Model Driven Architecture
Meta Modeling

Prof. Dr. Peter Thiemann

Universität Freiburg

24.05.2006
What?
- meta = above
- Define an ontology of concepts for a domain.
- Define the **vocabulary** and **grammatical rules** of a modeling language.
- Define a domain specific language (DSL).

Why?
- Concise means of specifying the set models for a domain.
- Precise definition of modeling language.

How?
- Grammars and attributions for textbased languages.
- Metamodelling generalizes to arbitrary languages (e.g., graphical)
Metamodeling

Uses

- Construction of DSLs
- Validation of Models
  (checking against metamodel)
- Model-to-model transformation
  (defined in terms of the metamodels)
- Model-to-code transformation
- Tool integration
Terms

**Domain**  restricted area of interest
- technical aspects
- factual aspects

**Syntax**  well-formedness rules
- abstract syntax
  just structure, how are the language concepts composed
- concrete syntax
  defines specific notation
- typical use:
  parser maps concrete syntax to abstract syntax
abstract syntax

\[
\text{Expr} = \text{Const} \text{ String} \\
\quad | \quad \text{Var} \text{ String} \\
\quad | \quad \text{Binop Op Expr Expr} \\
\text{Op} = \text{Add} | \text{Sub} | \text{Mul} | \text{Div}
\]

\[
\text{Binop Mul (Const "2")} \\
\quad (\text{Binop Add (Var "x") (Const "3"))}
\]

concrete syntax

\[
E ::= c | x | E \ B \ E | (E) \\
B ::= + | - | \ast | / \\
2 \ast (x + 3)
\]
Terms/Abstract Syntax
Example: UML class diagram

- **concrete syntax**

```
Person
name
salary
raise()
```

- **abstract syntax**

```
:Class
name = "Person"

:Attribute
name = "name"

:Operation
name = "raise"

:Attribute
name = "salary"
```
Static semantics defines well-formedness rules beyond the syntax.

Examples:
- "Variables have to be defined before use"
- Type system of a programming language
  "hello" * 4 is syntactically correct Java, but rejected

UML: static semantics via OCL expressions

Use: detection of modeling/transformation errors
Purpose: formal expression of key aspects of a domain

Metamodel of DSL defines abstract syntax and static semantics

Additionally:
- concrete syntax (close to domain)
- dynamic semantics
  - for understanding
  - for automatic tools

Different degrees of complexity possible
configuration options with validity check
graphical DSL with domain specific editor
Insight: **Every model is an instance of a metamodel.**

Essential: *instance-of* relationship

Model:Metamodel is like Object:Class

Definition of Metamodel by Meta-metamodel

⇒ infinite tower of metamodels

⇒ “meta” relation always relative to a model

Every element must have a classifying metaelement which

• contains the metadata and

• is accessible from the element
OMG defines a standard (MOF) for metamodeling

MOF (Meta-Object Facility) used for defining UML

Attention, confusion:
- MOF and UML share syntax (classifier and instance diagrams)
- MOF shares names of modeling elements with UML (e.g., Class)

Approach
- Restrict infinite number of metalevels to four
- Last level is deemed “self-describing”
OMG’s Four Metalevels

M3: Meta-Metamodel
- Typ: Classifier
- ID: 5346456
- Name: Classifier
  - describes

M2: Metamodel
- Typ: Klasse
- ID: 764535
- Name: Klasse
  - Attributes, Operations, Assocs
  - describes

M1: Model
- Typ: Person
- ID: 21436456
- Name: Person
  - Name, Firstn.
  - Operations: ...
  - Association: ...
  - describes

M0: Instances
- Typ: Person
- ID: 05034503
- Name: Doe
  - Given name: John
  - instanceof
Layer M0: Instances

- Level of the running system
- Contains actual objects, *e.g.*, customers, seminars, bank accounts, with filled slots for attributes etc
- Corresponds to object diagram
Layer M1: Model

- Level of system models
- Example:
  - UML model of a software system
  - Class diagram contains modeling elements: classes, attributes, operations, associations, generalizations, ...
- Concepts of M1 categorize (or classify) instances at layer M0
- Each element of M0 is an instance of M1 element
- No other instances are allowed at layer M0
Relation between M0 and M1

M0: System
Customer
title = "Dr"
name = "Joe Nobody"
Customer
title = "Mr"
name = "Mark Everyman"
Order
number = "200604"
name = "somename"

M1: Model of a System
Customer
title : String
name : String
Order
number : String
name : String
Layer M2: Metamodel
“Model of Model”

- Level of modeling element definition
- Concepts of M2 categorize instances at layer M1
- Elements of M2 model **categorize** M1 elements: classes, attributes, operations, associations, generalizations, ...  
- Examples
  - Each class in M1 is an instance of some class-describing element in layer M2 (in this case, a *Metaclass*)
  - Each association in M1 is an instance of some association-describing element in layer M2 (a *Metaassociation*)
  - and so on
Relation between M1 and M2

M1: Model

- **UML Class**
  - name: "Customer"

- **UML Class**
  - name: "Order"

- **UML Attribute**
  - name = "number"

M2: Model of a Model

- **UML Class**
  - name: String

- **UML Attribute**
  - name: String
Layer M3: Meta-Metamodell

- Level for defining the definition of modeling elements
- Elements of M3 model categorize M2 elements: Metaclass, Metaassociation, Metaattribute, etc
- Typical element of M3 model: MOF class
- Examples
  - The metaclasses Class, Association, Attribute, etc are all instances of MOF class
- M3 layer is self-describing
Relation between M2 and M3

M2: Model of a Model
- name: String

M3: Model of a Model of a Model

M2: Model of a Model

MOF Class
- name = "UML Class"

MOF Class
- name = "UML Attribute"
Overview of Layers

- **UML Attribute**
  - name: String

- **UML Class**
  - name: String

- **MOF Class**
  - name: String

- **M3: Model of a Model of a Model**

- **M2: Model of a Model**
  - **UML Class**
    - name: String
  - **UML Attribute**
    - name: String

- **M1: Model of a System**
  - **Customer**
    - title: String
    - name: String
  - **Order**
    - number: String
    - name: String

- **M0: System**
  - **Customer**
    - title = "Dr"
    - name = "Joe Nobody"
  - **Customer**
    - title = "Mr"
    - name = "M. Everyman"
  - **Order**
    - number = "200604"
    - name = "somename"
Models on the same metalevel may have different degrees of abstraction
Transformations map between models of different abstraction levels
Source and target model of a transformation may be defined by different metamodels
MOF vs UML

- UML (M2) is an instance of MOF (M3)
- UML is older than MOF
- UML had to change to suit MOF
- MOF reuses concrete syntax and some model elements
Designing a DSL

- Definition of a new M2 language too involved
- Typical approach: Extension of UML
- Extension Mechanisms
  - Extension of the UML 2 metamodel applicable to all MOF-defined metamodels
  - Extension using stereotypes (the UML 1.x way)
  - Extension using profiles (the UML 2 way)
MOF sanctions the derivation of a new metaclass 
**CM::Component** from **UML::Class**

**CM::Component** is an instance of **MOF::Classifier**

the generalization is an instance of MOF’s **generalizes** association
Extending the UML Metamodel/Concrete Syntax

1. Explicit instance of metaclass
2. Name of metaclass as stereotype
3. Convention
4. Tagged value with metaclass
5. Own graphical representation (if supported)
Adding to a Class

- "just" inheriting from `UML::Class` leads to an identical copy
- Adding an attribute to the `CM::Component` metaclass leads to
  - an attribute value slot in each instance
  - notation: tagged value (typed in UML 2)
Meta vs Generalization
Simple specialization mechanism of UML
No recourse to MOF required
Tagged Values untyped
No new metaassociations possible
Extension of the stereotype mechanism

Requires “Extension arrow” as a new UML language construct (generalization with filled arrowhead)

Not: generalization, implementation, stereotyped dependency, association, …

Attributes ⇒ typed tagged values

Multiple stereotypes possible
Profiles make UML into a family of languages.

Each member is defined by application of one or more profiles to the base UML metamodel.

Tools should be able to load profiles and corresponding transformations.

Profiles have three ingredients:
- stereotypes
- tagges values
- constraints

Profiles can only impose further restrictions.

Profiles are formally defined through a metamodel.
Example Profile for EJB

**Profile:**
- EJB

**Stereotypes:**
- Component
- Bean
- EntityBean
- SessionBean
- Artifact
- JAR
- StateKind
  - stateful
  - stateless
- Interface
- Remote
- Home

**Context Bean:**
- inv: realization-> select(hasStereotype("Remote"))->size()==1
- &&
- realization-> select(hasStereotype("Home"))->size()==1
Further Aspects of Profiles

- Stereotypes can inherit from other stereotypes
- Stereotypes may be abstract
- Constraints of a stereotype are enforced for the stereotyped classifier
- Profiles are relative to a reference metamodel e.g., the UML metamodel or an existing profile
- Most tools today do not enforce profile-based modeling restrictions, so why bother with profiles?
  - constraints for documentation
  - specialized UML tools
  - validation by transformer / program generator
OCL constraints are independent of the modeling language and the metalevel

OCL on layer $Mn + 1$ restricts instances on layer $Mn$