Simulation and Analysis of Domain Specific Languages

José E. Rivera, Francisco Durán, Antonio Vallecillo
Universidad de Málaga

Complexity (essential)

[Borrowed from Dov Dori’s Tutorial on SysML Modeling at TOOLS 2008]
How do we deal with that?

- We need to break this Gordian Knot
  - We should not center in PROGRAMMING
  - We need to raise the level of abstraction
- What happens in the rest of the Engineering disciplines?
  - Civil Engineering
  - Civil Architecture
  - Avionics and aerospace
  - ...
Traditional Engineers use “Models”

- Specify the system
  - Structure, behaviour, ...
  - Separate concepts at different conceptual levels
  - Communicate with stakeholders
- Understand the system
  - If existing (legacy applications)
- Validate the model
  - Detect errors and omissions in design
    - Mistakes are cheaper at this stage
  - Prototype the system (*execution* of the model)
  - Formal analysis of system properties
- Drive implementation
  - Code skeleton and templates, complete programs (?)

[Borrowed from Bran Selic Tutorial at JISBD 2008]

Model Characteristics [Selic, 2003]

- **Abstract**
  - Emphasize certain aspects…
  - And hide others
- **Understandable**
  - Expressed in a language that can be understood by users and *stakeholders*
- **Precise**
  - Faithful representations of the system being modeled
- **Predictive**
  - To infer correct conclusions
- **Cheap**
  - Easier and cheaper to build and analyze than the whole system
What is a Model?

- A **description** of (part of) a system written in a well-defined language. (= *specification.*) [Kleppe, 2003]
- A **representation** of a part of the function, structure and/or behavior of a system [MDA, 2001]
- A **description** or **specification** of the system and its environment for some certain *purpose.*
  A model is often presented as a combination of drawings and text. [MDA Guide, 2003]
- A **set of statements** about the system. [Seidewitz, 2003]
  *(Statement: expression about the system that can be true or false.)*
- M is a model of S if M can be used **to answer questions** about S [D.T. Ross and M. Minsky, 1960]

What is a Metamodel?

- A model of a well-defined language [Kлеппе, 2003]
- A model of models [MDA, 2001]
- A model that defines the language for expressing a model [MOF, 2000]
  - A **meta-metamodel** is a model that defines the language for expressing a metamodel. The relationship between a meta-metamodel and a metamodel is analogous to the relationship between a metamodel and a model.
- A model of a modelling language [Seidewitz, 2003]
  - That is, a metamodel makes statements about what can be expressed in the valid models of a certain modelling language.
OMG’s four-layers metamodel hierarchy

Level M^3
- the MOF

Level M^2
- the SPEM
- the UML
- the CWM

Level M^1
- a UML model m
- another UML model m'

Level M^0
- a particular use of m
- another use of m

MOF Metamodel (simplified)

ModelElement
- name: String

Package
- Classifier
  - Type
- Feature
- DataType
- Class
- Attribute
- AssociationEnd
  - role
  - 1 association
  - Association
Domain Specific Languages (DSL)

- Languages for representing different views of a system in terms of models
- Higher-level abstraction than general purpose languages
- Closer to the problem domain than to the implementation domain
- Closer to the domain experts, allowing modelers to perceive themselves as working directly with domain concepts
- Domain rules can be included into the language as constraints, disallowing the specification of illegal or incorrect models

DSLs

- DSLs are defined in terms of
  - Abstract syntax (domain concepts and rules)
  - Concrete syntax (language representation)
- Metamodels used to represent the abstract syntax
  - Models “conform to” metamodels
- Metamodels are models, too
  - A metamodel conforms to its meta-metamodel
- This tower usually ends at level 4
An example of a DSL

Domain Specific Modeling

» Several notations for Domain Specific Modeling (DSM) already available
  » Abstract and concrete syntaxes for the definition of models, metamodels and their representations
  » Enable the rapid and inexpensive development of DSLs and associated tools (e.g., model editors)

» Repositories of metamodels and model transformations already in place
  » Eclipse/GMT/AM3 project
  » MDWEnet initiative
  » …
KM3

- Specialized textual language for specifying metamodels
  - Abstract syntax based on Ecore and MOF 2.0
    - Notions of package, class, attribute, reference, data type
  - Simple and easy to work with
  - Possible conversions to/from MOF, Ecore
  - Good tool support
  - Integrated with MDD development environments (AMMA)
  - Growing interest and adoption

Very Simple State Machine

```java
package VerySimpleStateMachine {

class StateMachine {
    reference initialState [0-1]: State;
    reference containedState [*] container : State
       oppositeOf stateMachine;
}
class State {
    attribute name : String;
    reference stateMachine : StateMachine
       oppositeOf containedState;
    reference incoming [*] : Transition oppositeOf target;
    reference outgoing [*] : Transition oppositeOf src;
}
class Transition {
    attribute name : String;
    reference target : State oppositeOf incoming;
    reference src : State oppositeOf outgoing;
}
```
What is in a metamodel?

- A metamodel describes
  - the concepts of the language,
  - the relationships between them, and
  - the structuring rules that constrain the model elements and combinations in order to respect the domain rules.

Is that all?

- These descriptions only capture the "static" specification of the language...
  - It is not clear from the metamodel what happens if an event occurs and there is no transition that can be triggered.
    - Is the event lost, or is it held until the state machine reaches a state with a transition that can be triggered by the event?
    - What is the behavior of the system when it contains internal transitions? How do they exactly behave?

- [Robin Milner]: "A (meta)model consists of some concepts, and a description of permissible activity in terms of these concepts."

- [Chen et al]: Metamodel "semantics"
  - Structural semantics: describe the meaning of models in terms of the structure of model instances: all of the possible sets of components and their relationships, which are consistent with the well-formedness rules
  - Behavioral semantics: describe the evolution of the state of the modeled artifacts along some time model.
An example of a (more useful) DSL

http://www.youtube.com/watch?v=NZNTgglPbUA

MDE is more than Conceptual Modeling!

- **Current DSLs**
  - Toy-ish
  - Unanimated (mostly static)
  - Limited analysis capabilities
- **Several notations proposed for DSM**
  ...but formal and tool support is quite limited:
  - Most efforts focused on definition of models, metamodels and transformations between them
  - Other operations (e.g., model subtyping, difference, versioning) are also needed in industrial MDD practices
- **Almost inexistent tool support for**
  - Simulation, Analysis, Estimation, Quality evaluation and control, ...
- **Almost inexistent proven engineering methodologies**
  - For neither development nor modernization
We need to be able (at least) to:

- Deal with both the accidental and the essential complexity of complex systems
  - Use separate viewpoints to specify systems (each viewpoint uses its corresponding DSL)
  - Check the consistency of multi-viewpoint specifications
- Animate models
  - Explicitly define behavioral semantics of DSLs so that models can be understood, manipulated and maintained by both users and machines
  - Define different semantics (separate concerns)
- Analyze models
  - Add Non-Functional Properties to DSLs
  - Connect DSLs to Analysis tools

Definition of a DSL
How to implement the Mappings?

- As Model Transformations!!!

Types
- Domestic
- Horizontal
- Vertical
- Abstracting
- Refining
- Pruning
- Forgetful
- ...

Our approach

- Maudeling
  - Use of Maude as underlying platform (logic)
    - Semantic Mappings from EMF
  - Model Management
    - Model Difference
    - Model subtyping
    - Type Inference, Evaluating model metrics...

- Model Simulation and Analysis
  - e-Motions
    - Specification of the dynamic behavior of DSLs
  - Semantic Mappings from EMF, Graph Transformations to Maude
    - Simulation
    - Reachability Analysis
    - Model Checking
A Production System Example

Model Simulation and Analysis

- Specification of the dynamic behavior of a DSL
- Simulation
- Reachability Analysis
- Model Checking
Specifying dynamic behavior (with e-Motions)

- Use In-place Transformation Rules (e.g., graph transformations)
- Completely Independent from the underlying semantic framework (e.g., Maude)

Adding Time to Behavioral Specifications

- Part of the e-Motions modeling notation
- Rule duration
- Access to the Global Time Elapse
  - Time stamps, scheduled actions
Adding Action Executions to Behavioral Specs

- Specification of action properties
- Without the need of unnaturally modifying the metamodel

```
\[ \text{PickUp} \]
\begin{array}{ll}
\text{Init:} & \text{p} = \text{false} \\
\text{Action:} & \text{p} = \text{true} \\
\text{End:} & \text{p} = \text{false}
\end{array}
```

```
\[ \text{Collect} \]
\begin{array}{ll}
\text{Init:} & \text{w} = \text{false} \\
\text{Action:} & \text{w} = \text{false} \\
\text{End:} & \text{w} = \text{false}
\end{array}
```

```
\[ \text{Move} \]
\begin{array}{ll}
\text{Init:} & \text{a} = \text{false} \\
\text{Action:} & \text{a} = \text{false} \\
\text{End:} & \text{a} = \text{false}
\end{array}
```

```
\[ \text{PutDown} \]
\begin{array}{ll}
\text{Init:} & \text{y} = \text{false} \\
\text{Action:} & \text{y} = \text{false} \\
\text{End:} & \text{y} = \text{false}
\end{array}
```

Bridges between Semantic Domains

- Precise semantics
- A set of Analysis Tools
- Underlying logic
Semantic Mappings

Mutually compatible models can be extracted and injected with ATL Transformations.

Representing Models with Maude

ProductionSystem {
  < 'p : Plant | els : 'heg 'hag 'c1 'c2 't1 'a 'c3 't2 'u >
  < 'hag : HandleGen | in : null, out : 'c2, xPos : 1, yPos : 1 >
  < 'heg : HeadGen | in : null, out : 'c1, xPos : 1, yPos : 3 >
  < 'c1 : Conveyor | parts : nil, out : 't1, xPos : 2, yPos : 3 >
  < 'c2 : Conveyor | parts : nil, out : 't1, xPos : 2, yPos : 1 >
  < 't1 : Tray | parts : nil, capacity : 4, xPos : 3, yPos : 2 >
  < 'a : Assembler | in : 't1, out : 'c3, xPos : 4, yPos : 2 >
  < 'c3 : Conveyor | parts : nil, out : 't2, xPos : 5, yPos : 2 >
  < 't2 : Tray | parts : nil, capacity : 4, xPos : 6, yPos : 2 >
  < 'u : User | parts : nil, xPos : 6, yPos : 3 >
}
Representing Metamodels with Maude

```
op ProductionSystem : -> @Metamodel.
op PS : -> @Package.
sort PositionedEl.
subsort PositionedEl < @Class.
op PositionedEl : -> PositionedEl.
op xPos : -> @Attribute.
op yPos : -> @Attribute.
sort Container.
subsort Container < PositionedEl.
op Container : -> Container.
op parts : -> @Reference.
sort Machine.
subsort Machine < PositionedEl.
op in : -> @Reference.
op out : -> @Reference.

eq isAbstract(Machine) = true.
...
... type(in) = Tray.
eq lowerBound(in) = 0.
eq upperBound(in) = 1.
...
... type(out) = Conveyor.
eq opposite(out) = null.
eq lowerBound(out) = 1.
eq upperBound(out) = 1.
```

Representing Behavior with Maude

```
rl [Carry] :
ProductionSystem{
  < p : P:Part | xPos : XPOS, yPos : YPOS, SFS >
  < c : Conveyor | parts : (p PARTS), out : t, SFS' >
  < t : Tray | xPos : XPOS', yPos : YPOS', parts : PARTS', SFS'' > OBJSET}
=>
ProductionSystem{
  < p : P:Part | xPos : XPOS', yPos : YPOS', SFS >
  < c : Conveyor | parts : PARTS', out : t, SFS' >
  < t : Tray | xPos : XPOS', yPos : YPOS', parts : (p PARTS'), SFS'' > OBJSET }
```

...
Time and Action Executions in Maude

- Real-Time Maude used to provide semantics to E-Motions
  
- System behavior: instantaneous transitions
- Time elapse: Tick rule
  - Defined over clocks and Action Executions

Model Simulation and Analysis with Maudeling

- Simulation/Execution of specifications
  - (rew initModel in time <= 20.)

- Reachability Analysis
  - Deadlock
  - Invariants
  - Others

- LTL Model checking
  - Liveness properties
    - (mc (initModel) \=t
      [](ensimbl('he10.ha10') -> collected('he10.ha10'))
      in time <= 100.)
Model Management

- Model Difference
- Model subtyping
- Type Inference,
- Model metrics…

MAUDELIN

Model difference: Comparison process

- **Matching**
  - Finding different objects from both models that represent the same element
  - Model as a result :
  - Persistent identifiers VS structural similarities

- **Differencing:**
  - Makes use of matching models to detect modified elements
  - Model as a result:
    - Self-contained
    - Compact
    - Independent of the metamodel of the source models
Model Difference: An Example

(Subtrahend Model)
- SM : StateMachine | initState : ST1, containedStates : (ST1, ST2)
- TR : Transition | name : "TR", src : ST1, target : ST2

(Minuend Model)
- SM : StateMachine | initState : ST1, containedStates : (ST1, ST2)
- TR : Transition | name : "TR", src : ST1, target : ST2
- TR2 : Transition | name : "TR2", src : ST2, target : ST1

(Difference Model)
- ST1@MOD : ModifiedElement | element : ST1@NEW, oldElement : ST1@OLD
- ST1@NEW : State | incoming : TR2
- ST1@OLD : State | incoming : empty
- ST2@MOD : ModifiedElement | element : ST2@NEW, oldElement : ST2@OLD
- ST2@NEW : State | outgoing : TR
- ST2@OLD : State | outgoing : empty
- TR2@ADD : AddedElement | element : TR2@NEW
- TR2@NEW : Transition | name : "TR2", src : ST2, target : ST1

Difference related operations

- **Operation do**
  - do(Ms, Md) = Mm
  - Applies to a model all the changes specified in a difference model

- **Operation undo**
  - undo(Mm, Md) = Ms.
  - Reverts all the changes specified in a difference model
  
  \[
  \text{undo(do(Ms, Md), Md)} = Mm \\
  \text{do(undo(Mm, Md), Md)} = Mm.
  \]

- **Sequential composition of differences**
  - “Optimize” the process of applying successive modifications to the same model
Model subtyping

- **Model type**
  - Essentially its metamodel

- **Model subtyping**
  - Model operations reuse
  - Type safety
  - Polymorphism in MDSD
  - *Model bus, metamodel matchmaking*, metamodel evolution

> Metamodels \( M', M \): \( M' \leq M \leftrightarrow \)

\[
\forall K \in \{ M\text{.package} \} \exists K' \in \{ M'\text{.package} \} \bullet (K' \leq K)
\]

> Packages \( K', K \): \( K' \leq K \leftrightarrow \)

\[
isRelated(K\text{.name}, K\text{.name}) \land \\
\forall C \in \{ K\text{.class} \} \exists C' \in \{ K'\text{.class} \} \bullet (C' \leq C)
\]

> Classes \( C', C \): \( C' \leq C \leftrightarrow \)

\[
isRelated(C\text{.name}, C\text{.name}) \land \\
(C'\text{.isAbstract} \rightarrow C\text{.isAbstract}) \land \\
\forall C \in \{ C\text{.superTypes} \} \exists C' \in \{ C'\text{.superTypes} \} \bullet (C' \leq C) \land \\
\forall S \in \{ C\text{.structuralFeatures} \} \exists S' \in \{ C'\text{.structuralFeatures} \} \bullet (S' \leq S)
\]

> Attributes \( P', P \): \( P' \leq P \leftrightarrow \)

\[
isRelated(P\text{.name}, P\text{.name}) \land (P'\text{.type} \leq P\text{.type}) \land \\
(P'\text{.isUnique} = P\text{.isUnique}) \land (P\text{.lower} \leq P'\text{.lower}) \land (P'\text{.isOrdered} = P\text{.isOrdered}) \land \\
((P\text{.upper} = P'\text{.upper}) \lor (2 \leq P\text{.upper} \leq P'\text{.upper}))
\]

> References \( R', R \): \( R' \leq R \leftrightarrow \)

\[
isRelated(R\text{.name}, R\text{.name}) \land (R'\text{.type} \leq R\text{.type}) \land \\
(R'\text{.isUnique} = R\text{.isUnique}) \land (R\text{.lower} \leq R'\text{.lower}) \land (R'\text{.isOrdered} = R\text{.isOrdered}) \land \\
((R\text{.upper} = R'\text{.upper}) \lor (2 \leq R\text{.upper} \leq R'\text{.upper})) \land (R'\text{.opposite} \leq R\text{.opposite})
\]
mOdCL: our (Maude–based) OCL tool

- **mOdCL is a tool to**
  - Evaluate OCL expressions in general
  - Validate OCL constrains on UML models

- **Static and dynamic validation**
  - Static validation of system states
  - Dynamic validation (using execution strategies)

- **mOdCL can be used as back-end**
  - From Maude based tools requiring OCL expressions evaluation

\[
\text{eval(OCL-expr, state)}
\]

mOdCL

- **mOdCL to validate UML models**
  - System states represented as objects configurations
  - System behavior representation
    - Some rules regarding operation calls representation
    - The rest of the system can be represented according to the user preferences

- **Future tools on mOdCL**
  - System behavior skeleton generator
  - UML model \(\rightarrow\) mOdCL using model transformation tools (ATL)
Conclusions

- Formal support for MDE is required
  - For building tools
  - To explicitly and completely describe behavior
  - To disambiguate semantic variation points
- We make use of Maude:
  - To specify models, metamodels and their behavior
  - To make use of its analysis tools
  - To provide formal semantics to other visual approaches (based on Eclipse, Graph grammars, ...)
- Current Tools: Maudeling, E-Motions, mOdCL
- Future work:
  - Including further related time aspects: periodicity ...
  - Specification of non-functional properties of DSLs
  - Inverse transformations for analysis results
  - Connection with more analysis tools

Use of models to connect the tools

- Optimization Templates
- Simulation Building Blocks
- Simulation Templates of Diverse Behavior & Fidelity
- Evacuation Mgt.
- Propeller Hydrodynamics
- Damaged Stability
- Navigation Accuracy
- Augmented Descriptive Models
- System Description Tools & Resources
  - ECAD & MCAD Tools
    - Tribon, CATIA, NX, Cadence, ...
  - Systems & Software Tools
    - DOORS, Studio, MagicDraw, Eclipse, ...
  - Operation Mgt. Systems
  - Libraries & Databases
    - Classification Codes, Materials, Personnel, Procedures, ...
- Simulation Solvers
  - Evacuation Codes
    - Egress, Exodus, ...
  - General Math
    - Mathematica, Maple, Matlab ...
  - FEA
    - Abaqus, Ansys, Nastran, ...
  - Discrete Event
    - Arena, Quest, ...

(Legend: Tool Association)
Basic References and Resources

- MDD, MDE, MDA, DSM
  - [http://planet-mde.org](http://planet-mde.org)
  - [http://www.omg.org/mda](http://www.omg.org/mda)
  - [http://www.eclipse.org/gmt/](http://www.eclipse.org/gmt/)
  - [http://www.visualmodeling.com/DSM.htm](http://www.visualmodeling.com/DSM.htm)
  - [http://www.dsmforum.org](http://www.dsmforum.org)
  - [http://www.sysmlforum.com](http://www.sysmlforum.com)

- Atenea Tools

---

Simulation and Analysis of Domain Specific Languages

José E. Rivera, Francisco Durán, Antonio Vallecillo
Universidad de Málaga

---

Thanks!