Model-Driven Software Engineering

Metamodels and Domain-Specific Languages II

Dr. Jochen Küster  (jku@zurich.ibm.com)
Motivation: Common Aspects of MDSE Approaches

- In MDSE approaches, the use of models and model transformations is proposed.
- Models are expressed in UML, an extension of UML, or a domain-specific language.
- The syntax and semantics of models used in a MDSE approach has to be clearly defined.

Modeling language
Extension of UML
Domain-Specific Language
Syntax and semantics of modeling languages
Contents

- Defining DSLs using UML Profiles
- Defining and Using Profiles in Rational Software Architect
- Defining DSLs using Metamodeling and Constraints
- The XMI Standard for Model Interchange
- Summary and Literature
Defining DSLs using UML Profiles
Introduction to UML Profiles

- A UML profile is a lightweight extension mechanism of UML
- A UML profile can be used to customize UML to a specific domain

- A UML profile consists of
  - Stereotypes
  - Tagged values
  - Constraints

- Profiles are used widely for specializing UML
- Tools support the definition of profiles
Examples of Models with Profile Applications

```
<BusinessEntity>
  Account
  <UniqueId> number : Integer
  balance : Float
</BusinessEntity>

<BusinessEntity>
  Customer
  <UniqueId> id : String
  lastName : String
  firstName : String
</BusinessEntity>

<component>
  AccountManagement
</component>

<component>
  Customer Management
</component>

<component>
  Fulfillment
</component>
```
Reasons for Applying Profiles

- Define a new notation for already existing elements
  - Example: Special symbols for entity classes

- Define a syntax for constructs that do not yet have notation

- Add or refine semantics of elements
  - Example: Definition of semantics of PseudoStates in Statemachines

- Add constraints for elements in order to restrict their usage
  - Example: Additional constraints for BusinessEntities

- Add information to be used by model transformations and code generators
  - Example: EJBEntityBean is translated into an EJB entity bean by code generation

- Define and apply terminology that is appropriate for a specific platform or domain
Profiles – A Closer Look

- **Stereotype**
  - A stereotype describes how an existing metaclass is extended
  - A stereotype can change the graphical appearance of the extended model element by defining a new appearance
  - A stereotype uses the same notation as a class with the keyword `<<stereotype>>` being shown

- **Tagged Values**
  - A stereotype may have attributes which become tagged values when the stereotype is applied
  - Allows to add attributes

- **Constraints**
  - Constraints can be added to stereotypes

Clock with OSVersion attribute

A clock has no reference to…
Profile Definition Example in the UML Standard

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Base Class</th>
<th>Parent</th>
<th>Tags</th>
<th>Constraints</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EJBEntityBean</td>
<td>Component</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>Indicates that a Component represents an EJB Entity Bean, a component that manages the business logic of an application.</td>
</tr>
<tr>
<td>«EJBEntityBean»</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EJBSessionBean</td>
<td>Component</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>Indicates that a Component represents an EJB Session Bean, a component that processes transactions.</td>
</tr>
<tr>
<td>«EJBSessionBean»</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EJBMessagedrivenBean</td>
<td>Component</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>Indicates that a Component represents an EJB Message-Driven Bean, a component that handles messages, and is invoked on the arrival of a message.</td>
</tr>
<tr>
<td>«EJBMessagedrivenBean»</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EJBHome</td>
<td>Interface</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>Indicates that the Interface is an EJB Home interface that supports lifecycle and class-level operations.</td>
</tr>
<tr>
<td>«EJBHome»</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EJBRemote</td>
<td>Interface</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>Indicates that the Interface is an EJB Remote interface that supports business-specific operations.</td>
</tr>
<tr>
<td>«EJBRemote»</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EJBService</td>
<td>Interface</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>Indicates that the Interface is an EJB Service interface that is exposed as a webservice definition.</td>
</tr>
<tr>
<td>«EJBService»</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EJBCreate</td>
<td>Method</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>Indicates that the Method is an EJB Create Method that facilitates a create operation.</td>
</tr>
<tr>
<td>«EJBCreate»</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EJBBusiness</td>
<td>Method</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>Indicates that the Method is an EJB instance-level method that supports the business logic of the EJB associated with the Remote interface.</td>
</tr>
<tr>
<td>«EJBBusiness»</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EJBSecurityRoleRef</td>
<td>Association</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>Indicates an Association between an EJB client and an EJB Role Name Reference supplier.</td>
</tr>
<tr>
<td>«EJBSecurityRoleRef»</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[Source: OMG Unified Modeling Language Superstructure, Version 2.2]
Some More Standardized Profiles

- **UML Profile for CORBA**: Provides a standard means for expressing the semantics of CORBA IDL using UML notation and support for expressing these semantics with UML tools.

- **UML Profile for Enterprise Application Integration (EAI)**: Provides a metadata interchange standard for information about accessing application interfaces. The goal is to simplify application integration by standardizing application metadata for invoking and translating application information.

- **UML Profile for Modeling and Analysis of Real-time and Embedded Systems (MARTE)**: This specification of a UML™ profile adds capabilities to UML for model-driven development of Real Time and Embedded Systems (RTES).

- and many more.
Behind the UML Profile Mechanism

- A stereotype extends a class and adds new subclasses of this class to the language.
- Constraints further restrict the valid instance of the extended metamodel.
- Tagged values allow to describe attributes of stereotypes by (type,value) pairs.
- A profile is dependent on the reference metamodel which can either be UML or another profile.
UML Profile Definition Example – Conceptual Level

Classes from UML meta model

Stereotypes defined in profile

Extensions
Profile Definition in the UML metamodel

(Source: OMG Unified Modeling LanguageTM (OMG UML), Superstructure Version 2.3
OMG Document Number: formal/2010-05-05)
Benefits of UML Profiles for MDSE

- Profiles allow to define a domain-specific language based on standard UML
- Profiles are a lightweight extension, no meta-modeling is required
- Many UML tools support definition of UML profiles
- Generators in MDSD can make use of profiles to provide customized code generation
Defining and Using Profiles in Rational Software Architect
Guidelines for Creating UML Profiles

- Each stereotype in a UML profile should specialize semantics of the stereotyped element
- A UML profile can only make use of stereotypes, constraints and tagged values
- A UML profile should be exchangeable between tools and should be reusable
- A UML profile should list relevant existing elements
- A UML profile can contain special visualizations of stereotypes
Creating a new Profile in IBM Rational Software Architect (in a nutshell)

1. Create a new UML profile project where stereotypes, tagged values

2. Create a new stereotype
   1. Specify its name
   2. Specify its base class (if required is selected it will be automatically applied to all derived classes)
   3. Specify its presentation (optional)
   4. Specify its attributes (optional)

3. Release the profile

4. Reference the profile from the project where it should be used

5. Apply stereotypes from the profile to appropriate model elements
Create a Profile in Rational Software Architect
Release the Profile
Reference Profile from the Project where it is used
Defining DSLs by Metamodeling and Constraints
Metamodeling - Motivation

- **UML profiles have limitations**
  - Each stereotyped class extends a metaclass of an existing metamodel (e.g. UML)
  - stereotyped class can specialize semantics but not completely redefine semantics

- Metamodeling allows to start from the beginning

- **Two flavors**
  - Completely new metamodel
  - Existing metamodel is extended

- Metamodelling is a heavyweight approach to defining a DSL compared to UML profiles
Using metamodeling one can define a new modeling language.

Syntax of the new modeling language is defined using meta classes (on M2 level).

Usually one uses MOF as a meta-metamodel.
Metamodeling Example: Feature Metamodel

- Feature modeling is a common approach in product line engineering
- All classes are instance of MOF Class
- groupKind is instance of a MOF Attribute

Adapted from T. Stahl et al.
What does the Feature Metamodel express?

- A Feature can have a number of SubFeature groups
- A SubFeature contains various subfeatures for which it acts as a parent
- A Concept is a special form of a Feature which does not have a parent
- A SubFeature group can either be “required”, “alternative”, “optional” or “nOfM”
Instantiation of the Feature Metamodel
Defining Concrete Syntax for the Feature Metamodel

- **Concrete syntax can be defined by a mapping of metamodel elements to a visualization**

- **Example: SubfeatureGroup -> Feature association**
Steps for Creating DSLs using Metamodelling

1. Define the Metamodel of the DSL
   - Identify concepts and how they relate to each other

2. Define constraints on the metamodel to restrict possible unwanted instances
   - Using informal notation and then formalizing them using OCL

3. Define concrete syntax
   - How should the model look like if it is drawn by the end user?

4. Define abstract syntax to concrete syntax mapping
   - In order to relate concepts to the way how they should be represented

5. Define semantics of the language
   - Often done informally using explanations in natural language

6. Define model transformations, code generators or editors
The XMI Standard for Model Interchange
XMI

- XMI = XML Metadata Interchange
- OMG Standard for exchanging MOF-based models
- Current version is XMI version 2.1.1
- Defines how a MOF-based model is translated into an XML based presentation
- Uses DTD production rules
- Can then be used for interchange of models
Bootstrapping of XMI documents

```xml
<?xml version="1.0" encoding="UTF-8"?>

- XML processing instruction

<uml:Model xmi:version="2.1" xmlns:xmi=http://schema.omg.org/spec/XMI/2.1
  ...

- defines version and namespace
  - here xmi is declared as a namespace with the URI
Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<uml:Model xmi:version="2.1" xmi:instanceID="http://schema.omg.org/spec/XMI/2.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:uml="http://schema.omg.org/spec/UML/2.1.1"
xsi:schemaLocation="http://schema.omg.org/spec/UML/2.1.1 http://www.eclipse.org/uml2/2.0.0/UML"
xmi:id="_yPGbtUNAEeC-bPCVvM7xgA" name="Blank Model">
  <packageImport xsi:type="uml:PackageImport" xmi:id="_yPGbtUNAEeC-bPCVvM7xgA">
    <importedPackage xsi:type="uml:Model"
      href="jar:file:/C:\%5CProgram%20Files%5CIBM%5CSDP70Shared%5Cplugins%5Corg.eclipse.uml2.uml.resources_2.0.3.v2008022622248.jar!/libraries/UMLPrimitiveTypes.library.uml#_0"/>
  </packageImport>
  <packagedElement xsi:type="uml:Class" xmi:id="_yPGbtkNAEeC-bPCVvM7xgA" name="Library">
    <ownedAttribute xsi:type="uml:Property" xmi:id="_yPGbt0NAEeC-bPCVvM7xgA" name="name"
      visibility="private">
      <type xsi:type="uml:PrimitiveType"
        href="jar:file:/C:\%5CProgram%20Files%5CIBM%5CSDP70Shared%5Cplugins%5Corg.eclipse.uml2.uml.resources_2.0.3.v2008022622248.jar!/libraries/UMLPrimitiveTypes.library.uml#String"/>
    </ownedAttribute>
  </packagedElement>
  ...
</uml:Model>
```
Class Representation and Attribute Representation

- A class is represented by an XML element, with an XML element or attribute for each property
  - `packagedElement` is the XML element

- An attribute is represented by an XML element or by an XML attribute
  - `ownedAttribute` is the XML element
Object Identity

- id XML attribute has type ID
- value of id attribute must be unique within a document
- used to specify relationships among objects

- uuid XML attribute must contain a globally unique identifier
- must be unique across documents
OCL Constraints and Associations in XMI

<ownedRule xmi:type="uml:Constraint" xmi:id="_yPGbv0NAEeC-bPCVvM7xgA" name="PageConstraint" constrainedElement="_yPGbvkNAEeC-bPCVvM7xgA">
  <specification xmi:type="uml:OpaqueExpression" xmi:id="_yPGbwENAeC-bPCVvM7xgA">
    <language>OCL</language>
    <body>self.pages >0&amp;#xD;&amp;#xA;</body>
  </specification>
</ownedRule>

<packagedElement xmi:type="uml:Association" xmi:id="_yPGbyUNAEeC-bPCVvM7xgA" memberEnd="_yPGbuENAeC-bPCVvM7xgA _yPGbykNAEeC-bPCVvM7xgA">
  <ownedEnd xmi:type="uml:Property" xmi:id="_yPGbykNAEeC-bPCVvM7xgA" name="" visibility="private" type="_yPGbtkNAEeC-bPCVvM7xgA" association="_yPGbyUNAEeC-bPCVvM7xgA"/>
</packagedElement>
Summary and References
Summary of Lecture and References

- UML profiles can be used to define Domain-Specific Languages
- Metamodels and OCL can be used to define Domain-Specific Languages
- XMI is used for model interchange

References:

- B. Pierce. Types and Programming Languages. MIT Press, 2002