Towards xMOF: Executable DSMLs based on fUML

www.modelexecution.org

Tanja Mayerhofer, Philip Langer, Manuel Wimmer

Business Informatics Group
Institute of Software Technology and Interactive Systems
Vienna University of Technology
Favoritenstraße 9-11/188-3, 1040 Vienna, Austria
phone: +43 (1) 58801-18804 (secretary), fax: +43 (1) 58801-18896
office@big.tuwien.ac.at, www.big.tuwien.ac.at
Domain-Specific Modeling Languages

- Success of MDE depends on availability of means for defining DSMLs

- No standard means for specifying behavioral semantics of DSML exist

⇒ Efficient development of model execution facilities impossible
Semantics of Domain-Specific Modeling Languages

- **Denotational / translational semantics**

  *Examples:*
  - *Abstract State Machines*
  - *Maude*

  *Pros:*
  - Execution and analysis tools can be reused

  *Cons:*
  - Mapping model into target language is complex
  - Results have to be mapped back
Semantics of Domain-Specific Modeling Languages

- **Operational semantics**

  *Approaches:*

  - *Graph transformations*

  - *Action language*
    Kermetta, MXF, Smalltalk, Eiffel, xCore, Epsilon Object Language

  ➔ **Use fUML as action language**
Foundational UML (fUML)

- **OMG standard** V1.0 released 02/2011

- Specification of behavioral semantics of **foundational UML subset**
  - **Structural kernel**: class, association, data type, etc.
  - **Behavioral kernel**: behavior, event, signal, etc.
  - **Activities**: activities, parameters, nodes, flows
  - **Actions**: communication, object, structural feature, link actions

- **Operational semantics** approach specifying a fUML virtual machine

  ➔ **UML activity diagrams can be executed**
Specifying Semantics with fUML

- **Level mismatch** for specifying semantics of DSML using fUML activities
### Specifying Semantics with fUML

<table>
<thead>
<tr>
<th>Current Situation</th>
<th>(1) Push down DSML to M1 / Pull up DSML to M2</th>
</tr>
</thead>
<tbody>
<tr>
<td>M3</td>
<td>![Diagram of MOF]</td>
</tr>
<tr>
<td>M2</td>
<td>![Diagram of fUML and aDSML MM]</td>
</tr>
<tr>
<td>M1</td>
<td>![Diagram of afUML Model and aDSML Models]</td>
</tr>
</tbody>
</table>

#### Pros
- Approach can be implemented using existing tools

#### Cons
- High effort for transformation needed
- UML environment has to be used instead of metamodeling environments

---

```
MM  ... Meta-Model
OS  ... Operational Semantics
AD  ... Activity Diagram
CD  ... Class Diagram
OD  ... Object Diagram
```
Specifying Semantics with fUML

- Integrated metamodeling language **eXecutable MOF (xMOF)**
  - Abstract syntax: MOF
  - Behavioral semantics: fUML
Example: Petri Net

- Provides runtime representation
- Separation of syntax and semantics
Example: Petri Net

**Syntax Metamodel**

- **Net**
  - `conf`: integer
  - `places`: Place
  - `transitions`: Transition

- **Place**
  - `initialTokens`: integer
  - `input`: Place
  - `output`: Place

- **Transition**
  - `input`: Place
  - `output`: Place

- **PlaceConf**
  - `tokens`: integer
  - `addToken`: void
  - `removeToken`: void

- **TransitionConf**
  - `fire`: void
  - `isEnabled`: boolean

- **NetConf**
  - `run`: void

**Runtime Configuration Metamodel**

- **n**: NetConf
  - `place configs`:
    - **p1**: PlaceConf
      - `tokens = 1`
    - **p2**: PlaceConf
      - `tokens = 0`
  - `transition configs`:
    - **t1**: TransitionConf
      - `input`
      - `output`
Example: Petri Net

```java
package NetConf;

public class NetConf {
    public void run()
    {
        List<TransitionConf> transitionConfigs = readTransitionConfigs();
        transitionConfigs = filterEnabledTransitions(transitionConfigs);
        if (transitionConfigs.size() > 0)
        {
            List<TransitionConf> targetTransitions = selectEnabledTransitions(transitionConfigs);
            transitionConfigs.forEach((transition) -> transition.fire());
        }
    }

    private List<TransitionConf> readTransitionConfigs()
    {
        List<TransitionConf> transitionConfigs = new ArrayList<>();
        // Read transition configurations
        return transitionConfigs;
    }

    private List<TransitionConf> filterEnabledTransitions(List<TransitionConf> transitionConfigs)
    {
        List<TransitionConf> enabledTransitions = new ArrayList<>();
        // Filter enabled transition configurations
        return enabledTransitions;
    }

    private List<TransitionConf> selectEnabledTransitions(List<TransitionConf> transitionConfigs)
    {
        List<TransitionConf> selectedTransitions = new ArrayList<>();
        // Select enabled transition configurations
        return selectedTransitions;
    }

    private void fire(TransitionConf transition)
    {
        // Fire the transition
    }
}
```
Example: Petri Net

```
transitionconfigs
input output
0 1

PlaceConf::addToken()
result : PlaceConf[*]

PlaceConf::removeToken()
result : PlaceConf[*]
```

```
ReadSelf
Read

ReadStructuralFeature
Read output

result : TransitionConf
object : TransitionConf

ReadStructuralFeature
Read input

result : PlaceConf[*]
object : TransitionConf
```
Extensibility of Semantics

- Model everything down to the very last detail may not be feasible
- May require **utilization of libraries** which are not available for the fUML virtual machine

*Examples:*
- Complex mathematical calculations
- Control of external resources

**Integration of external libraries with fUML virtual machine**
- No extension of fUML metamodel and virtual machine
- Transparent usage of external libraries
Example: Petri Net

```
NetConf::run()

ReadSelf
Read
result : NetConf

ReadStructuralFeature
Read transition configs
object : NetConf
result : TransitionConf[*]

parallel

Select enabled transition configs
transition configs : TransitionConf[*]

Call isEnabled()
(TransitionConf::isEnabled)
result : boolean
target : TransitionConf

decisionInputFlow
[true]

transition configs : TransitionConf[*]

{t1, t2, t3}
list : TransitionConf[*]

Get : ListGet
index : int
{t1}
target : TransitionConf

Call fire()
(TransitionConf::fire)
result : TransitionConf

ValueSpecification
Specify 1

Random
+ next(int)
+ nextInt()
+ nextInt(int)
...```
Extensibility of Semantics

1. Import of external libraries
   1. **Reverse engineering** of library for extracting API classes
   2. **Import of classes** into fUML model specifying the semantics of a DSML
   3. Create empty **activities** for each operation acting as **place holder**

2. Integration of external libraries at runtime
   
   **Call of library operation**
   1. **Suspend** execution at entry of place holder activity
   2. **Forward invocation** to actual operation of external library
   3. **Integrate result** into runtime model of execution

   **Instantiation / Modification of library instances**
   1. Maintain mapping between fUML instances and library instances
Conclusion

- Integrated metamodeling language **eXecutable MOF (xMOF)**
- Specification of **behavioral semantics** using **fUML**
- Usage of **external libraries**
Future Work

- **Implementation** of xMOF
  - Provide means for using fUML as semantics specification language
  - Conduct *case study* to show feasibility of our approach

- **(Semi-)Automatic generation** of model execution facilities
  - **Analysis** of execution: trace model
  - Runtime *observation* and *control*: event model, command API

- **Reusability** of semantics specifications
  - Definition of *kernel semantics*, e.g. data flow, control flow, signal sending
  - **Composition** of *kernel semantics* for semantics specification of DSML
  - **Specialization** of existing semantics specification for variation point / profile
Thank you!

Debugging and Testing Models Based on fUML

www.modelexecution.org