Software Engineering
Model Driven Architecture
Applications of Metamodelling

Prof. Dr. Peter Thiemann
Universität Freiburg

21.07.2011
Applications of Metamodelling

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

- **MOF::Class**
- **FM::Feature**
- **FM::SubfeatureGroup**
- **FM::GroupKind**
- **MOF::Attribute**

### Class Attributes
- **type**: String
- **value**: String

### Feature Modeling...
- **features**: n
- **groups**: n
- **parent**: 1

### FM::GroupKind Attributes
- **value**:
  - "required"
  - "optional"
  - "alternative"
  - "nOfM"
Feature Modeling

Feature Model in Abstract Syntax

- **additionalFeatureSFG**
  - kind="optional"
  - **AdditionalFeature**
    - name="AdditionalFeatures"

- **stackFeature**: FM::Concept

- **optimizationDFG**
  - kind="optional"
  - **optimizationFeature**
    - name="Optimization"

- **addFeatureTwoSFG**
  - kind="nOfM"
  - **threadFeature**
    - name="ThreadSafety"
  - **boundsFeature**
    - name="BoundsCheck"
  - **typeFeature**
    - name="TypeCheck"
  - **speedFeature**
    - name="Speed"
  - **memoryFeature**
    - name="MemoryUsage"

The diagram illustrates the relationships and features of a feature model, highlighting concepts such as optimization, stack, additional features, thread safety, bounds check, type check, speed, and memory usage.
Feature Modeling

Extended Metamodel and Concrete Syntax

Metamodel

<table>
<thead>
<tr>
<th>FM::Feature</th>
<th>open:boolean</th>
</tr>
</thead>
</table>

Object diagram

```
optimizationFeature
FM::Feature
name="Optimization"
onopen=true
```

Feature diagram

Optimization

New feature $\Rightarrow$
- 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

```uml
<<application>>
  SMSApp
  CallIF
  EMSIF
  SMSIF

GSMStack

MenuUtilities

TextEditor

UIManager

lookAndFeel:String
```
Component Modeling

Simple Component Metamodel

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

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

Diagram:

- **Component** to **Port**: 1
- **Port** to **Interface**: 1
- **Application** to **Component**: *
- ** RequiredPort** from **Port**: *
- ** ProvidedPort** to **Port**: *
Component Modeling

MOF-based Simple Component Metamodel
Pitfalls in Metamodeling

How to avoid

▶ confusion with UML notation
▶ mixing metalevels

Central question

▶ what is the mapping to a programming language?
Interfaces

Every instance of **Entity** should implement **SomeInterface**

► **wrong approach**

```plaintext
<<interface>>
SomeInterface
```

Entity

► **book solution** use OCL or subsetting of metaassociation

```plaintext
Entity

realization

→exists(oclIsTypeOf(SomeInterface))
```

Entity

{subsets realization}

1

SomeInterface
Every instance of **Entity** should implement **SomeInterface**

▶ **correct solution** use OCL

```
realization
  -->select(hasStereotype("interface"))
  -->select(name="SomeInterface")
  -->size() = 1
```

```
 Entity
    \<<instanceof>>>
    \<<instanceof>>>

:Entity
```

```
  implements

  SomeInterface
```

Dependency

- **Problem:** A Component may depend on multiple Interfaces because the Component may invoke operations of the Interfaces.

- **Wrong approach** “metaclass Component depends on metaclass Interface”

- **Correct solution** a metaassociation “uses”
Identifying Attribute

An **Entity** must have an identifying attribute with name **ID** and type **String**. **Entity** is a subclass of **UML::Class**.

» **wrong approach**

<table>
<thead>
<tr>
<th>Entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID : String</td>
</tr>
</tbody>
</table>

defines a tagged value **ID** for all **Entity** instances in the model
Identifying Attribute

- correct solution

Entity

- there must be exactly one attribute with name ID
- all attributes named ID must have type String
Identifying Attribute

▶ correct solution

Entity

\[(\text{Attribute} \rightarrow \text{select( Name = "ID") \rightarrow \text{size}=1})\]
\[\text{and}\]
\[(\text{Attribute} \rightarrow \text{select( Name= "ID") \rightarrow \text{forAll( Type.Name = "String")})\]

▶ there must be exactly one attribute with name ID
▶ all attributes named ID must have type String

▶ incorrect attempt

class context Entity inv:

Attribute

\[\rightarrow \text{select (Name="ID" and Type.Name="String")}\]
\[\rightarrow \text{size() = 1}\]
Primary Key Attribute

Each instance of Entity must have exactly one attribute of type EntityPK, where EntityPK is a subclass of Attribute.

- wrong approach
- correct solution
Objects are instances of classes

Links are instances of associations
Metalevels and Instanceof
Model Elements as Instances of Metamodel Elements

The **Auto** and **Person** classes are instances of the MOF metaclass **UML::Class**

The objects **me:** and **myFather:** are instances of the MOF metaclass **UML::Object**
Metalevels and Instanceof
A Look at the Metamodel

▶ two different instanceof relations
Summary

- Metamodelling required for customizing UML
- OMG relies on MOF to define profiles
- OCL defines static semantics of models
- Metalevels should not be confused