

Meta-modeling and model transformations

Prof.dr. Mark van den Brand



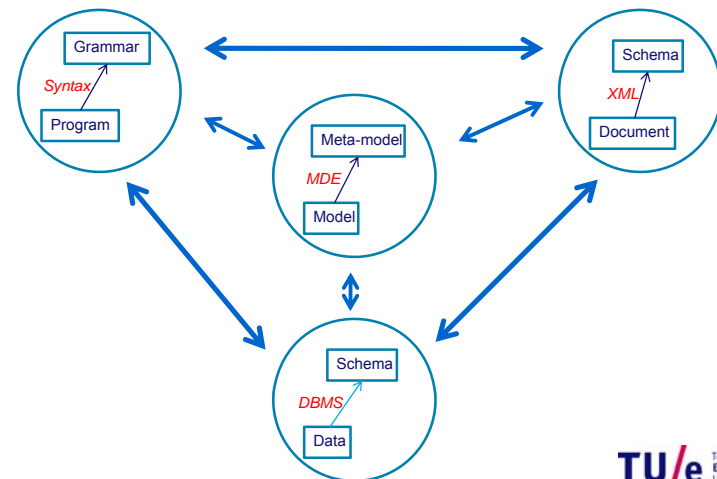
Introduction

- **Meta-modeling** is the analysis, construction and development of the frames, rules, constraints, models and theories applicable and useful for modeling a predefined class of problems.
- **Relation between meta-models and grammars?**
 - Is every meta-model a context-free grammar?
 - ...
 - Is every context-free grammar a meta-model?
 - ...

Models and meta-models

- Warmer et.al.:
 - “A model is a description of (part of) a system written in a well-defined language”
 - “A meta-model is a model that defines the language for expressing a model”
 - “A meta-model is the collection of ‘concepts’ (things, terms, etc.) within a certain domain and an attempt at describing the world around us for a particular purpose. Meta-models are also referred to as a precise definition of the constructs and rules needed for creating semantic models.”
 - “The meta-model of a language, also known as abstract syntax, is a description of all the concepts that can be used in that language.”

MDE: technology spaces



(Meta)-Meta-Model world

The 4-layer architecture

- **M3 Meta-Meta-Model Layer**
 - defines structure of the meta-metadata
- **M2 Meta-Model Layer**
 - defines the structure of the metadata
- **M1 Model Layer**
 - describes the data in the information layer
- **M0 Information Layer**
 - data we wish to describe

Grammar world

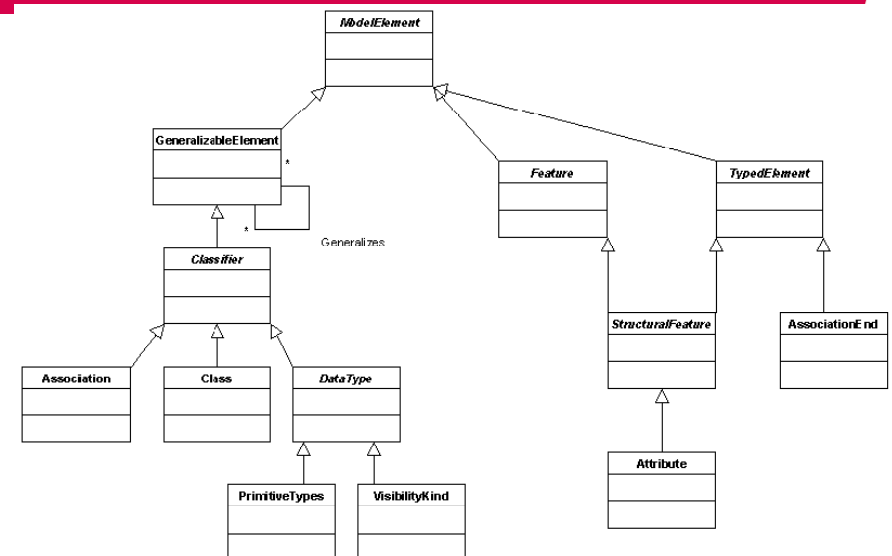
The 4-layer architecture

- **M3 (E)BNF/SDF grammar**
 - defines structure of the (E)BNF/SDF in (E)BNF/SDF
- **M2 Java grammar**
 - defines the structure of Java in (E)BNF/SDF
- **M1 Java program**
 - describes the manipulation (algorithm) of objects in the object layer
- **M0 Object layer**
 - Objects we wish to manipulate

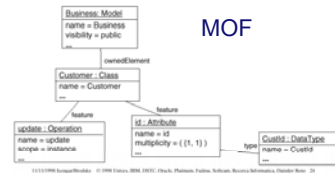
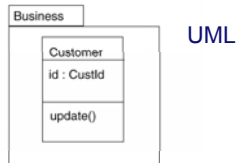
M2 and M3 Relationship

- Assuming that the model is described by the meta-model
 - The meta-model is described by the meta meta-model
 - In this case let's use as example the Meta Object Facility (MOF)

M3: MOF



UML - MOF via XMI



```
<Model>
<name>Business</name>
<visibility xmi.value="public"/>
<Class>
<name>Customer</name>
<feature>
<Attribute>
<name>id</name>
<multiplicity>
<XML.field>1</XML.field>
<XML.field>1</XML.field>
</multiplicity>
</Attribute>
</feature>
</Class>
</Model>
```

EMF Model Definition

What is an EMF "model"?

- Specification of an application's data
 - Object attributes
 - Relationships (associations) between objects
 - Operations available on each object
 - Simple constraints (e.g., multiplicity) on objects and relationships
- Essentially the Class Diagram subset of UML

What does EMF provide?

- **Meta-model**
 - A general model of models from which any model can be defined
 - Models classes, attributes, relationships, data types, etc.
 - Referred to as Ecore
 - Ecore is just another EMF model
- **EMF is used to implement EMF!**
- **Tooling support within the Eclipse framework**
- **Runtime environment**
 - Reflective and dynamic model invocation
 - XML/XMI default model serialization
- **Originally based on MOF (Meta Object Facility)**

What does EMF provide?

- **EMF is middle ground in the modeling vs. programming world**
 - Focus is on class diagram subset of UML modeling (object model)
 - Transforms models into efficient, correct, and easily customizable Java code
 - Provides the infrastructure to use models effectively in your code
- **Very low cost of entry**
 - Full scale graphical modeling tool not required
 - EMF is free
 - Small subset of UML

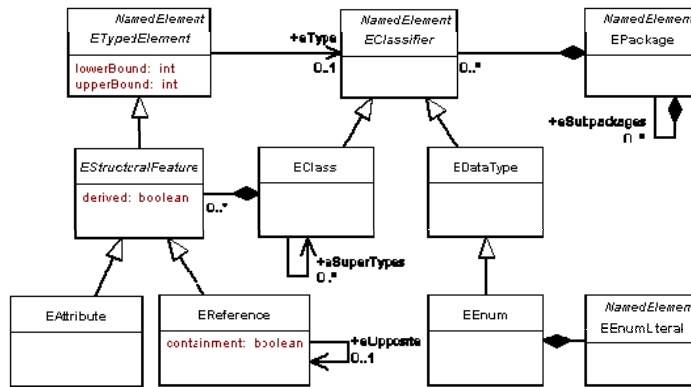
EMF Components

- **EMF Core**
 - Ecore meta model
 - Model change notification
 - Persistence and serialization
 - Reflection API
 - Runtime support for generated models
 - Change model and recorder
 - Validation framework
- **EMF Edit**
 - Helps integrate models with a rich user interface
 - Used to build editors and viewers for your model
 - Includes default reflective model editor
- **EMF Codegen**
 - Code generator for core and edit based components
 - Model importers from Rose, XML, or Java interfaces

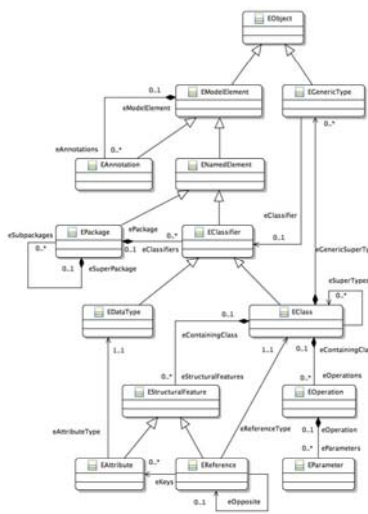
Ecore Model Creation

- Ecore model can be created within an Eclipse project via wizard using one of sources previously described
- **Output is:**
 - *modelname.ecore* file
 - Ecore model file in XMI format
 - Canonical form of the model
 - *modelname.genmodel* file
 - A “generator model” for specifying generator options
 - Decorates *.ecorefile*
 - EMF code generator is an EMF *.genmodel* editor
 - *.genmodel* and *.ecore* files automatically kept in sync

Simplified Ecore Meta-model



Extended Ecore Meta-model



Grammars, signatures and meta-models

- Context-free grammars are mapped to signatures
- A signature describes the structure of abstract syntax trees
- A meta-model can also describe the structure of abstract syntax trees, plus
 - Relations between identifiers
 - Attributes to store scope information
 - Attributes to store type information

Grammars, signatures and meta-models

SDF definition of the Booleans:

context-free syntax

```
"true"      -> BoolCon {cons{"true"}}
"false"     -> BoolCon {cons{"false"}}
Bool "&" Bool -> Bool {cons{"and"}}
Bool "|" Bool -> Bool {cons{"or"}}
"not" Bool  -> Bool {cons{"not"}}
"(" Bool ")" -> Bool {bracket}
```

Grammars, signatures and meta-models

Signature definition of the Booleans:

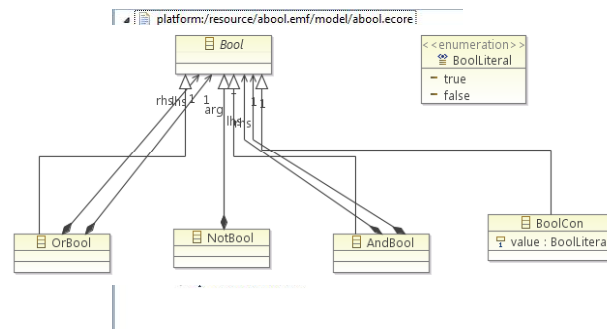
```
"true"()      -> BoolCon
"false"()     -> BoolCon
"con"(BoolCon) -> Bool
"and"(Bool, Bool) -> Bool
"or"(Bool, Bool) -> Bool
"not"(Bool)   -> Bool
```

Grammars, signatures and meta-models

- First alternative meta-model for Booleans:
 - Every operator (not, or, and) is modeled as a separate class
 - Constructors (true, false) are modeled as enumeration
- Second alternative meta-model for Booleans:
 - Binary operator is modeled as a separate class
 - Unary operator is modeled as a separate class
 - Operators are modeled as enumerations
 - Constructors (true, false) are modeled as enumeration

Grammars, signatures and meta-models

- First alternative for Boolean meta-model



Grammars, signatures and meta-models

- Xtext is a way to define the context-free syntax of a languages, alternatives are:
 - EMFtext (<http://www.emftext.org>)
 - TCS (ATLANMOD)

Grammars, signatures and meta-models

- Based on ANTLR
- Difference with SDF:
 - No modularity
 - Restricted to LL(k), so refactoring of grammar:
 - No left recursion
 - Left factorization
 - Predefined lexicals:
 - ID
 - STRING
 - INT
 - Cross references

Grammars, signatures and meta-models

- Xtext specification for Booleans:

```

Model : OrBool ;

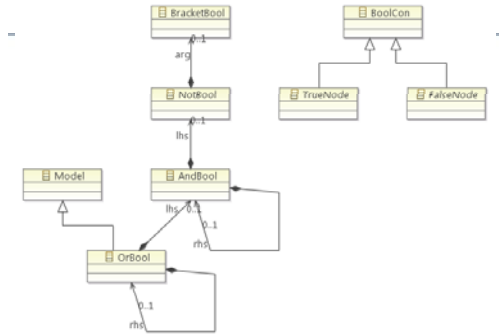
OrBool : (lhs=AndBool ('|' rhs=OrBool)? ;
AndBool : (lhs=NotBool ('&' rhs=AndBool)? ;
NotBool : '~' arg=BracketBool | arg=BracketBool ;
BracketBool : '(' arg=OrBool ')' | arg=BoolCon ;
BoolCon : {TrueNode} 'true' | {FalseNode} 'false' ;
    
```

Annotations in the original image:

- An arrow points from "No left recursion" to the `(lhs=AndBool ('|' rhs=OrBool)?` part of the `OrBool` rule.
- An arrow points from "Left factorized" to the `(lhs=NotBool ('&' rhs=AndBool)?` part of the `AndBool` rule.

Grammars, signatures and meta-models

- Resulting meta-model (of syntax tree) for Booleans:



Grammars, signatures and meta-models

- Xtext specification for Pico:

```

Model : Program ;

Program :
  'begin' decls=Decls? stats=Statements? 'end' ;

Decls: 'declare' idtypes=IdTypes ';' ;

IdTypes : pairs+=IdType (',' pairs+=IdType)*;

IdType : name=ID ':' type=Type ;

Type : {naturalType} 'natural' / {stringType} 'string' ;

Statements : statements+=Statement (';' statements+=Statement)*;
    
```

Grammars, signatures and meta-models

- Xtext specification for Pico (continued):

```

Statement : AssignStatement |
           IfStatement |
           WhileStatement ;

AssignStatement : lhs=ID '=' rhs=Exp ;

IfStatement : 'if' Exp 'then' thenpart=Statements? 'else'
             elsepart=Statements? 'fi' ;

WhileStatement : 'while' Exp 'do' dopart=Statements? 'od' ;

Exp : lhs=Term (bop=BinOp rhs=Exp)? ;

Term : id=ID / literal=STRING / number=INT / '(' Exp ')';

BinOp: {plusOp} '+' / {minOp} '-' / {concOp} '||' ;
    
```

Grammars, signatures and meta-models

- Xtext specification for Pico (with cross references):

```

Statement : AssignStatement |
           IfStatement |
           WhileStatement ;

AssignStatement : lhs=[IdType/ID] :=' rhs=Exp ;

IfStatement : 'if' Exp 'then' thenpart=Statements? 'else'
             elsepart=Statements? 'fi' ;

WhileStatement : 'while' Exp 'do' dopart=Statements? 'od' ;

Exp : lhs=Term (bop=BinOp rhs=Exp)? ;

Term : id=[IdType/ID] / literal=STRING / number=INT / '(' Exp ')';

BinOp: {plusOp} '+' / {minOp} '-' / {concOp} '||' ;
    
```

Cross reference

Tools and Platforms

- **Modelware**
 - **Platform:**
 - Eclipse + EMF
 - **Transformation languages:**
 - XTend
 - ATL
 - QVT
- **Grammarware**
 - **Platform:**
 - Meta-environment
 - **Transformation languages:**
 - ASF
 - Stratego/XT
 - Tom