example of non validation in CT

- $15^\circ C$
- $16^\circ C$
- $17^\circ C$
- $18^\circ C$
- $19^\circ C$
- $20^\circ C$
- $21^\circ C$

State = heating

$\Delta T$ determined according to the precision on $T$

The upper threshold is reached, but we don’t know when.

The snapshot is not valid

The upper threshold is reached, the date is precise enough.

State changes

State = cooling

$\Delta T < \varepsilon$
Snapshot computation with non validation
Snapshot computation with non validation

12V
1A
6Ω

Schedule → battery
Snapshot computation with non validation

Update battery
Snapshot computation with non validation

```
12V 1A
6Ω
```

```
U 12V U
I 12V I
I 0A
I 2A
I 0A
```

```
Propagate
```

```
Schedule → battery
Update battery

Schedule → fuse
Update fuse

Schedule → resistor
Update resistor

Done
Validate
Reset
EndOfSnapshot
```
Snapshot computation with non validation

1A

12V

6Ω

Schedule → fuse
Snapshot computation with non validation

12V
1A
6Ω

Update fuse
Snapshot computation with non validation

Propagate
Snapshot computation with non validation

12V

1A

6Ω

Schedule → resistor
Snapshot computation with non validation

Update resistor
Snapshot computation with non validation

Propagate
Snapshot computation with non validation

\[ U = 12V \]
\[ I = 1A \]
\[ R = 6\Omega \]

\[ U = 12V \]
\[ I = 0A \]
\[ U = 0V \]
\[ I = 0A \]

\[ U = 12V \]
\[ I = 2A \]
\[ U = 0V \]
\[ I = 0A \]

Schedule → fuse
Snapshot computation with non validation

Update fuse

Poof!
Snapshot computation with non validation

Propagate
Snapshot computation with non validation

12V

1A

6Ω

U

I

12V

0A

2A

2A

0A

12V

12V

12V

12V

Schedule → battery

Poof!
Snapshot computation with non validation

Update battery
Snapshot computation with non validation

\[ U = 12V \]
\[ I = 1A \]
\[ R = 6\Omega \]

\[ U = 12V \]
\[ 0A \]
\[ 2A \]

Poof!

Done

Semantic Adaptation for Models of Computation
Appendix
Snapshot computation with non validation

12V

6Ω

1A

Validate
Snapshot computation with non validation

12V 1A
6Ω

Poof!

Reset
 Snapshot computation with non validation

$12V$ $\rightarrow$ $6\Omega$

$1A$

Schedule $\rightarrow$ battery
Snapshot computation with non validation

12V 1A 6Ω

Update battery
Snapshot computation with non validation

Propagate
Snapshot computation with non validation

Schedule → fuse
Snapshot computation with non validation

Update fuse
Snapshot computation with non validation

Propagate
Snapshot computation with non validation

Schedule $\rightarrow$ battery
Snapshot computation with non validation

```
<table>
<thead>
<tr>
<th>12V</th>
<th>1A</th>
</tr>
</thead>
<tbody>
<tr>
<td>6Ω</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>U</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>12V</td>
<td>12V</td>
</tr>
<tr>
<td>0A</td>
<td>0A</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>0V</td>
</tr>
<tr>
<td>0A</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>U</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>0A</td>
<td>0A</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>0V</td>
</tr>
<tr>
<td>0A</td>
</tr>
</tbody>
</table>
```

Poof!

Update battery
Snapshot computation with non validation

Schedule → resistor
Snapshot computation with non validation

Update resistor
Snapshot computation with non validation

\[ U = 12V \quad I = 1A \quad R = 6\Omega \]

\[ U = 12V \quad I = 0A \]

Poof!

\[ U = 12V \quad I = 2A \]

Schedule $\rightarrow$ fuse
Snapshot computation with non validation

12V 1A
6Ω

Poof!

Update fuse
12V

6Ω

1A

Poof!

U
12V
12V
0A
0A
0V
0A

I

I

I

I

U

U

U

U

0V

0V

0A

0A

Done

Semantic Adaptation for Models of Computation

Appendix

June 24 2011
Snapshot computation with non validation

12V 1A 6Ω

Validate

Schedule → battery
Update battery
Propagate Schedule → fuse
Update fuse Schedule → resistor
Update resistor
Done
Validate

Poof!

U 12V 12V 0V 0V
I 0A 0A 0A 0A

U 12V 12V 0V 0V
I 0A 0A 0A 0A

U 12V 12V 0V 0V
I 0A 0A 0A 0A
Snapshot computation with non validation

12V 1A

6Ω

EndOfSnapshot
Goal

Model causality between simultaneous events
Sequencing of instantaneous actions (≈ synchronous microsteps)

\[ t \in \mathbb{R} \times \mathbb{N} \]
**Goal**

Model causality between simultaneous events
Sequencing of instantaneous actions (≈ synchronous microsteps)

\[ t = \langle t_0, 0 \rangle \]
Goal

Model causality between simultaneous events
Sequencing of instantaneous actions ($\approx$ synchronous microsteps)

\[ t = \langle t_1, 0 \rangle \]

\[ \mathbf{v}_1 = 0 \quad \mathbf{v}_2 = 0 \quad \mathbf{v}_3 = V \]
Goal

Model causality between simultaneous events
Sequencing of instantaneous actions (≈ synchronous microsteps)

\[ \vec{V}_1 = 0 \quad \vec{V}_2 = V \quad \vec{V}_3 = 0 \]

\[ t = \langle t_1, 1 \rangle \]
Goal

Model causality between simultaneous events

Sequencing of instantaneous actions (≈ synchronous microsteps)

\[ \begin{align*}
V_1 &= V \\
V_2 &= 0 \\
V_3 &= 0 \\
t &= \langle t_1, 2 \rangle
\end{align*} \]
Goal

Model causality between simultaneous events
Sequencing of instantaneous actions ($\approx$ synchronous microsteps)

\[
t = \langle t_2, 0 \rangle
\]