Metamodeling

Janos Sztipanovits
ISIS, Vanderbilt University
janos.sztipanovits@vanderbilt.edu
• Overview of Metamodeling
• Abstract Syntax Metamodeling
  Concepts
• Metamodeling languages
  - UML Class Diagrams and OCL
  - Meta-GME
  - More Stuff on Metamodelling
• Standardization Trends
  - ECLIPSE
  - OMG
Domain Specific Modeling Languages

\[ L = \langle C, A, S, M_S, M_C \rangle \]
Concrete Syntax

- Notation for constructing and presenting models
  - Textual syntax: uses structured text to represent models. Common approach in modeling languages is to use XML as a textual syntax.

```xml
  + <State id="ON" _id="ida18" active="false" master="false" initial="false" initialState="ZERO">
  + <State id="ZERO" _id="ida74" active="false" master="ON" initial="false" initialState=""/>
  + <State id="ONE" _id="ida75" active="false" master="ON" initial="false" initialState=""/>
  <Transition id="T11" _id="idade" dst="ONE" src="ZERO" guard="true" preemptive="false" outputEvent=""
  triggerEvent='LocalEvent.one'/>
```

- Visual syntax: represents the program diagrammatically.
Abstract Syntax

• Describes:
  - the concepts of the modeling language
  - relationships that may exists among the concepts
  - well-formedness rules specifying how the concepts can be combined

• Abstract syntax defines the set of syntactically correct models

• Abstract syntax is decoupled from concrete syntax and semantics

• ...but how to represent it?!
Semantics

- Semantics describes meaning of the model - usually in terms of a mathematical domain. This mathematical domain is called “semantic domain”.
- The same model can be associated with different semantic domains.
- Mapping the concepts and relations of the abstract syntax to the semantic domain is part of the specification of the modeling language.
Metamodelling

• The metamodel is the model of the modeling language.

• Complete specification of a modeling language includes all elements of

\[ L = \langle C, A, S, M_S, M_C \rangle \]

As a first step, we focus on \( C, A, M_C \).

• Most DSML specifications are limited to the abstract syntax and concrete syntax metamodelling, we will discuss the metamodelling of semantics later.
A metamodel is defined in a metamodeling language.

The metamodeling language is designed to model concepts, relations and well-formedness rules.

We need a type-language with constraints and set-valued semantics.
Selection of a Metamodelling Language

- UML Class Diagram (UML-CD) and Object Constraint Language (OCL)
- Meta Object Facility (MOF)
- Several other alternatives but with the same set-valued semantics.
Why Do We Need Metamodel Architecture?

• Models written in a language are instances of the concepts defined in the metamodel; metamodels written in a metamodeling language are instances of concepts defined in the meta-metamodel.
**M0:** Data in a model (e.g. parameters in a look-up table)

**M1:** Model (e.g. model of a non-linear controller)

**M2:** Metamodel of the modeling language (e.g. Look-up Table Class)

**M3:** The properties of all metamodels (e.g. properties of Classes)
Content

• Overview of Metamodelling
• Abstract Syntax Metamodelling
  - Concepts
• Metamodelling languages
  - UML Class Diagrams and OCL
  - Meta-GME
  - More Stuff on Metamodelling
• Standardization Trends
  - ECLIPSE
  - OMG
What is UML?

• The Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems.

• Main Goals of UML
  - Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
  - Provide extensibility and specialization mechanisms to extend the core concepts.
  - Be independent of particular programming languages and development processes.
  - Support higher-level development concepts such as collaborations, frameworks, patterns and components.
The core package is a complete metamodel. A metamodel is a model that defines the language for expressing a model. UML, CWM, MOF and Profile each depends on a common core. The profiles package defines the mechanism used to tailor existing metamodels towards specific platform, such as C++, CORBA, or EJB.
UML 2.0 Infrastructure Architecture: Four-Layer Metamodeling Hierarchy

- **M3: Meta-metamodelling Layer**
  - Such as MOF

- **M2: Metamodelling Layer**
  - Such as UML and CWM

- **M1: Modeling Layer**
  - Such as Class Diagrams

- **M0: Run-time Instances of Model Elements**

Sztipanovits
UML Diagrams

- Static Structure Diagram
  - Class Diagram
  - Package Diagram
  - Object Diagram

- Behavior Diagram
  - Activity Diagram
  - Interaction Diagram
  - State Machine Diagram
  - Use Case Diagram

- Implementation Diagram
  - Component Diagram
  - Deployment Diagram

Can be used for Metamodelling

Used for general modeling
UML Class Diagram

• Class Diagram
  - A diagram that shows a collection of static model elements, such as classes, types, and their contents and relationships.

• Main Modeling Constructs in Class Diagrams
  - Classes and their structure and operations
  - Association, aggregation, composition and generalization relationships
  - Multiplicity and navigation indicators
  - Role names
Classes

• Classifier
  - A collection of instances that have something in common.
• Class
  - A classifier that describes a set of objects that share the same specifications of features, constraints, and semantics.
  - Instances of classes are called objects.
• Attribute
  - A structural feature of a classifier that characterizes instances of the classifier.
• Operation
  - A feature which declares a service that can be performed by instances of the classifier of which they are instances.
• Value
  - An element of a type domain.
Associations

• An association specifies a semantic relationship that may occur between instances of classifiers.
• Instances of associations are called links.

\[
\text{Person} \quad \text{WorksFor} \downarrow \\
\text{Organization}
\]

\[
\begin{array}{c|c}
\text{Works-for} \\
\hline
\text{Steve Jobs} & \text{Apple} \\
\text{Bill Gates} & \text{Microsoft} \\
\text{Larry Ellison} & \text{Oracle} \\
\cdots & \cdots
\end{array}
\]
N-ary Associations

- An association among three or more classes.
- The arity of an association is the number of objects that participate in each occurrence of the association.

Sztipanovits
Association: Multiplicity Constraints

- A specification of the range of allowable cardinalities that a set may assume.
- A multiplicity is a (possible infinite) subset of the non-negative integers.
• An association class is a modeling element that has both association and class properties.
  - An association that also has class properties, or a class that also has association properties.

• An association class is depicted by an association symbol (a line) and a class symbol (a box) connected with a dash line.
Aggregation

- Aggregation is a special form of association that specifies a part-whole relationship between the aggregate (whole) and a component part.
- The aggregation relationship is transitive and antisymmetric across all aggregation links.
- A part can belong to more than one aggregate, and it may exist independently of the aggregate.
A composition is a form of aggregation with **strong ownership** and coincident lifetime of parts with whole.

A composition requires that a part instance be included in at most one composite at a time.

The composite object is responsible for the creation and destruction of parts.
Generalization

- A generalization is a taxonomic relationship between a more general classifier and a more specific classifier.
- A specific classifier inherits all properties of the more general classifier.
- Specialization is performed by adding extra properties to a specific classifier.
A constraint is a condition that has to be satisfied by any correct implementation of a design.

The formal constraints can be written in OCL, the Object Constraint Language (developed by IBM).
Object Constraint Language (OCL)

• OCL is a formal language used to describe expressions on UML models.
• Developed as a business modeling language with IBM Insurance division.
• OCL provides a way to develop more precise models using UML.
  - To specify invariant conditions that must be hold for the system being modeled.
  - To specify application-specific constraints in models.
  - To specify queries on the UML model.
  - To specify pre- and post conditions on Operations and Methods.
  - To specify Guards.
Main Features

- **OCL** is a pure specification language and easy to read and write.
- **OCL** expression is guaranteed to have no side effects.
- **OCL** expressions are not by definition directly executable.
- **OCL** is a typed language. Each Classifier defined within a UML model represents a distinct OCL type.
- The evaluations of an OCL expression is instantaneous.

Sztipanovits
Basic Types & Operations

- **Boolean**
  - and, or, xor, not, implies, if-then-else

- **Integer**
  - *, +, -, /, abs()

- **Real**
  - *, +, -, /, floor()

- **String**
  - concat(), size(), substring()
- OCL expressions can refer to Classifiers in UML class diagrams.
  - Such as classes, interfaces, associations and datatypes.
- **Self** is a OCL reserved word to refer to the classifier instance.
- OCL expressions can navigate UML class diagrams.

**Invariants on Attribute:**
```
context Person inv:
    self.age() > 0
```

**Navigation to Association Class:**
```
context Person inv:
    self.employeeRanking[bosses]->sum() > 0
```
• Collection is an abstract type. It has three collection types:
  - Set
    • In a set, each element may occur only once.
  - Bag
    • In a bag, elements may be present more than once.
  - Sequence
    • Sequence is a bag in which elements are ordered.

• OCL allows you to treat any instance like a collection. You can ask if a single attribute is empty.
Collection Operations

• Select Operations
  - **Example:** Specify a invariant that all employee in a company older than 50 years is not empty.

    ```
    context Company inv:
    self.employee->select(age>50)->noEmpty()
    ```

• ForAll Operations

    ```
    context Company inv:
    self.employee->forAll( age <= 65 )
    ```

• Exists Operation

    ```
    context Company inv:
    self.employee->exist(p:Person | p.forname = 'Jack')
    ```
References

Content

- Overview of Metamodelling
- Abstract Syntax Metamodelling
  - Concepts
- Metamodelling languages
  - UML Class Diagrams and OCL
  - Meta-GME
  - More Stuff on Metamodelling
- Standardization Trends
  - ECLIPSE
  - OMG
Metamodeling Architecture

Sztipanovits
GME and MetaGME

- Meta-programmable modeling tool
- Platform for the development of many model-based embedded systems tools
- Includes a metamodeling language with roots which predate MOF: MetaGME

\[ \text{MetaGME} T_{1,GME/Meta} : \text{MetaGME}^{A_{DSML}} \rightarrow \text{GME/Meta}^{A_{DSML}} \]
MetaGME

- Based on UML Class Diagrams and OCL
- Class Stereotypes imply part of the syntax of elements in the metamodel:

  - <<Model>>: Compositional containers
  - <<Atom>>: Primitive objects
  - <<Reference>>: Pointers to model objects
  - <<Set>>: Membership-based groupings
  - <<Connection>>: Relationships with object identity
  - <<Aspect>>: Logical visual partitions of a system

Sztipanovits
• FCOs (First-Class Objects) are classes which must be abstract but can serve as the base type of a class of any other stereotype in an relationship. For example, Atoms may only inherit from other Atoms and FCOs, but a particular FCO might serve as the base type of both Atoms and Models.

• Models are composite classes. An attribute on the Containment connection allows users to specify whether a given component type is graphically exposed by its containing Model type as a port.

• Atoms are elementary, non-composite classes.

• Sets are classes which group other classes together using a special meta-level relationship which captures the semantics of UML non-composite aggregation.

• References are classes that refer (through a special meta-level relationship) to other classes.

• Connections are analogous to UML association classes.

• Aspects are classes which group other classes together (through a special meta-level relationship) into logical visibility partitions to present different views of a model.

Sztipanovits
The GME tool suite uses the same editor for editing metamodels as for editing domain models. This is because the metamodeling environment is just a domain modeling language in the "Metamodeling" domain. In fact, the metamodeling environment was generated from its own metamodel, known as the meta-metamodel.
Now in GME it is possible to “import” metamodels into the metamodeling environment, and combine them using the composition operators. This allows metamodel reuse, facilitated by the fact that the metamodeling environment is a full fledged domain environment. Also, extending the metamodels leaves the originals intact, while the extensions reflect changes to the originals when they take place.
Content

• Overview of Metamodelling
• Abstract Syntax Metamodelling
  - Concepts
• Metamodelling languages
  - UML Class Diagrams and OCL
  - Meta-GME

  More Stuff on Metamodelling

• Standardization Trends
  - ECLIPSE
  - OMG
Changing Them...

What if you hate MetaGME and love MOF?

Metamodelling of DSML Using MOF

\[ DSML \rightarrow_{MOF}^{MM_{DSML}} MOF \rightarrow_{MetaGME}^{MM_{MOF}} MetaGME \]

Metamodelling of MOF Using MetaGME

\[ MOF \rightarrow_{T_1}^{GME-MOF} MetaGME \]

Sztipanovits
Composing Them...

• Composition allows metamodels to contain other metamodels
  - This allows a metamodeler to take advantage of existing metamodels
  - Metamodeling development time is decreased
  - Systems may be independently developed, and later brought together

• Other features of the composable metamodeling environment allow for models describing similar systems to be brought together in a common modeling environment
Composing Them...But How?

- How is composition accomplished?
  - UML containment and inheritance are used to describe the relationships of domain entity types, not of the metamodeling entities. In other words, it is difficult to customize the metamodeling constructs in terms of other metamodeling constructs.
  - Thus, UML was extended to include new operators to express the relationships between metamodeling constructs.
• The Union operator
  - This operator describes a full union of two UML classes. All attributes and capabilities are combined into one class, thus creating a domain entity that is the union of two domain entities

• Example
  - A tire manufacturer gauges its plant output with light gates (for height of tire stack) and sonic detectors (for width of tire, to determine tire type). The data from these actuators are coalesced, and interpreted by two different computers.
  - Modeling these two actuators as one could make data interpretation easier, by combining height and width into one class, rather than two
- **Implementation Inheritance operator**
  - In this composition type, the child inherits all of the attributes of its parent, but receives only the associations where its parent is the container.

- **Example**
  - A man and his son exhibit Implementation Inheritance. The child gets the attributes of his father (e.g. same house, same looks), and will “inherit” the belongings owned (“contained”) by his father.
  - However, the child does not go to his father’s job, although his father has an association with work.
• **Interface Inheritance operator**
  - The complement of Implementation Inheritance
  - In this composition type, only the associations where its parent is NOT the container, and inherits no attributes whatsoever

• **Example**
  - The new manager of a pro sports team gets Interface Inheritance from the previous manager. The new manager has the same responsibility as the old manager.
  - However, he has different perks (attributes), and without the same respect from the players (container associations)
Illustrative DSML Example

• We have a Signal Flow (SF) environment (paradigm) with Compounds (which contain primitives) and Primitives (connected by I/O signals) whose behavior is specified by C code.

• We have a FSM paradigm with States connected by Transitions

• Now, we want to be able to express the behavior of Primitives using an FSM, instead of C code.

• We want to combine the two, resulting in:
  - Primitive “FSMNode” that contains an FSM spec
  - FSMNodes should contain States, but States should not contain FSMNodes
  - FSMNodes can have I/O signals connecting it to States

Sztipanovits
Illustrative Example
Illustrative Example
• Overview of Metamodeling
• Abstract Syntax Metamodeling
  – Concepts
• Metamodeling languages
  – UML Class Diagrams and OCL
  – Meta-GME
  – More Stuff on Metamodeling
• Standardization Trends
  ➔ ECLIPSE
  – OMG
Eclipse: Ecore Metamodel
• Metamodeling means designing how we want to think about a system.
• How should we start the metamodeling process?
• What does it mean: the MOF Metamodel is a meta-metamodel.
• Why do we consider abstract syntax metamodeling as structural modeling?
• What is the semantics of structural modeling?
Additional comments
UML Activity Diagram

- A diagram that depicts behavior using a control and data-flow model.
- Activity modeling emphasizes the sequence and conditions for coordinating lower-level behaviors, rather than which classifiers own those behaviors.
UML Interaction Diagrams

- Including several types of diagrams that emphasize object interactions.
  - Sequence diagram
  - Communication diagram
  - Interaction overview diagram
- A sequence diagram example: Withdraw Cash
UML State Machine Diagrams

- A state machine diagram depicts discrete behavior modeled through finite state-transition systems.
- It specifies the sequence of states that an object or an interaction goes through during its life in response to events, together with its responses and actions.
• Use cases are a mean for specifying required usages of a system.

• A use case diagram is a diagram that shows the relationships among actors, the systems, and use cases.
UML Component Diagrams

- The component concept addresses the area of component-based development and component-based system structuring.
- A component diagram shows the organizations and dependencies among components.
UML Deployment Diagrams

- A diagram that depicts the execution architecture of systems.
- It represents systems artifacts as nodes, which are connected through communication paths to create network systems.
- Nodes may represent either hardware devices or software execution environments.