Software Engineering
Lecture 08: Model Driven Engineering and Metamodelling

Peter Thiemann
University of Freiburg, Germany

SS 2013
Model Driven Engineering

Material


What is MDA?

- MDA = Model Driven Architecture
  - also: MD (Software/Application) Development, Model Based [Development/Management/Programming]
  - Model Driven Engineering, Model Integrated Computing

- Initiative of the OMG (trade mark)
  - OMG = Object Management Group: CORBA, UML, ...
  - open consortium of companies (ca. 800 Firmen)

- Goal: Improvement of software development process

- Approach: Shift development process from code-centric to model-centric
  - Reuse of models
  - Transformation of models
  - Code generation from models
Goals of MDA
Software Development at High Level of Abstraction

Portability and Reusability
- Development abstracts from target platform
- Technology mapping in reusable transformations
- New technology $\Rightarrow$ new transformation

Productivity
Each phase contributes to the product, not just the implementation

Documentation and Maintenance
- Changes through changes of the models
- Models are documentation $\Rightarrow$ consistency
Models in MDA

Platform

- Hardware, Virtual machine, API, ... 
- Examples: Operating system, JVM, EJB

Platform Independent Model (PIM) vs Platform Specific Model (PSM)

- Relative concepts, several levels of models possible 
- Inverse transformation PSM $\Rightarrow$ PIM unlikely

Transformation

- Formally defined mappings between models 
- Code is the ultimate model (PSM) 
- Model-to-code is a special case
Models in MDA/2

- Fachliche Spezifikation
  - PIM (Platform Independent Model)
  - Model-to-model transformation
  - PSM (Platform Specific Model)
  - Model-to-code transformation
  - Implementation

- CORBA-Modell
  - CORBA/C++ Code

- J2EE-Modell
  - J2EE/Java Code

- XML-Modell
  - XML Code
Models and Transformations
Metamodeling
Metamodeling

Intro

- 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 text-based languages.
  - Metamodeling 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
Excursion: Classifiers and Instances

- **UML Classifier**: class, interface, component, use case
- **Instance**: entity described by classifier
- **Instance description may include**
  - name (optional)
  - classification by zero or more classifiers
  - kind of instance
    - instance of class: object
    - instance of association: link
    - etc
  - optional specification of values
Excursion: Notation for Instances

- Box to indicate the instance
- Name compartment contains
  
  \[
  \text{name: classifier, classifier...} \\
  \text{name: classifier} \\
  : \text{classifier} \quad \text{anonymous instance} \\
  : \quad \text{unclassified, anonymous instance}
  \]
- Attribute in the classifier may give rise to like-named slot with optional value
- Association with the classifier may give rise to link to other association end
  direction must coincide with navigability
Excursion: Notation for Instances

- Box to indicate the instance
- Name compartment contains
  - name: classifier, classifier...
  - name: classifier
  - :classifier anonymous instance
  - : unclassified, anonymous instance
- Attribute in the classifier may give rise to like-named slot with optional value
- Association with the classifier may give rise to link to other association end
direction must coincide with navigability

Attention
Instance notation is similar to classifier notation.
Excursion: Notation for Instances (Graphical)

Ship
- name: String
- gross weight: Integer
- country: String

Sailor
- name: String
- rank: String

QE2: Ship
- name = "QE2"
- gross weight = 70327
- country = "GB"

captainBates: Sailor
- name = "N. Bates"
- rank = "Captain"

top: classes; bottom: instances
Terminology/Syntax

Syntax: well-formedness rules for phrases / sentences

- abstract syntax
  typically a tree or graph structure, how are the language concepts composed

- concrete syntax
  defines specific notation (character string or picture)

- typical use:
  parser maps concrete syntax to abstract syntax
Terminology/Abstract Syntax

Example: Traditional abstract syntax; arithmetic expressions

- Abstract syntax (in F# notation)
  
  ```fsharp
type Expr = Const of string
             | Var of string
             | Binop of Op * Expr * Expr

type Op = Add | Sub | Mul | Div

val aTree = Binop (Mul, Const "2",
                    Binop (Add, Var "x", Const "3"))
```

- Concrete syntax (context-free grammar)
  
  ```
  E ::= c | x | E B E | (E)
  B ::= + | - | * | /
  
  2 * (x + 3)
  ```
Terminology/Abstract Syntax
Example: UML class diagram

- Concrete syntax

- Abstract syntax (instance of the metamodel)
Terminology/Static Semantics

- **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
Terminology/Domain Specific Language (DSL)

- 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
Model and Metamodel
Insight: **Every model is an instance of a metamodel.**

Essential: *instance-of* relationship

Every element must have a classifying metaelement which
- contains the metadata and
- is accessible from the element

Relation Model: Metamodel is like Object: Class

Definition of Metamodel by Meta-metamodel

⇒ infinite tower of metamodels

⇒ “meta” relation always relative to a model
Metamodeling a la OMG

- OMG defines a standard (MOF) for metamodeling
- MOF (Meta Object Facilities) used for defining UML
- Confusion alert:
  - MOF and UML share syntax (classifier and instance diagrams)
  - MOF shares names of modeling elements with UML (e.g., Class)
- Approach taken in MOF
  - 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

M2: Metamodel
- Typ: Klasse
- ID: 764535
- Name: Klasse
- Features: Attributes, Operations, Assoc’s, ...

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

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

- Level of the running system
- Contains actual objects, e.g., customers, seminars, bank accounts, with filled slots for attributes etc
- Example: 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, ...
- Elements of M1 categorize elements at layer M0
- Each element of M0 is an instance of M1 element
- No other instances are allowed at layer M0
Metamodeling OMG’s Four Metalevels

Relation between M0 and M1

M0: System

- :Customer
  - title = "Dr"
  - name = "Joe Nobody"

- :Customer
  - title = "Mr"
  - name = "Mark Everyman"

M1: Model of a System

- Order
  - number = "200604"
  - name = "somename"
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

<<instance of>>

<<instance of>>

<<instance of>>
Layer M3: Meta-Metamodel

- 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
name = "UML Class"

M3: Model of a Model of a Model
name = "UML Attribute"
Overview of Layers

Metamodeling OMG's Four Metalevels

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

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

M2: Model of a Model
- Customer
  - title: String
  - name: String
- Order
  - number: String
  - name: String

M3: Model of a Model of a Model
- MOF Class
  - name: String

UML Class
- name: String

UML Attribute
- name: String
Excerpt from MOF/UML
Applications of Metamodeling
Applications of Metamodeling

Feature Modeling

- Feature models are a tool for domain analysis
  - Provide a hierarchical view of features and their dependencies
  - Establish an ontology for categorization
- Visualized by feature diagrams
- Popularized for Generative Programming by Czarnecki and Eisenäcker
- Also for analyzing other domains
Feature Modeling

Example

- Hierarchical, but **not** is-a relation (as in a class diagram)
- Features may be qualified as required, optional, alternative, or *n-of-m* (selection)
Feature Modeling

MOF-based Metamodel
Feature Modeling

Feature Model in Abstract Syntax

- **additionalFeatureSFG**
  - FM::SubfeatureGroup
  - kind="optional"

- **AdditionalFeature**
  - FM::Feature
  - name="AdditionalFeatures"

- **addFeatureTwoSFG**
  - FM::SubfeatureGroup
  - kind="nOfM"

- **threadFeature**
  - FM::Feature
  - name="ThreadSafety"

- **boundsFeature**
  - FM::Feature
  - name="BoundsCheck"

- **typeFeature**
  - FM::Feature
  - name="TypeCheck"

- **speedFeature**
  - FM::Feature
  - name="Speed"

- **memoryFeature**
  - FM::Feature
  - name="MemoryUsage"

- **stackFeature**
  - FM::Concept

- **optimizationDFG**
  - FM::SubfeatureGroup
  - kind="optional"

- **optimizationFeature**
  - FM::Feature
  - name="Optimization"

- **optimizationKindSFG**
  - FM::SubfeatureGroup
  - kind="alternative"
Feature Modeling
Extended Metamodel and Concrete Syntax

New feature ⇒
- new attribute in metamodel
- new slot in model
- extension of concrete syntax
Applications of Metamodelling

Component Modeling

- Domain specific modeling language for small and embedded systems
- Main abstraction: component
- A component may
  - provide services via interfaces
  - require services via interfaces
  - have configuration parameters
  - be an application (does not provide services)
Component Modeling

Example
Component Modeling

Simple Component Metamodel

```
context Application
  inv: ports->select(oclIsKindOf(ProvidedPort))->isEmpty

context PortDependency
  {subsets Attributes}
  ConfigParam *
  ports * 1
tofrom Port
  ProvidedPort
  RequiredPort

context Application
  inv: to.Interface = from.Interface
```

Peter Thiemann (Univ. Freiburg) Software Engineering SWT 40 / 42
Component Modeling

MOF-based Simple Component Metamodel

```plaintext
context PortDependency
inv: to.Interface = from.Interface
```
Summary

- Model Driven Engineering requires customized models on many levels
- Metamodeling required for defining custom models
- MOF is OMG sanctioned toolbox for metamodeling