A General Framework for Formalizing UML with Formal Languages

William E. McUmber and Betty H. C. Cheng
in Proc. Of ICSE 2001

Mahsa Hasani Sadi
Department of Computer Science
University of Toronto
mhsadi@cs.toronto.edu

CS2125: Modeling Methods Tools and Techniques, Fall 2012
Instructor: Marsha Chechik
What is the problem?

A bunch of UML diagrams are developed just for the purpose of documentation!!!

Need to make models *useful*
How to address the problem?

- **Need to make models **useful**
  - Models are means of abstraction and therefore an appropriate way to deal with complexity

- **Making models executable**
  - Analyzing models
Solution Context Feasibility Analysis

- Embedded systems / Critical systems
- Ad hoc software development

- Flourishing industrial application
- Funding

✖ Development of small to medium size usual software applications
Analyzing Models

Software Design

Software behavior simulation and verification

Hardware design
Hardware behavior simulation
Hardware behavior verification

5 of 25
Behavior Simulation and Verification

- Hardware input – output behavior verification

- Model checking

  - Specification contains safety requirements such as the absence of deadlocks and similar critical states that can cause the system to crash.
Overall Idea (1)

Promela (Protocol / Process Meta Language)

Is input of

SPIN (Simple Promela Interpreter)

Is an instance of

Translated to / Transformed into

UML Models

Requires

Automated Model Checker

model

specification

model checking

not satisfied

satisfied

error trace

OK
Overall Idea (2)

UML Models

Translate to / Transform into

Promela (Protocol / Process Meta Language)

Class Diagram

State Diagram

Terminal A

Terminal B
The proposed framework (1)

- **Metamodel:**
  - Defines the language and processes based on which a model is built
  - Identifies the entities of a model and the relationships between them.
Metamodelling

- Using class diagram to define metamodel
The proposed framework (2)

Formalized UML Metaodel Class Model → UML Metaodel Class Model → UML Metaodel → UML Models → Formalized Promela Metaodel Class Model → Promela Metaodel Class Model → Promela Metamodel → Promela (Protocol / Process Meta Language)
Metamodel Formalization

- Domain / Entities / Relationships [first-order logic]

D1
- e1
- e2
- e3
- e4
- e5
- e6
- r(e1, e2)
  - ......
The proposed framework (3)
The proposed framework (4)

- The property of mapping function $h$:
  - $h$ is **homomorph**ic: It preserves the structural relationships of the source metamodel.
The proposed framework (5)

The homomorphic property of h:

D1
- e1
- e2
- e3
- e4
- r (e1, e2)
- ...
- r (e1, e4)

h

D2
- e’1
- e’2
- e’3
- e’4
- r’ (e’1, e’2)
- ........
- r’ (e’2, e’4)
The proposed framework (6)

Q: Why \( h \) must be homomorphic?

- It preserves semantics
The overview of the proposed framework (1)

Problem

Solution

UML Models

Translate to / Transform into

Executable / Simulation Models

Metamodels

Formalization of Metamodels

Definition of a homomorphic mapping between formalizations

Models

Models

Metamodels

Metamodels
The overview of the proposed framework (2)

Output

UML Meta Model → A set of **rules** for transforming → Executable / Simulation Metamodel

(S, T, h, R)
An Issue about the Proposed Framework

- UML is in a higher level of abstraction than the executable / simulation target language
  - Define patterns of target language statements for each concept in the UML metamodel
Promela (Protocol / Process Meta Language)

- A process modeling language to verify the logic of **parallel systems**
  - Processes,
  - Message channels
  - and variables

- SPIN (Simple Promela Interpreter)
  - Verifies by simulating the system's execution
  - Generates a C program that performs a fast exhaustive verification of the system state space
  - Checks for the absence of deadlocks, unspecified receptions, and unexecutable codes
Case Study (1)

- UML state diagram metamodel
Case Study (2)

- Promela / SPIN metamodel
Case Study (3)

- Mapping of UML metamodel entities to Promela metamodel

<table>
<thead>
<tr>
<th>UML Metamodel</th>
<th>Promela Metamodel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Model</td>
</tr>
<tr>
<td>Class</td>
<td>Object-Protoype</td>
</tr>
<tr>
<td>Relationships</td>
<td>Relationships</td>
</tr>
<tr>
<td>State Vertex</td>
<td>State Vertex</td>
</tr>
<tr>
<td>Transition</td>
<td>Transition</td>
</tr>
<tr>
<td>Pseudostate</td>
<td>Pseudostate</td>
</tr>
<tr>
<td>State</td>
<td>State</td>
</tr>
<tr>
<td>CompositeState</td>
<td>Protoype</td>
</tr>
<tr>
<td>SimpleState</td>
<td>State Block</td>
</tr>
<tr>
<td>ActionSequence</td>
<td>ActionSequence</td>
</tr>
</tbody>
</table>
The rule for mapping UML simple states to Promela specifications

```
1  state-name:
2     H_composite-state-name = st_statename;
3     <entry actions>
4     evt!<event-name 1>,.pid;
5     evt!<event-name 2>,.pid;
6     .
7     .
8     .
9     evt!<event-name n>,.pid;
10    if
11     ::= <transition event expression 1>
12        -> <guard list 1>
13        <action list 1>
14        <send list 1>
15     <exit actions 1>
16     goto nextstate
17     ::= <transition event expression 2>
18        -> <guard list 2>
19        <action list 2>
20        <send list 2>
21     <exit actions 2>
22     goto nextstate
```
Summary

- Development of a framework for formalizing the semantics of a set of UML diagrams

- Development of a prototype tool (Hydra) for transforming UML diagrams to Promela statements