Software Engineering

Lecture 08: Model Driven Engineering and Metamodeling

Peter Thiemann

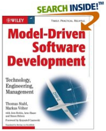
University of Freiburg, Germany

SS 2013

Model Driven Engineering

Material

- Thomas Stahl, Markus Völter. Model-Driven Software Development. Wiley & Sons. 2006.



- Anneke Kleppe, Jos Warmer. MDA Explained: The Model Driven Architecture: Practice and Promise. Pearson. 2003.
- Stephen J. Mellor, Axel Uhl, Kendall Scott, Dirk Weise. MDA Distilled: Solving the Integration Problem with the Model Driven Architecture. Pearson. 2004.

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

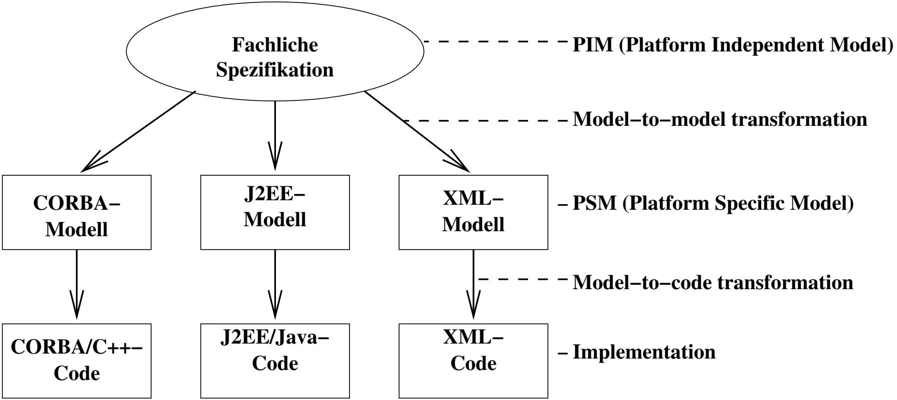
Platform Independent Model (PIM) vs Platform Specific Model (PSM)

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

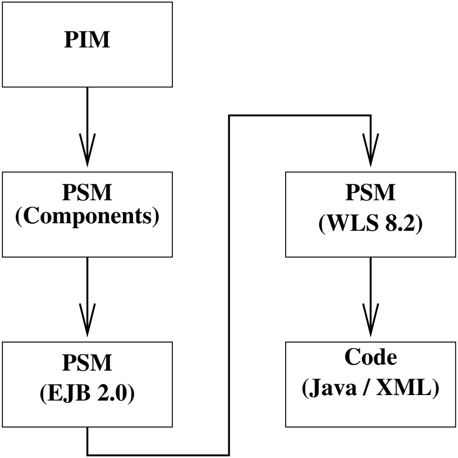
Transformation

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

Models in MDA/2



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
  <u>*name*: *classifier*, *classifier*...</u>
  <u>*name*: *classifier*</u>
  <u>: *classifier*</u>   anonymous instance
  <u>:</u>   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

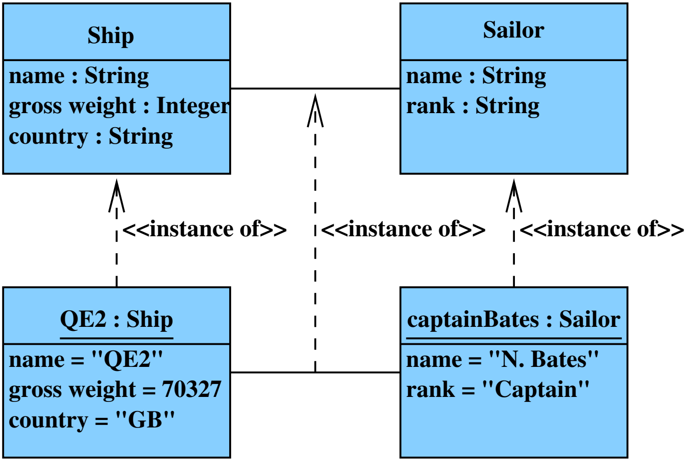
Excursion: Notation for Instances

- Box to indicate the instance
- Name compartment contains
  <u>*name*: *classifier*, *classifier* ...</u>
  <u>*name*: *classifier*</u>
  <u>: *classifier*</u> anonymous instance
  <u>:</u> 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)



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)

```
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)

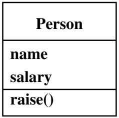
$$\begin{array}{lcl} E & ::= & \mathtt{c} \mid \mathtt{x} \mid E\ B\ E \mid (E) \\ B & ::= & + \mid - \mid * \mid / \end{array}$$

```
2 * (x + 3)
```

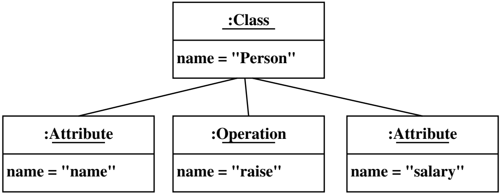
Terminology/Abstract Syntax

Example: UML class diagram

- Concrete syntax



- Abstract syntax (instance of the metamodel)



Terminology/Static Semantics

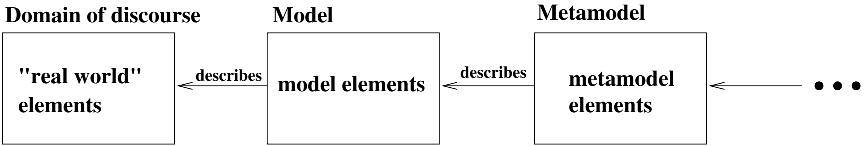
- **Static semantics** defines well-formedness rules beyond the syntax
- Examples
  - “Variables have to be defined before use”
  - Type system of a programming language
    `"hello" * 4` is syntactically correct Java, but rejected
- UML: static semantics via OCL expressions
- Use: detection of modeling/transformation 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

Model and Metamodel

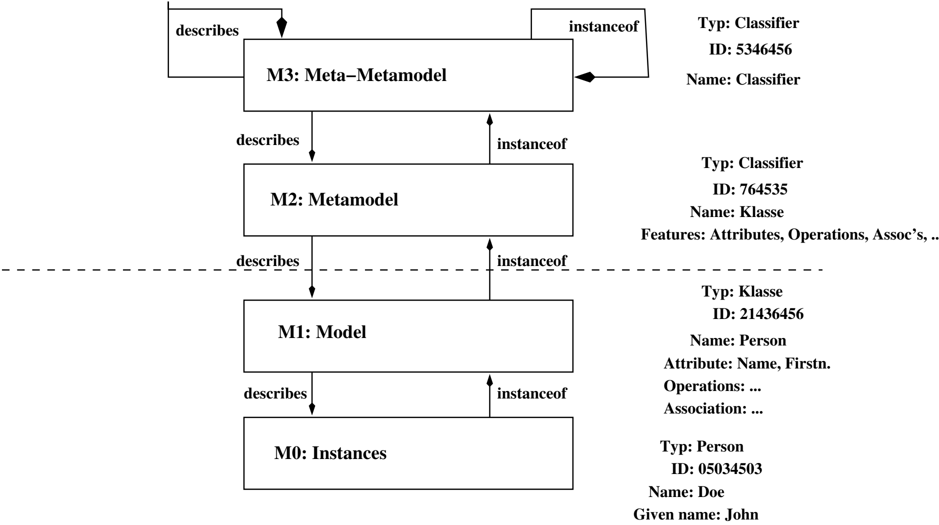


- Insight: **Every model is an instance of a metamodel.**
- Essential: *instance-of* relationship
- Every element must have a classifying metaelement which
  - contains the metadata and
  - is accessible from the element
- Relation Model:Metamodel is like Object:Class
- Definition of Metamodel by Meta-metamodel
- $\Rightarrow$ infinite tower of metamodels
- $\Rightarrow$ "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



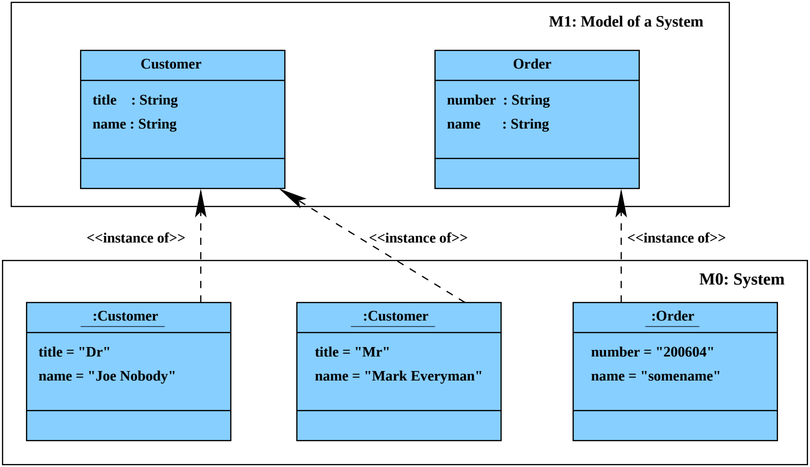
Layer M0: Instances

- 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

Relation between M0 and M1

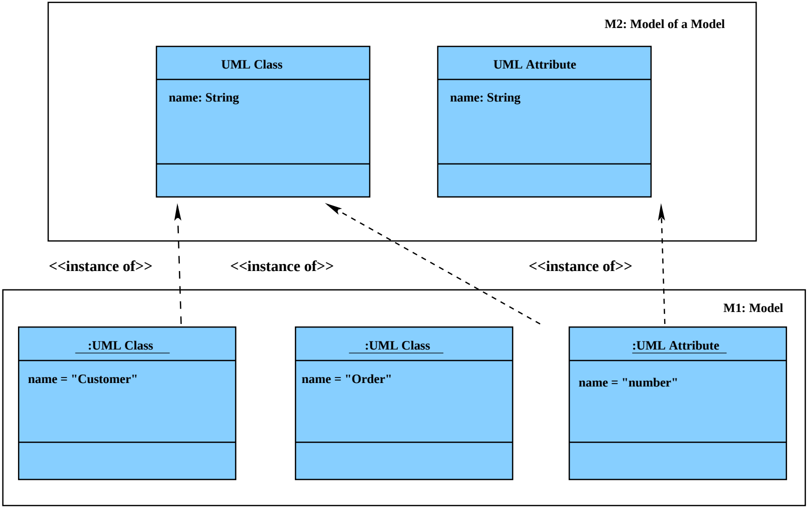


Layer M2: Metamodel

"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

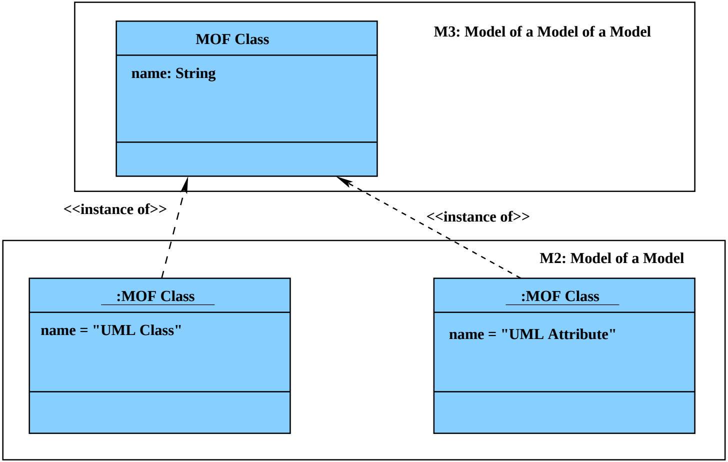
Relation between M1 and M2



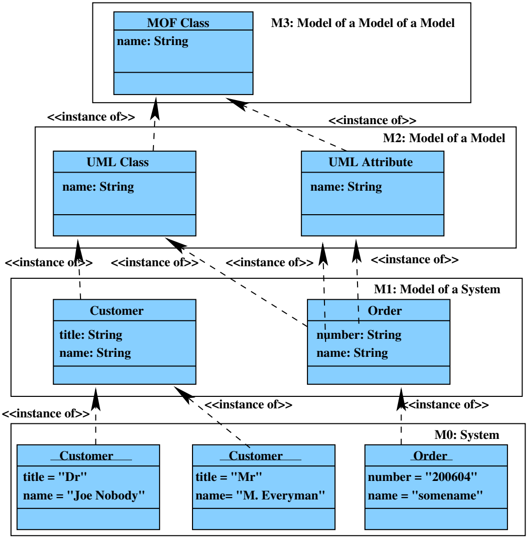
Layer M3: Meta-Metamodel

- 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

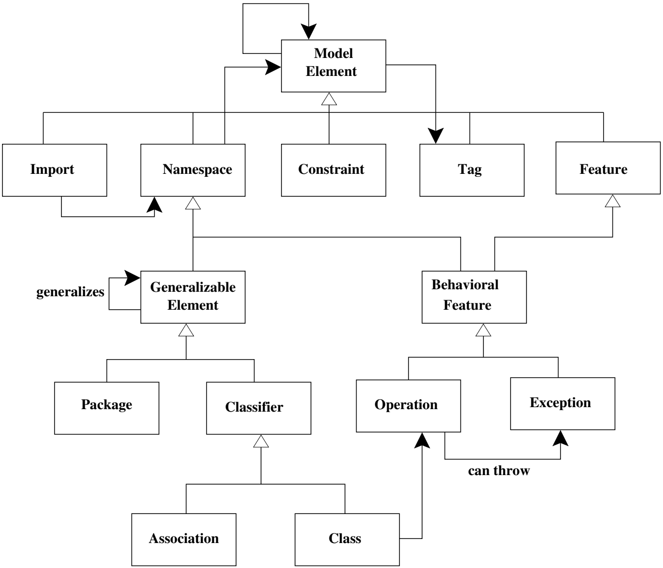
Relation between M2 and M3



Overview of Layers



Excerpt from MOF/UML



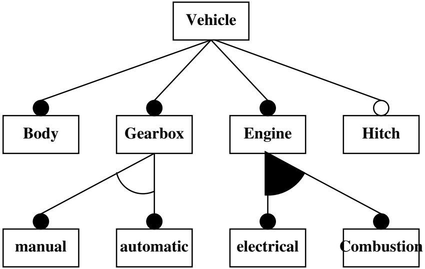
Applications of Metamodeling

Applications of Metamodeling

Feature Modeling

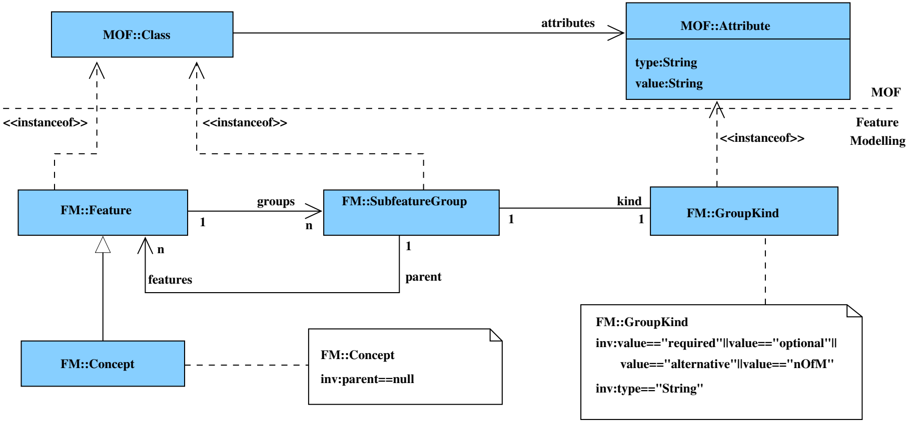
- Feature models are a tool for domain analysis
  - Provide a hierarchical view of features and their dependencies
  - Establish an ontology for categorization
- Visualized by feature diagrams
- Conceived for software domain analysis: Kang, Cohen, Hess, Novak, Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report CMU/SEI-90-TR-21. 1990.
- Popularized for Generative Programming by Czarnecki and Eisenäcker
- Also for analyzing other domains

Feature Modeling
Example



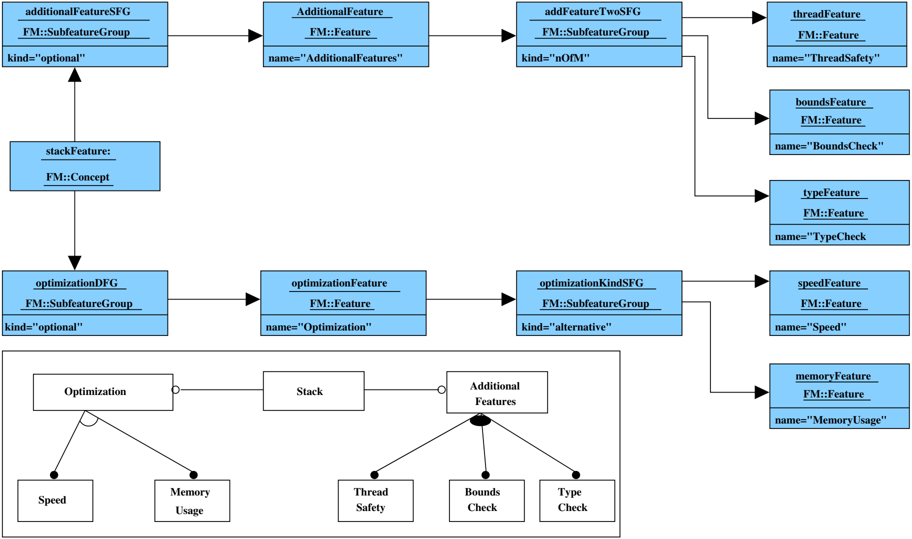
- Hierarchical, but **not** is-a relation (as in a class diagram)
- Features may be qualified as required, optional, alternative, or *n*-of-*m* (selection)

Feature Modeling
MOF-based Metamodel



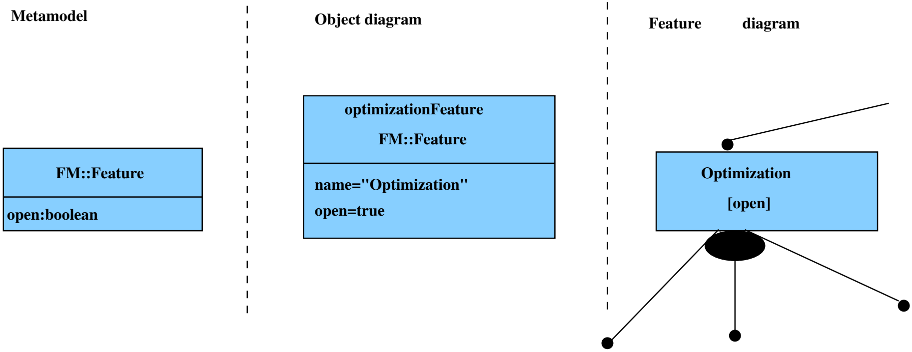
Feature Modeling

Feature Model in Abstract Syntax



Feature Modeling

Extended Metamodel and Concrete Syntax



New feature ⇒

- new attribute in metamodel
- new slot in model
- extension of concrete syntax

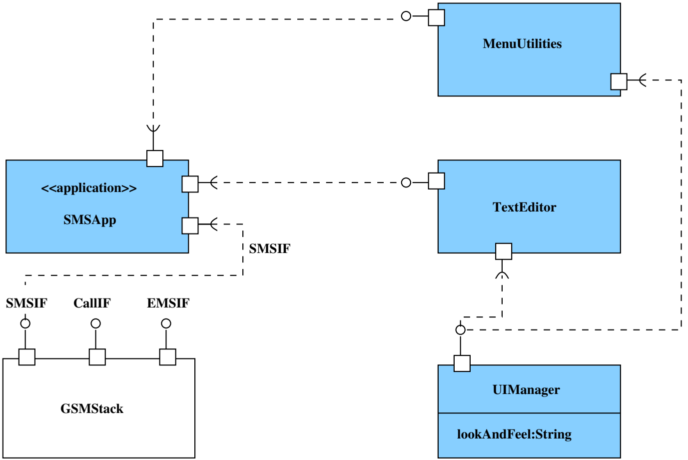
Applications of Metamodeling

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)

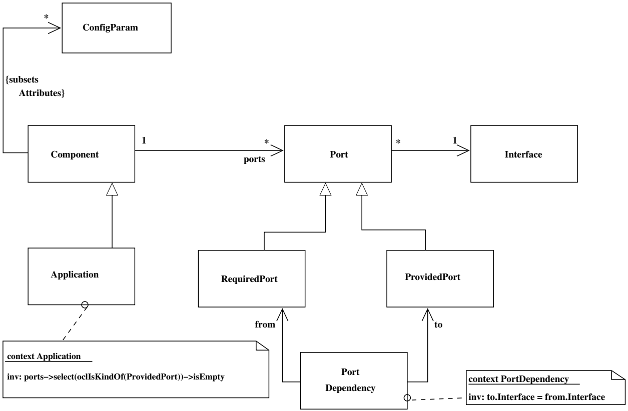
Component Modeling

Example



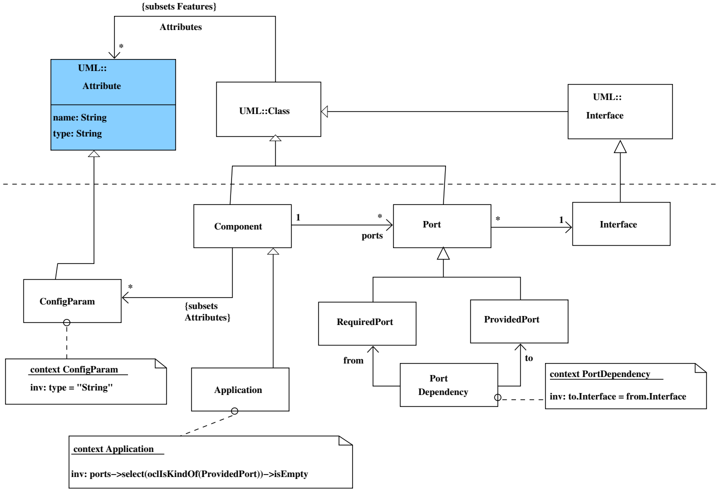
Component Modeling

Simple Component Metamodel



Component Modeling

MOF-based Simple Component Metamodel



Summary

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