Meta-modelling

Juan de Lara, Elena Gómez, Esther Guerra
{Juan.deLara, MariaElena.Gomez, Esther.Guerra}@uam.es
Computer Science Department
Universidad Autónoma de Madrid

Masters: I2TIC and Formal methods

Index

- **Two-level meta-modelling** 
  - **MOF**
  - **EMF**
- Multi-level meta-modelling
- Profiles
- Graph grammars
- Bibliography

Meta-modelling

- A model is a description of the system under study, using a language.

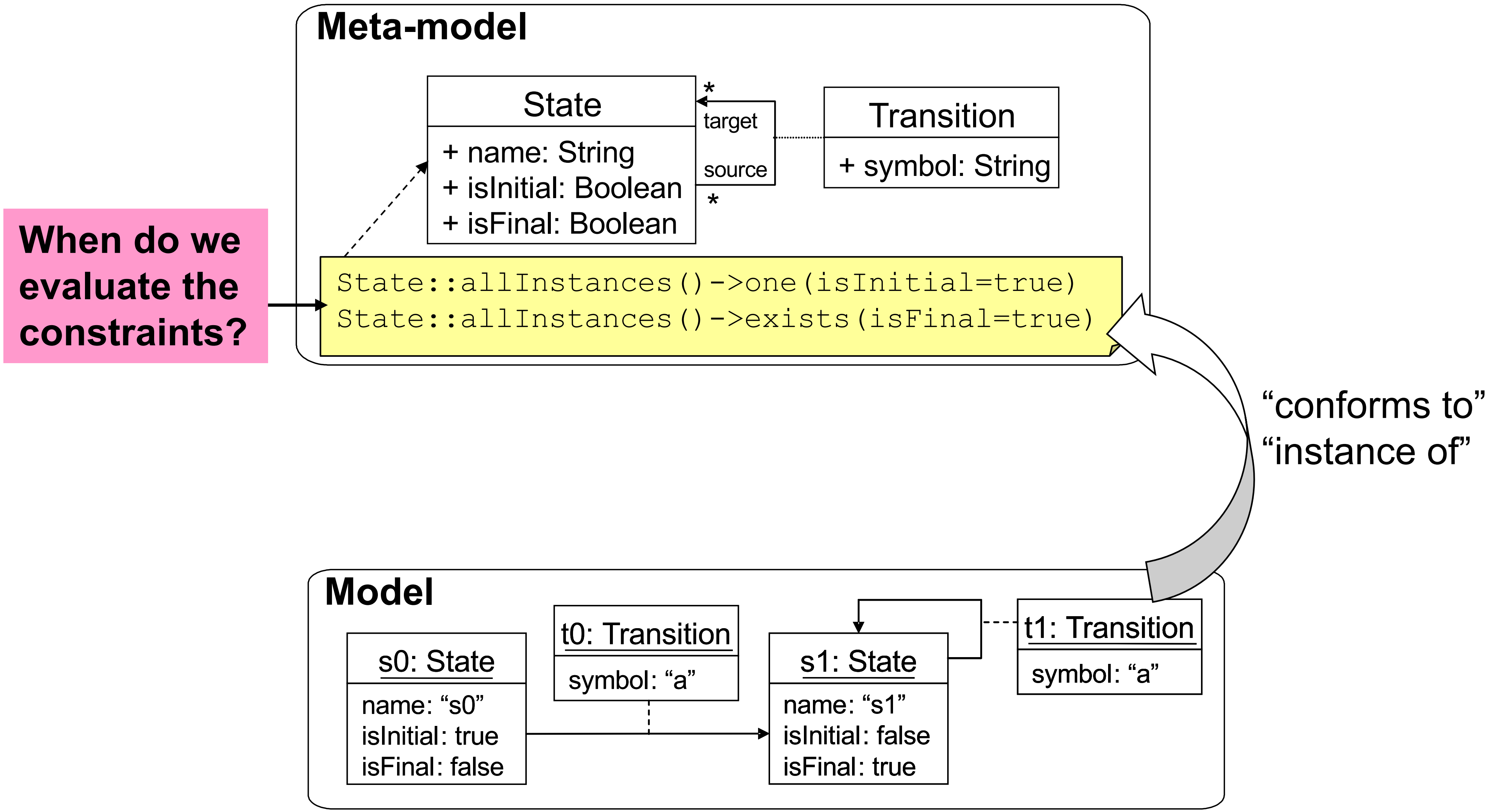
- A meta-model is a model that describes a language (i.e. it describes all syntactically valid models).

- That is, the meta-model describes the abstract syntax of the language.

- Meta-models are usually defined using class diagrams or entity-relationship diagrams.

- Additional OCL constraints.

Meta-modelling



Meta-modelling

- Strict meta-modelling: “*one element in meta-level n is instance of exactly an element in meta-level n-1*”.

- A model is a valid instance of a meta-model if…:
  - The model is structurally valid:
    - the model objects are instances of classes in the meta-model.
    - the model links are instances of associations in the meta-model.
  - The model satisfies the following constraints:
    - cardinality in associations.
    - unique keys.
    - additional OCL constraints.

  - Similar to relation between class and object diagrams.

Meta-modelling

Evaluation of constraints

- When do constraints are evaluated?

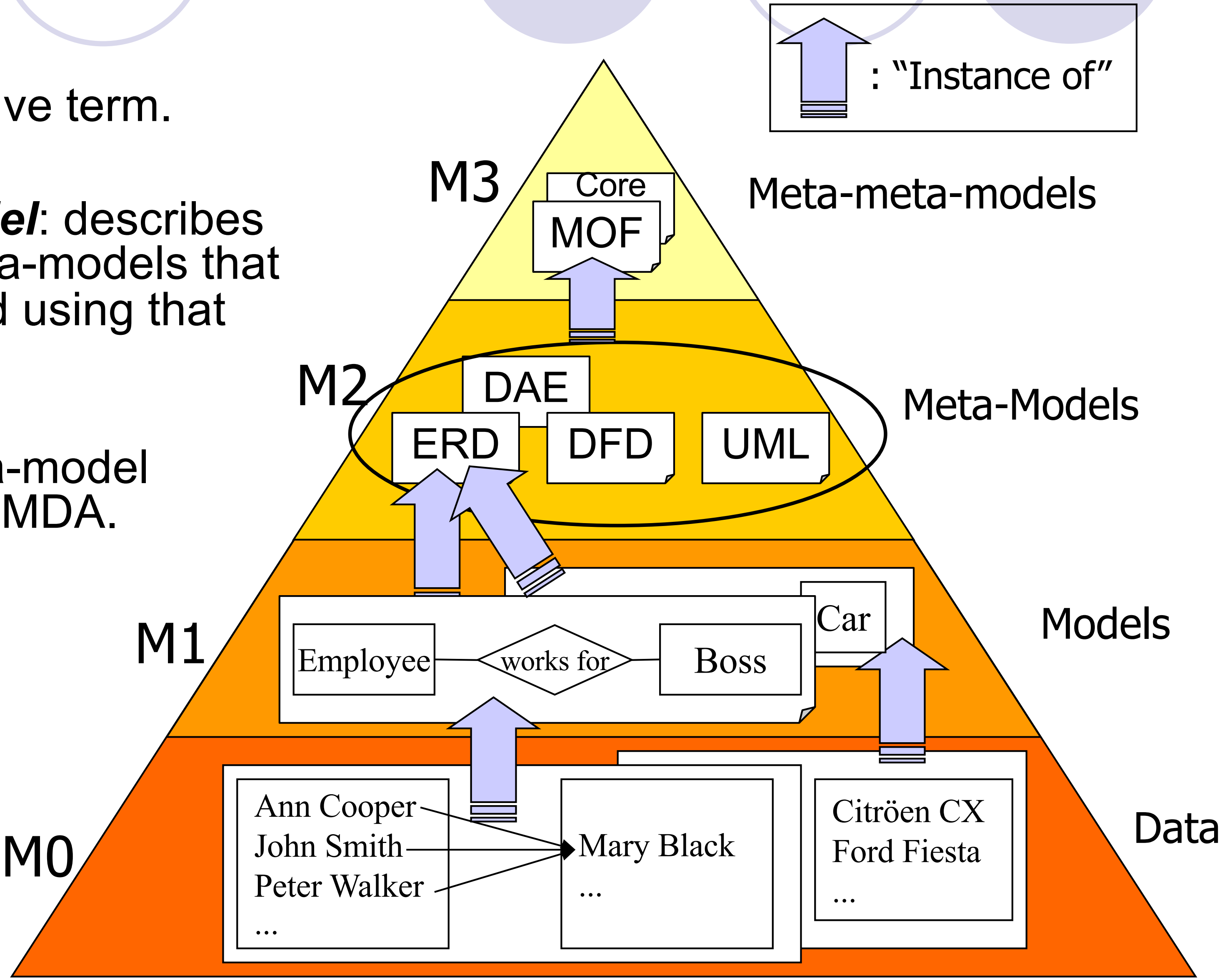
- Cardinality in associations:
  - If we have the cardinality interval [i..j]
  - Lower bound: wait until the user wants to validate the model.
  - Upper bound: report the error as soon as more than j links are created.

- What about the additional constraints? One possibility is to associate them to editing events (as pre- and post-conditions).

Meta-modelling

Levels

- "meta-" is a relative term.
- ***Meta-meta-model***: describes the set of all meta-models that can be described using that language.
- **MOF**: meta-meta-model proposed by the MDA.



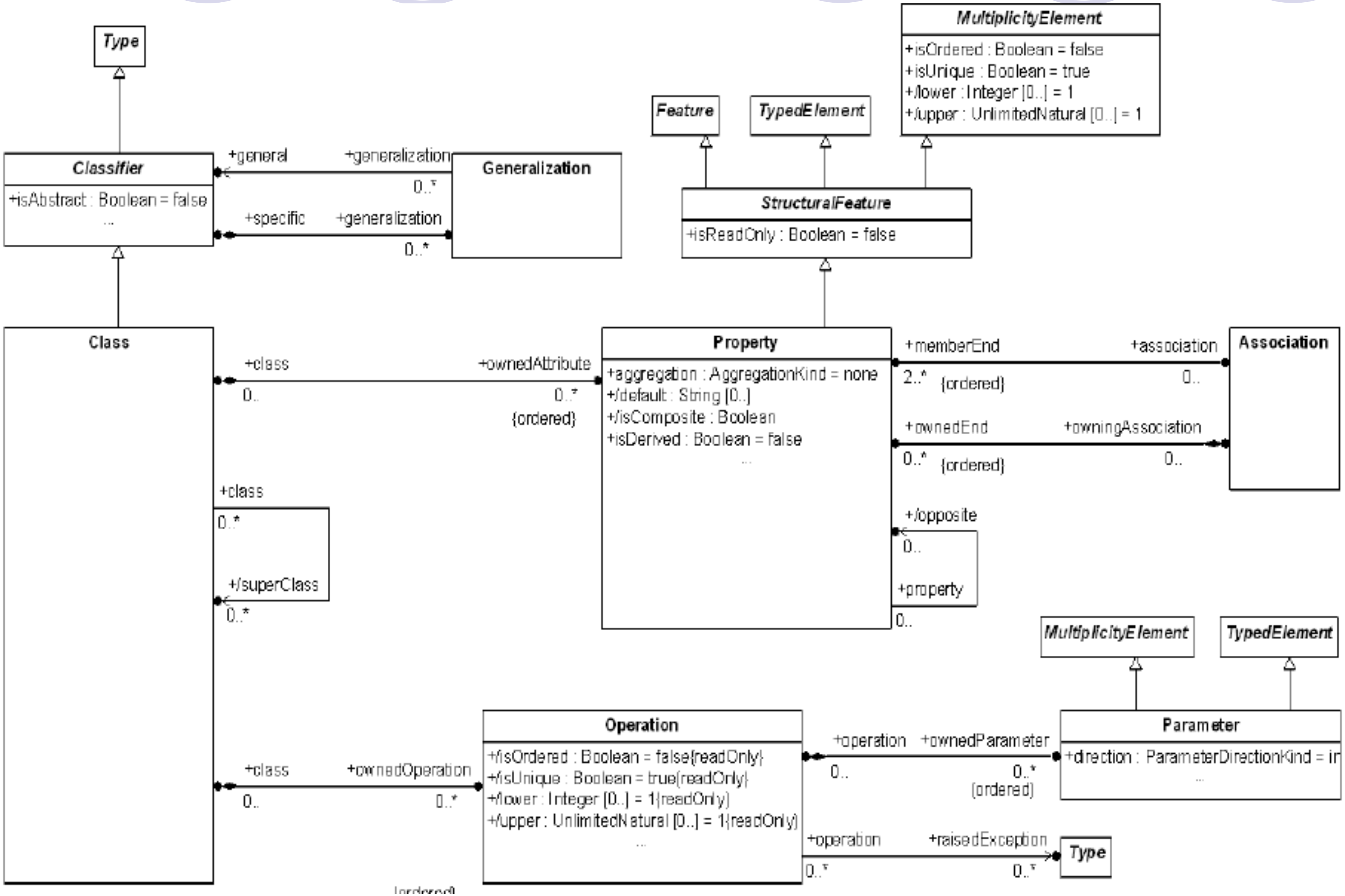
MOF

- Meta-Object Facility: OMG standard for meta-modelling

- Meta-meta-model that describes a meta-modelling language, similar to class diagrams

- Two levels:
  - Essential MOF (EMOF)
  - Complete MOF (CMOF)

MOF meta-meta-model



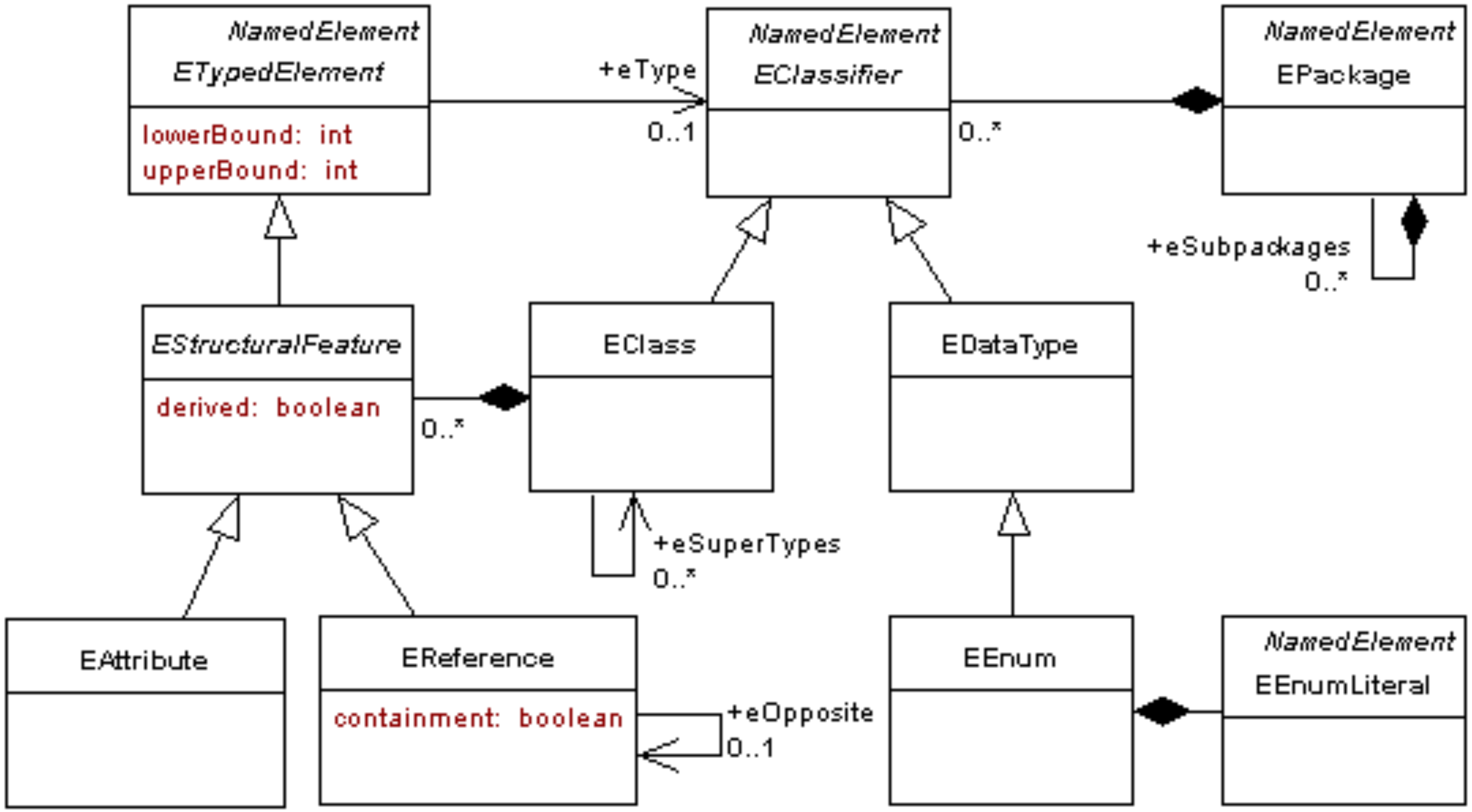
Eclipse Modeling Framework (EMF)

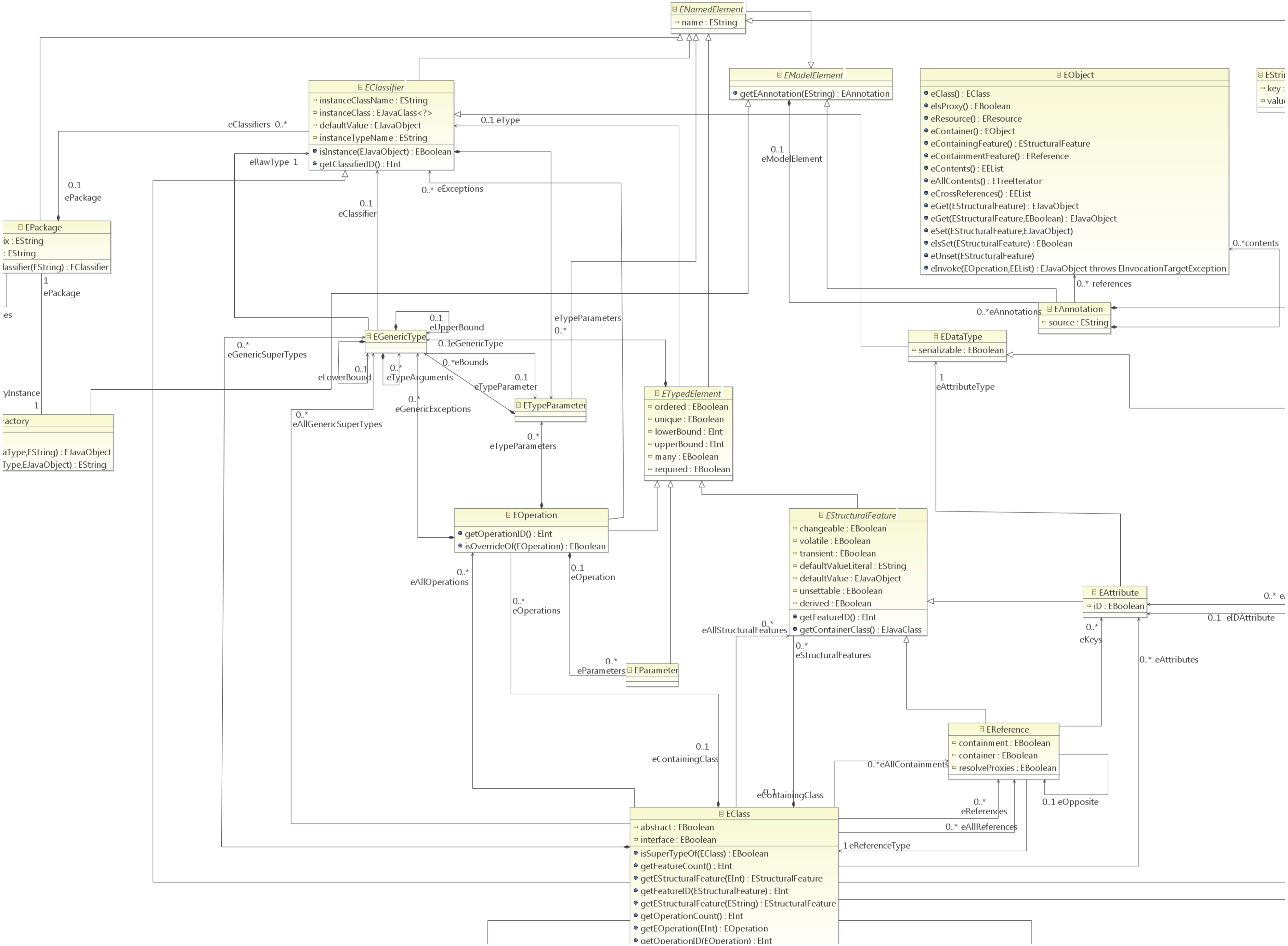
- EMF is a modelling framework for Eclipse, based on EMOF

- From a data model specification described in XMI, it produces Java implementation classes for this model, and a basic tree-like editor.

- Web page: http://www.eclipse.org/modeling/emf
- Tutorials: http://www.eclipse.org/modeling/emf/docs/
- Tutorial: http://www.vogella.com/tutorials/EclipseEMF/article.html
- How to install EMF in Eclipse:
  - *Help / Install New Software*
  - *Work with*: 2018-19 – http://download.eclipse.org/releases/2018-09
  - Open "Modelling"
  - Select Ecore Diagram Editor (SDK)
EMF – Eclipse Modeling Framework (SDK)
OCL Examples and Editors SDK

EMF meta-meta-model (Ecore)




ENamedElement
name : EString
EClassifier
instanceClassName : EString
instanceClass : EJavaClass<?>
defaultValue : EJavaObject
instanceTypeName : EString
isInstance(EJavaObject) : EBoolean
getClassifierID() : EInt
EModelElement
getEAnnotation(EString) : EAnnotation
EObject
eClass() : EClass
eIsProxy() : EBoolean
eResource() : EResource
eContainer() : EObject
eContainingFeature() : EStructuralFeature
eContainmentFeature() : EReference
eContents() : EEList
eAllContents() : ETreeIterator
eCrossReferences() : EEList
eGet(EStructuralFeature) : EJavaObject
eGet(EStructuralFeature,EBoolean) : EJavaObject
eSet(EStructuralFeature,EJavaObject)
eIsSet(EStructuralFeature) : EBoolean
eUnset(EStructuralFeature)
eInvoke(EOperation,EEList) : EJavaObject throws EInvocationTargetException
EPackage
ix : EString
: EString
lassifier(EString) : EClassifier
Factory
aType,EString) : EJavaObject
Type,EJavaObject) : EString
EGenericType
ETypeParameter
ETypedElement
ordered : EBoolean
unique : EBoolean
lowerBound : EInt
upperBound : EInt
many : EBoolean
required : EBoolean
EAnnotation
source : EString
EDataType
serializable : EBoolean
EOperation
getOperationID() : EInt
isOverrideOf(EOperation) : EBoolean
EStructuralFeature
changeable : EBoolean
volatile : EBoolean
transient : EBoolean
defaultValueLiteral : EString
defaultValue : EJavaObject
unsettable : EBoolean
derived : EBoolean
getFeatureID() : EInt
getContainerClass() : EJavaClass
EAttribute
iD : EBoolean
EParameter
EReference
containment : EBoolean
container : EBoolean
resolveProxies : EBoolean
EClass
abstract : EBoolean
interface : EBoolean
isSuperTypeOf(EClass) : EBoolean
getFeatureCount() : EInt
getEStructuralFeature(EInt) : EStructuralFeature
getFeatureID(EStructuralFeature) : EInt
getEStructuralFeature(EString) : EStructuralFeature
getOperationCount() : EInt
getEOperation(EInt) : EOperation
getOperationID(EOperation) : EInt
eClassifiers 0..*
0..1 eType
eRawType 1
0..1 ePackage
1 ePackage
0..* eExceptions
0..1 eClassifier
0..1 eModelElement
0..*contents
0..* references
0..*eAnnotations
0..1 eUpperBound
0..1 eLowerBound
0..1eGenericType
0..*eBounds
0..* eTypeArguments
0..1 eTypeParameter
eTypeParameters 0..*
0..* eTypeParameters
0..* eGenericSuperTypes
0..* eAllGenericSuperTypes
0..* eGenericExceptions
yInstance 1
1 eAttributeType
0..* eAllOperations
0..1 eOperation
0..* eOperations
0..* eParameters
0..* eAllStructuralFeatures
0..* eStructuralFeatures
0..1 eContainingClass
0..1 eContainingClass
0..* eKeys
0..* eAttributes
0..1 eIDAttribute
0..*eAllContainments
0..* eReferences
0..1 eOpposite
0..* eAllReferences
1 eReferenceType

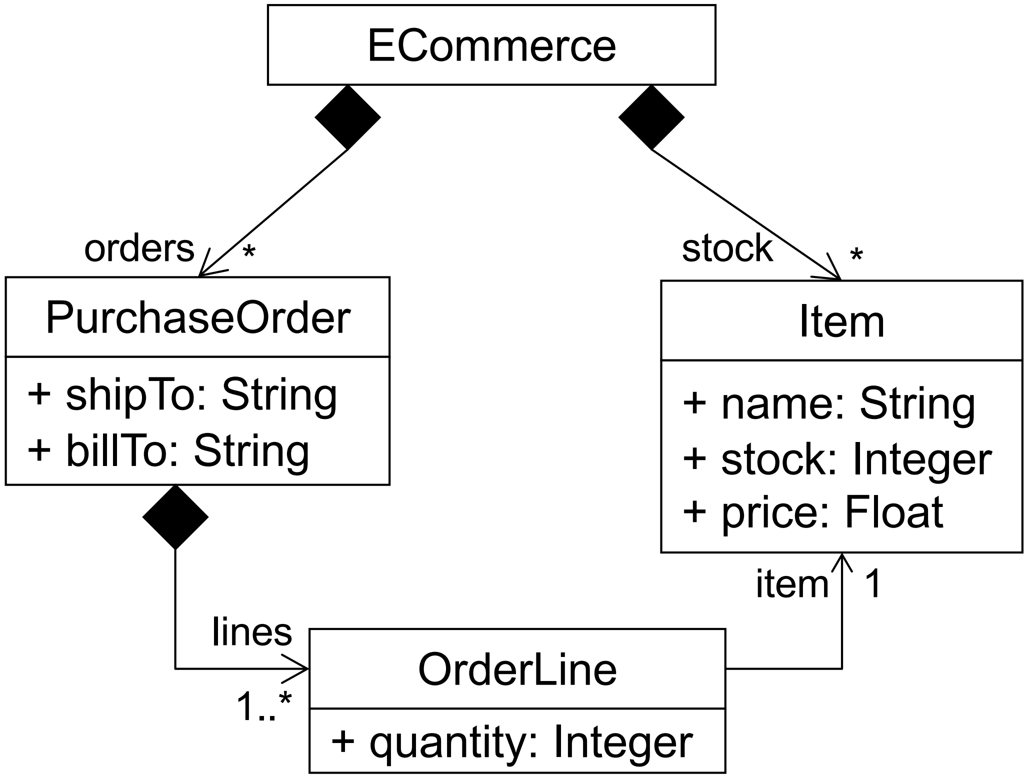
Ecore and Genmodel

- Ecore: allows the definition of meta-models (also called *domain models*). An ecore model contains a root object representing the model, with packages that contain the definition of the following elements:
  - EClass: class, it can have 0 or more attributes and references
  - EAttribute: attribute, it has a name and a type
  - EReference: end of an association between two classes
    - Inverse constraint
  - EDataType: type of an attribute
- Genmodel: contains information for the code generation, like path and file information.

Generation of Java code

- Code generation from ecore and genmodel files:
  - model: interfaces and factories for object creation
  - model.impl: concrete implementation of the interfaces
  - model.util: adapter factory
- Each generated interface has getters/setters methods. Each setter notifies to observers of the model.
- Each generated method is annotated with @generated. Regeneration is possible, but methods annotated with @generated get overwritten.
- Generation of an EMF editor plug-in is also possible.

Meta-model used for the demo



The EMF project for this meta-model is available in moodle.
Download, unzip and import the project into Eclipse.
To import a project into Eclipse: *File / Import / General / Existing projects into workspace*, and then select the project folder.

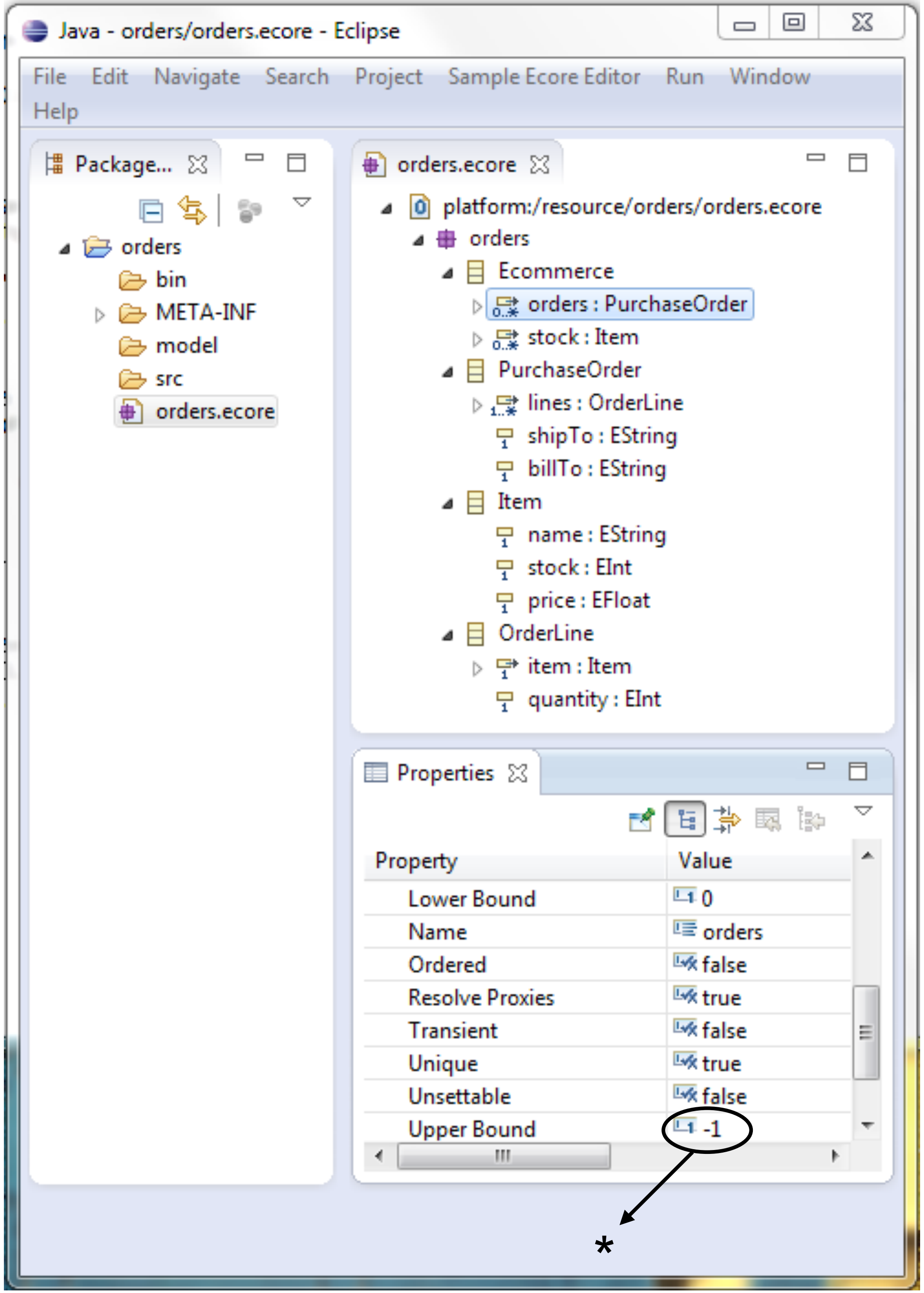
Eclipse Modeling Framework

How to define a DSL and generate code

1. **Create Ecore file (meta-model)**
   - Create a new empty EMF project
     File / New / Other / Eclipse Modeling Framework / Empty EMF Project
   - Create Ecore model
     File / New / Other / Eclipse Modeling Framework / Ecore model
   - Add classes, attributes and references to diagram
2. Create EMF generator model (generator)
3. Generate Java code
4. Generate tree-based model editor

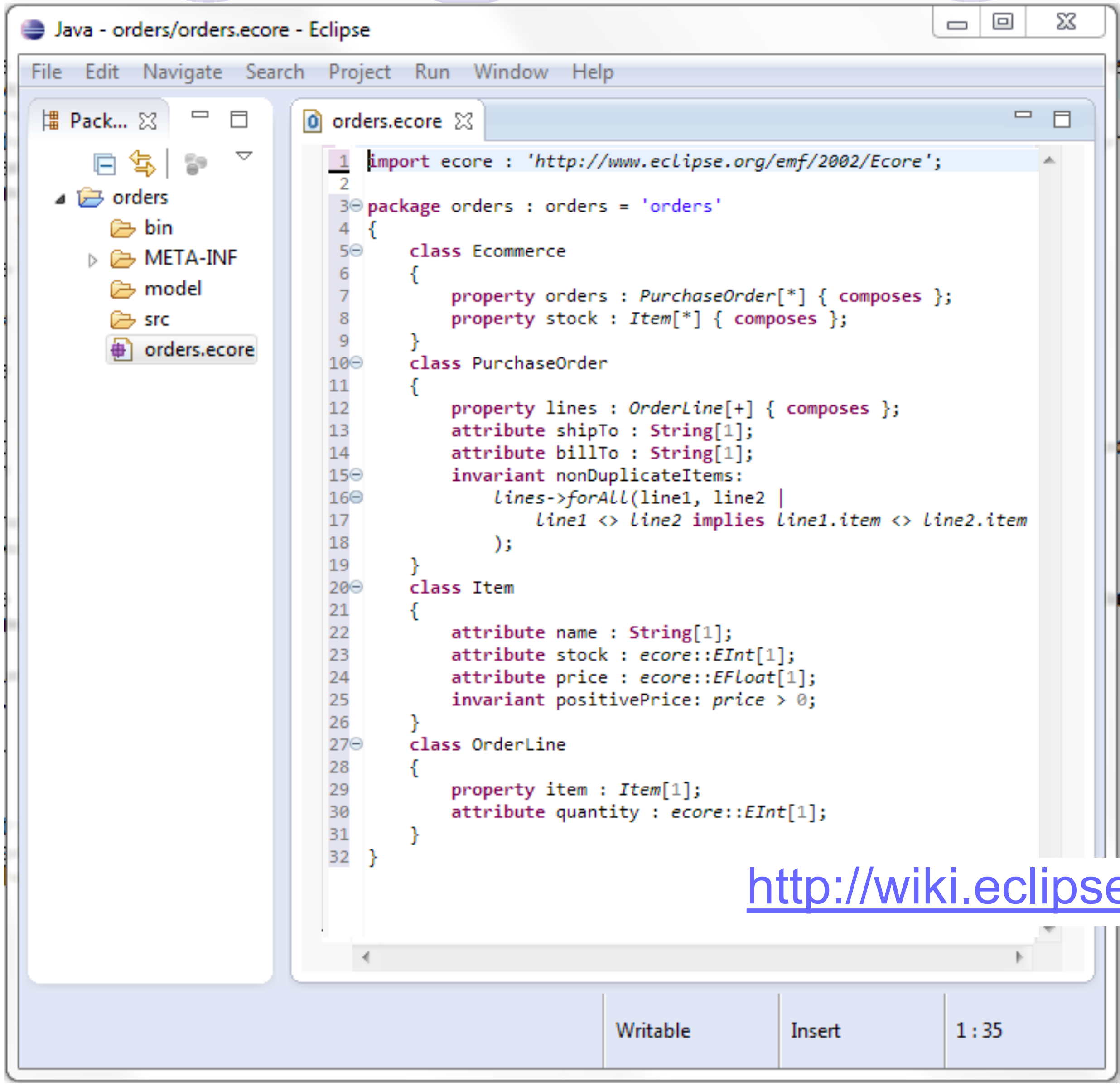
Ecore
Ecore elements

- **EPackage:** one by default
  - *Ns Prefix*
  - *Ns URI*
- **EClass**: classes
  - It defines attributes and references.
  - *ESuperTypes*: parent classes
  - *Abstract*: abstract class
- **EAttribute**: attributes
  - *EType*: type (int, float, ...)
  - *Lower Bound:* minimum cardinality
  - *Upper Bound*: maximum cardinality
- **EReference**: association end
  - *EType*: referenced class
  - *Containment*: containment relation
  - *Lower Bound:* minimum cardinality
  - *Upper Bound*: maximum cardinality
  - *EOpposite*: opposite association end (for bidirectional associations)



Ecore

Create ecore diagram with OCLinEcore



http://wiki.eclipse.org/MDT/OCLinEcore

Eclipse Modeling Framework
How to define a DSL and generate code

1. Create Ecore diagram (meta-model)
2. **Create EMF generator model (generator)**
   - Select ecore file, right mouse-button
     New / Other / Eclipse Modeling Framework / EMF Generator Model
   - If the .ecore file changes, we *reload* the generator model
3. Generate Java code
4. Generate tree-based model editor

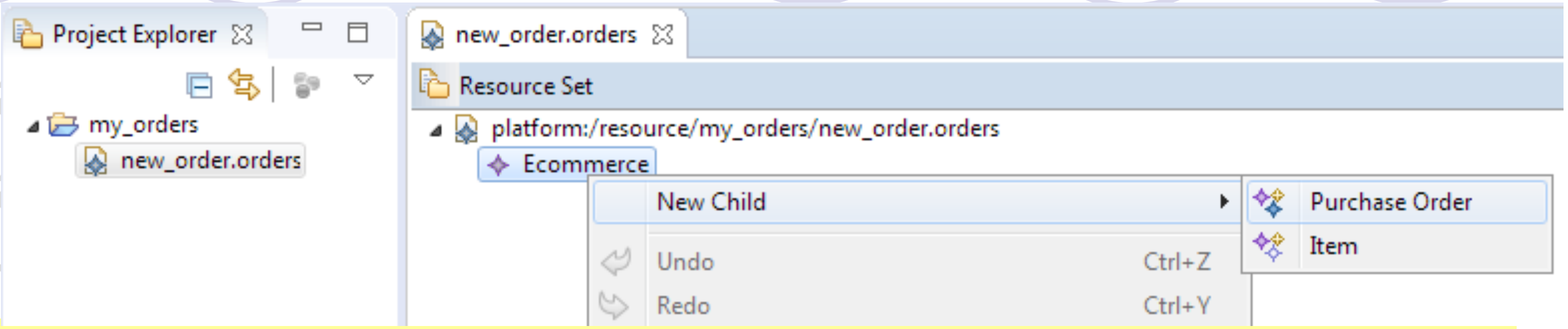
Eclipse Modeling Framework
How to define a DSL and generate code

1. Create Ecore diagram (meta-model)
2. Create EMF generator model (generator)
3. **Generate Java code**
   - Open generator model, right mouse-button in root
     Generate Model Code
4. Generate tree-based model editor

Eclipse Modeling Framework

How to define a DSL and generate code

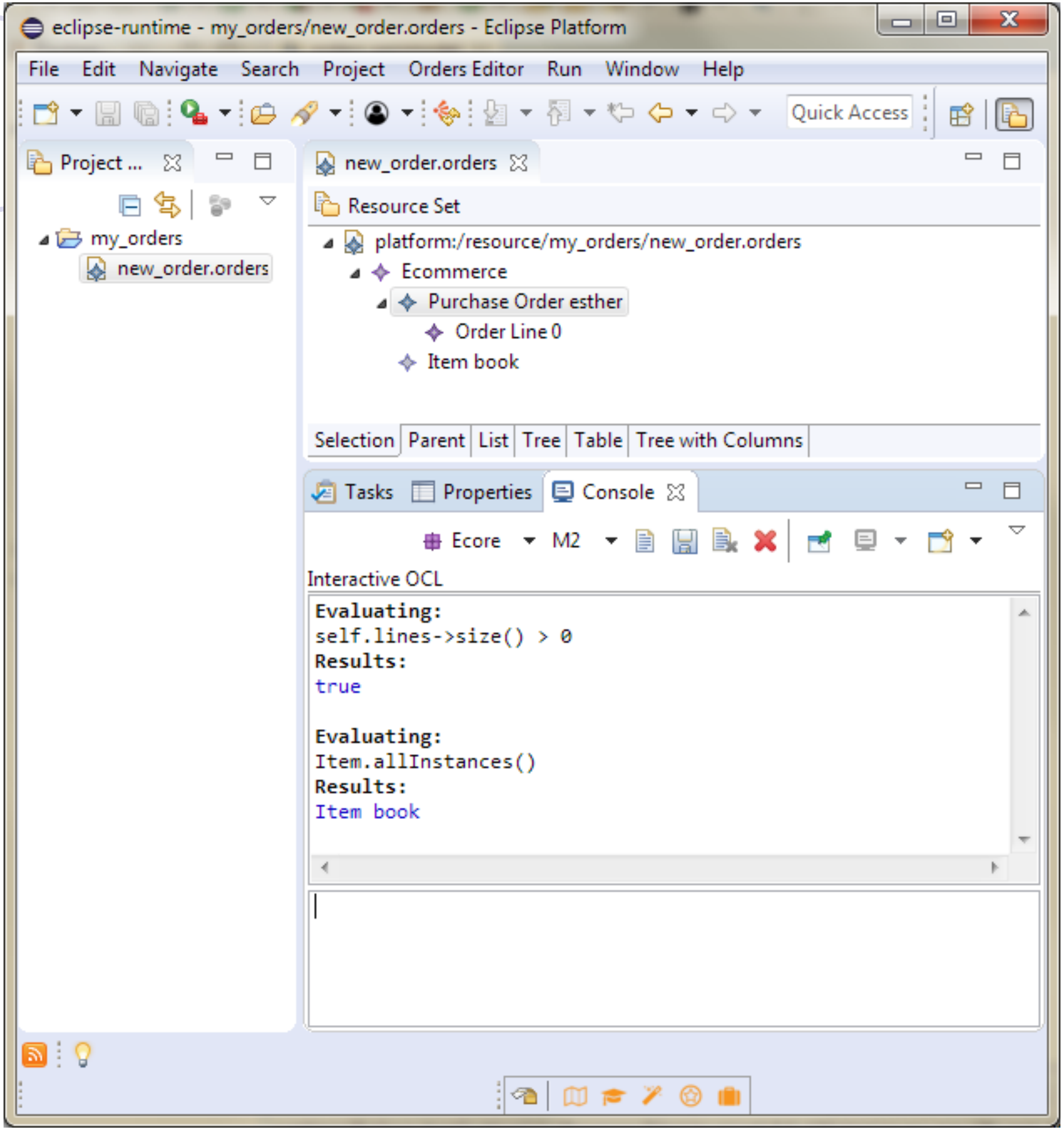
1. Cre
2. Cre
3. Gen



4. **Generate tree-based model editor**
   - Open generator model, right mouse-button in root
     Generate Edit Code, and then *Generate Editor Code*
   - Execute editor
     Run / Run As / Eclipse Application
     (it deploys the generated plug-in in the new instance of Eclipse)
   - Create empty project in new instance of Eclipse
   - Create model
     File / New / Other / Example EMF Model Creation Wizards / new language
     (select root class for the model)

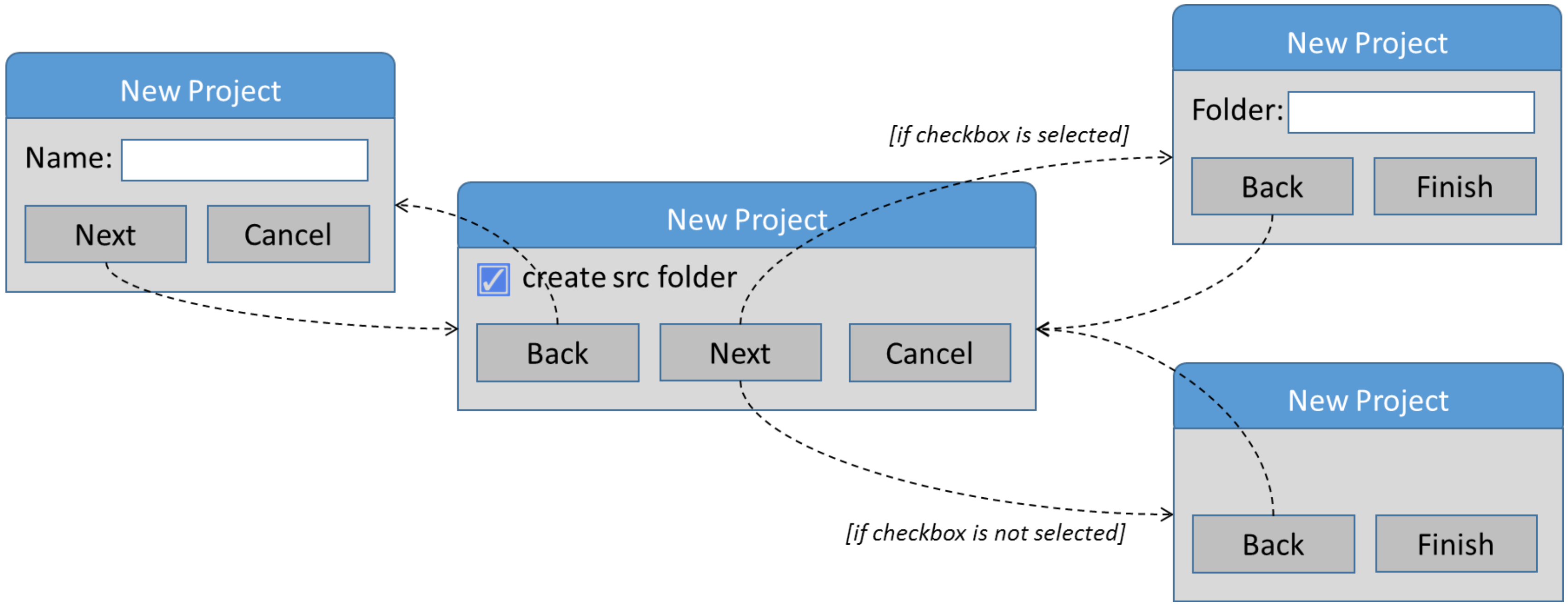
Interactive OCL console

- Right-click on a model element
- Select OCL / Show OCL Console



Exercise

- Build an EMF meta-model for defining wizards (the detailed list of requirements is published in moodle)
- Build the corresponding tree-like editor
- Build a model using the tree-like editor

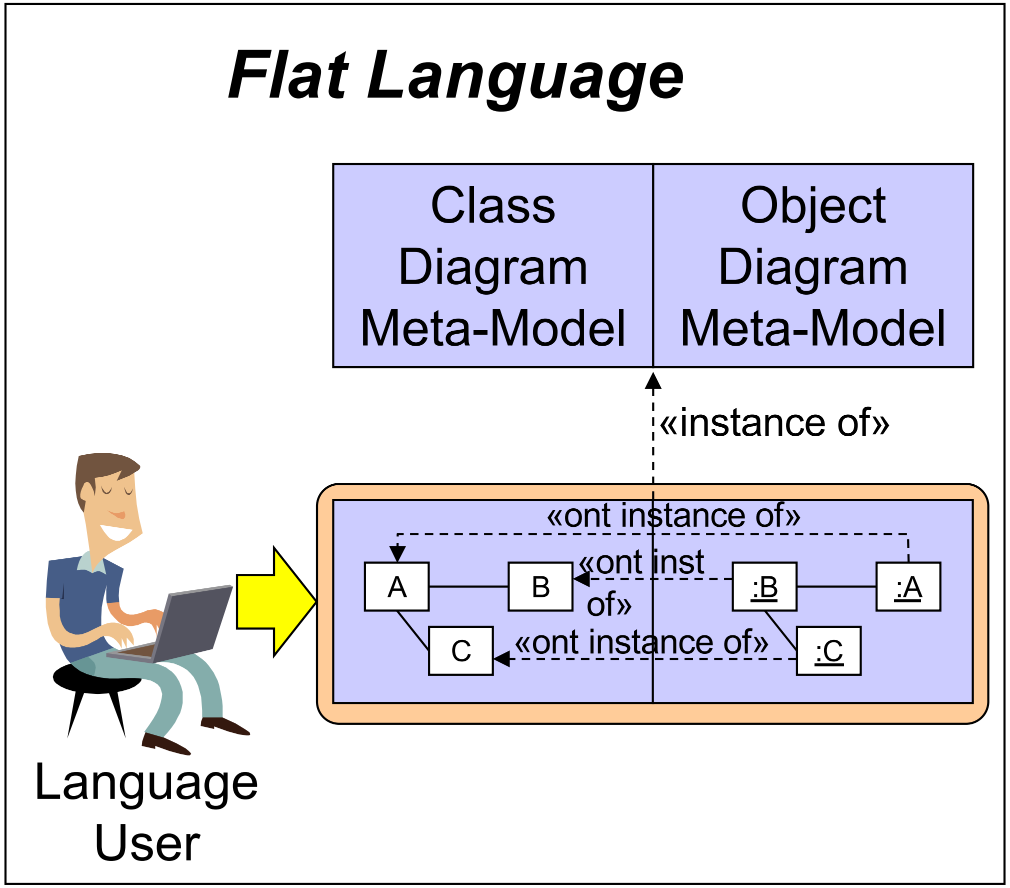


Hand-in: October 19th (together with Xtext editor)

Index

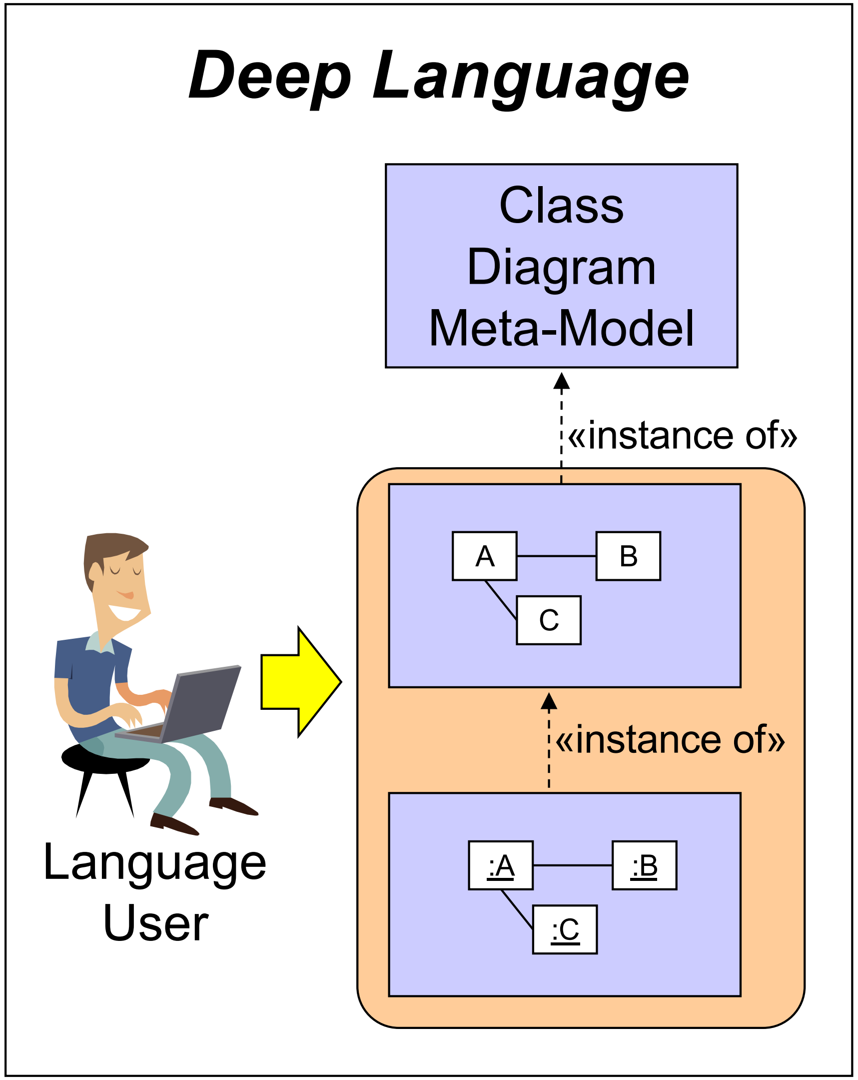
- Introduction
- Two-level meta-modelling
- **Multi-level meta-modelling** 
  - **MetaDepth**
- Profiles
- Graph grammars
- Bibliography

Rearchitecting the UML infrastructure



- In UML, both class and object diagrams are defined with a "big" meta-model.
- Both class and object diagrams are at the same meta-level.

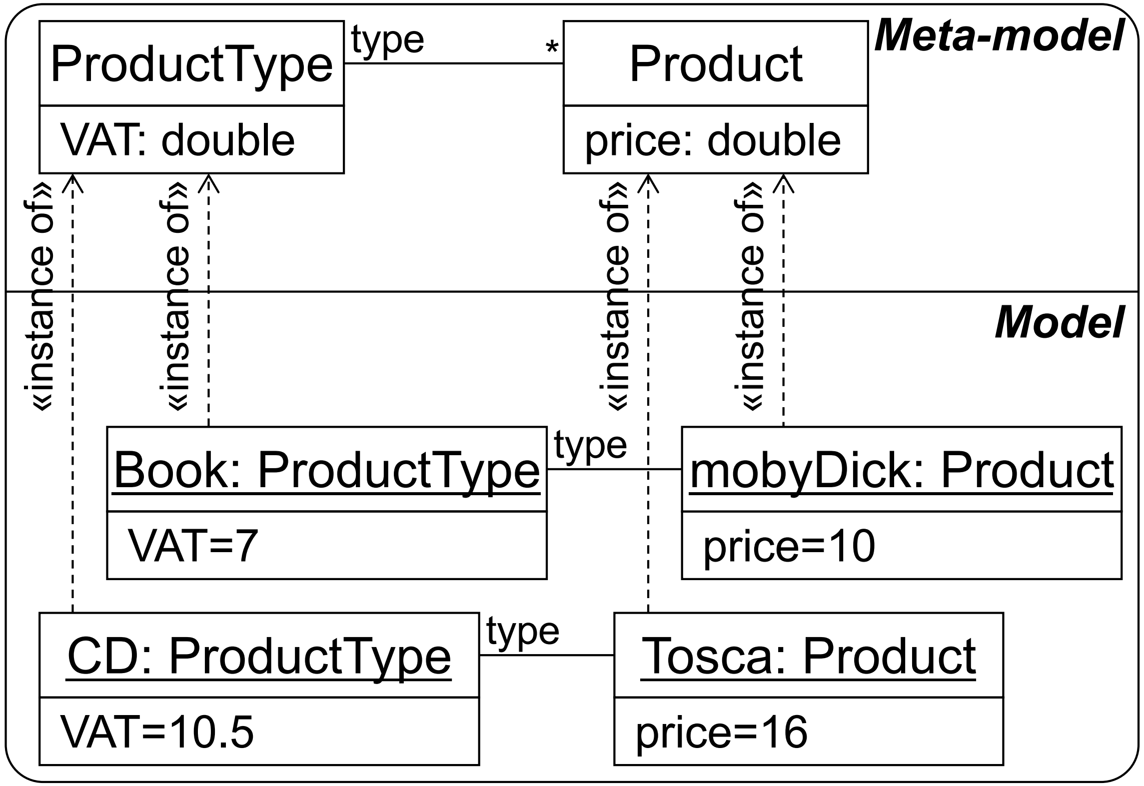
Rearchitecting the UML infrastructure



- We can use multi-level modelling to set object diagrams as instances of class diagrams.
- This way, we only need to define the meta-model of class diagrams at the top-level: object diagrams are instances of the class diagrams.
- We obtain a simpler description of the UML.

Multi-level meta-modelling

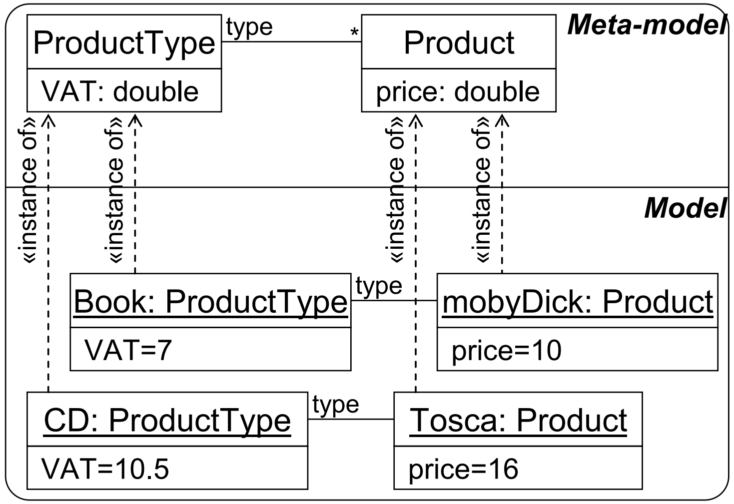
Let's consider the following system.



Typical example of the type-object pattern.

Multi-level meta-modelling

- Disadvantages:
  - The type relation is a kind of <<instance of>> relation, but the architecture does not provide support for it.
  - Manual maintenance of typing relations at the model level.

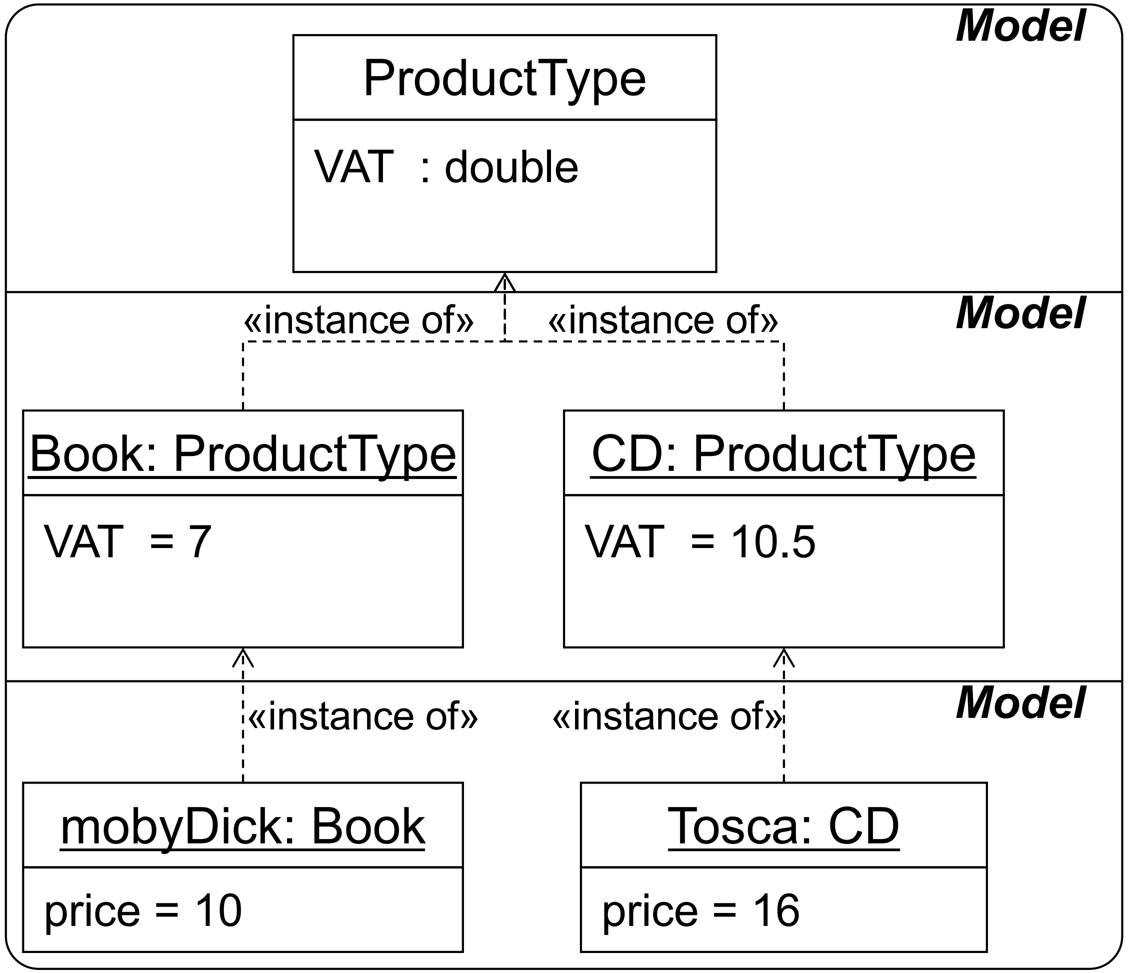


- Are three meta-levels shoehorned in two?

- This is not the only case: UML class/object diagrams, web languages (type nodes/nodes, user types/users, etc).

Multi-level meta-modelling

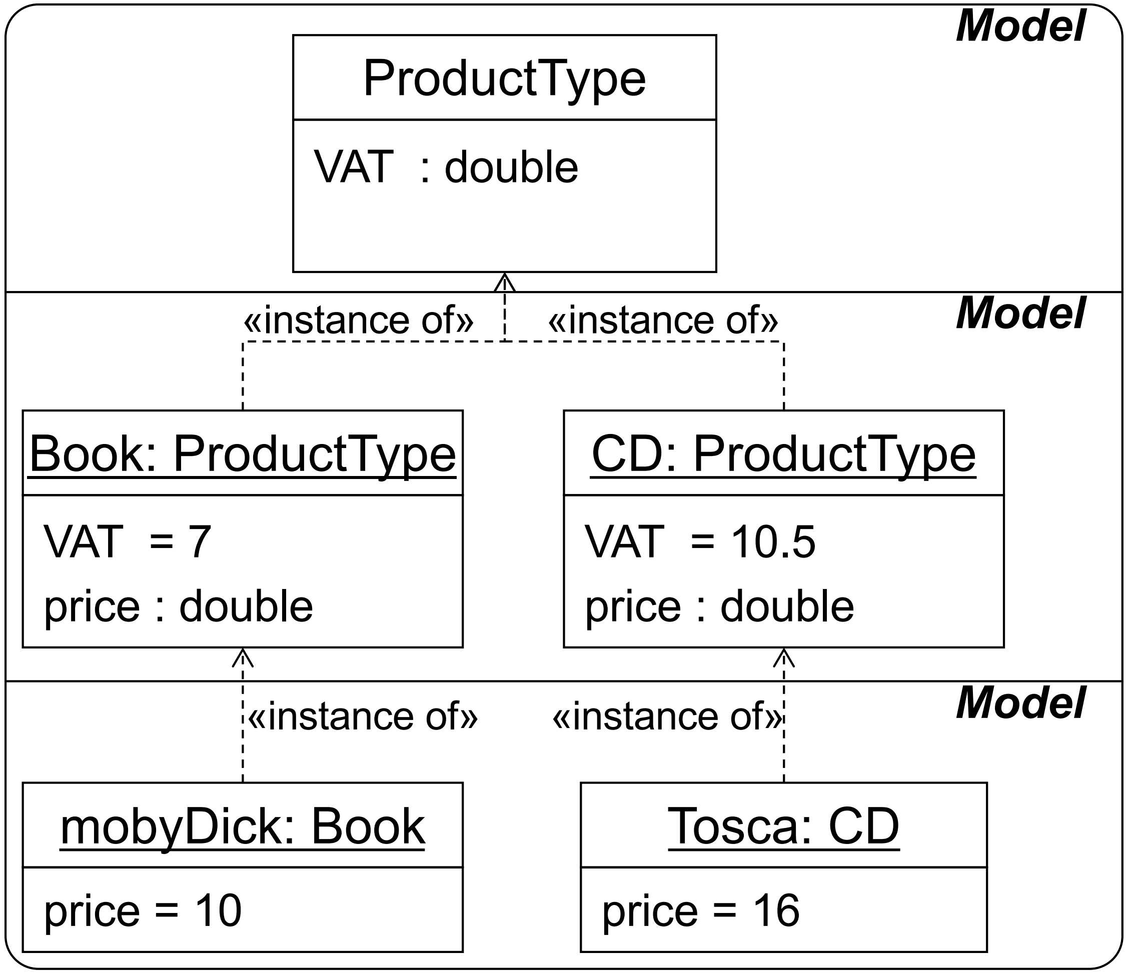
We can organise the system in 3 levels:



Multi-level meta-modelling

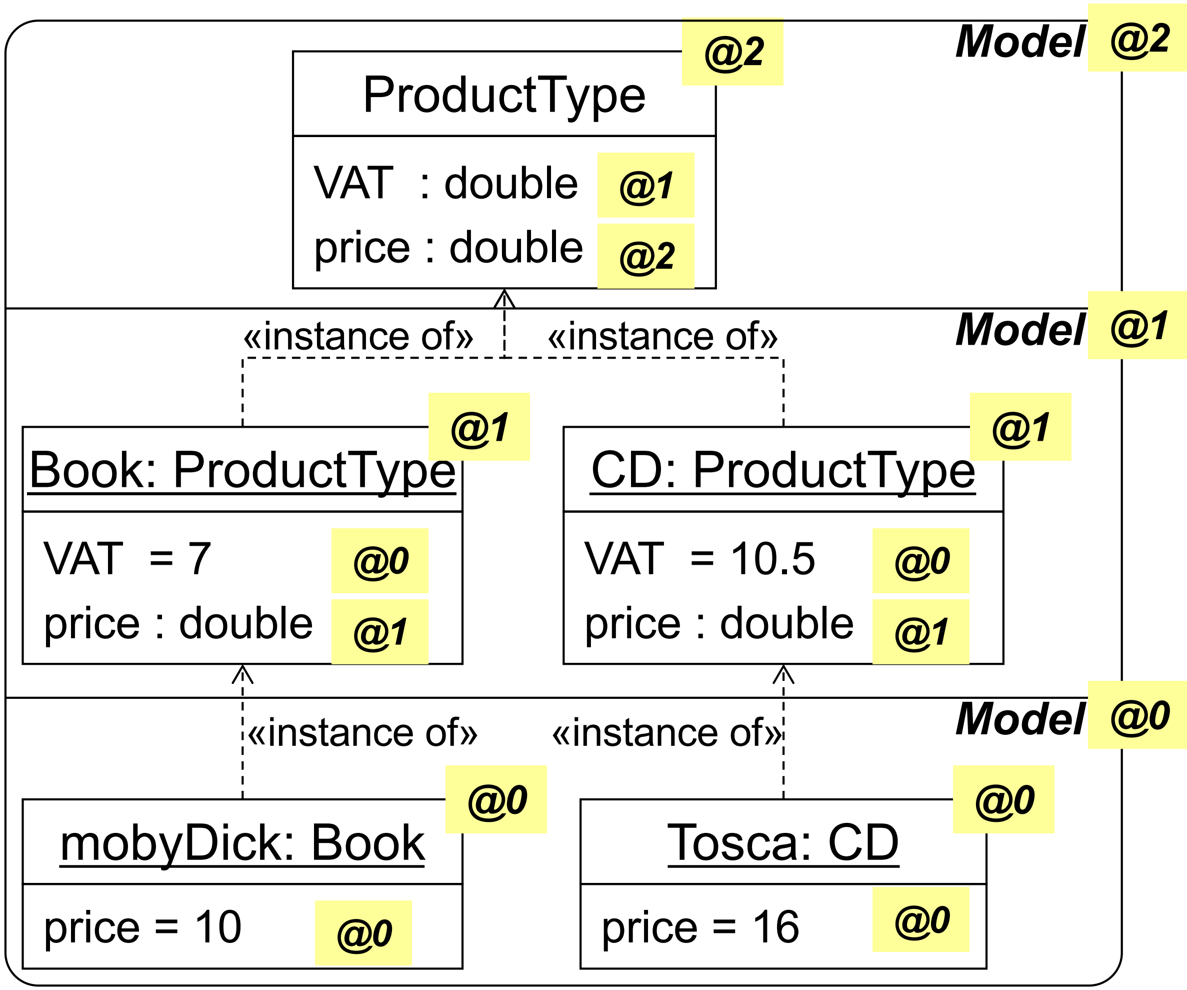
Clabjects

Elements with both type (class) and object (object) facets.



Multi-level meta-modelling

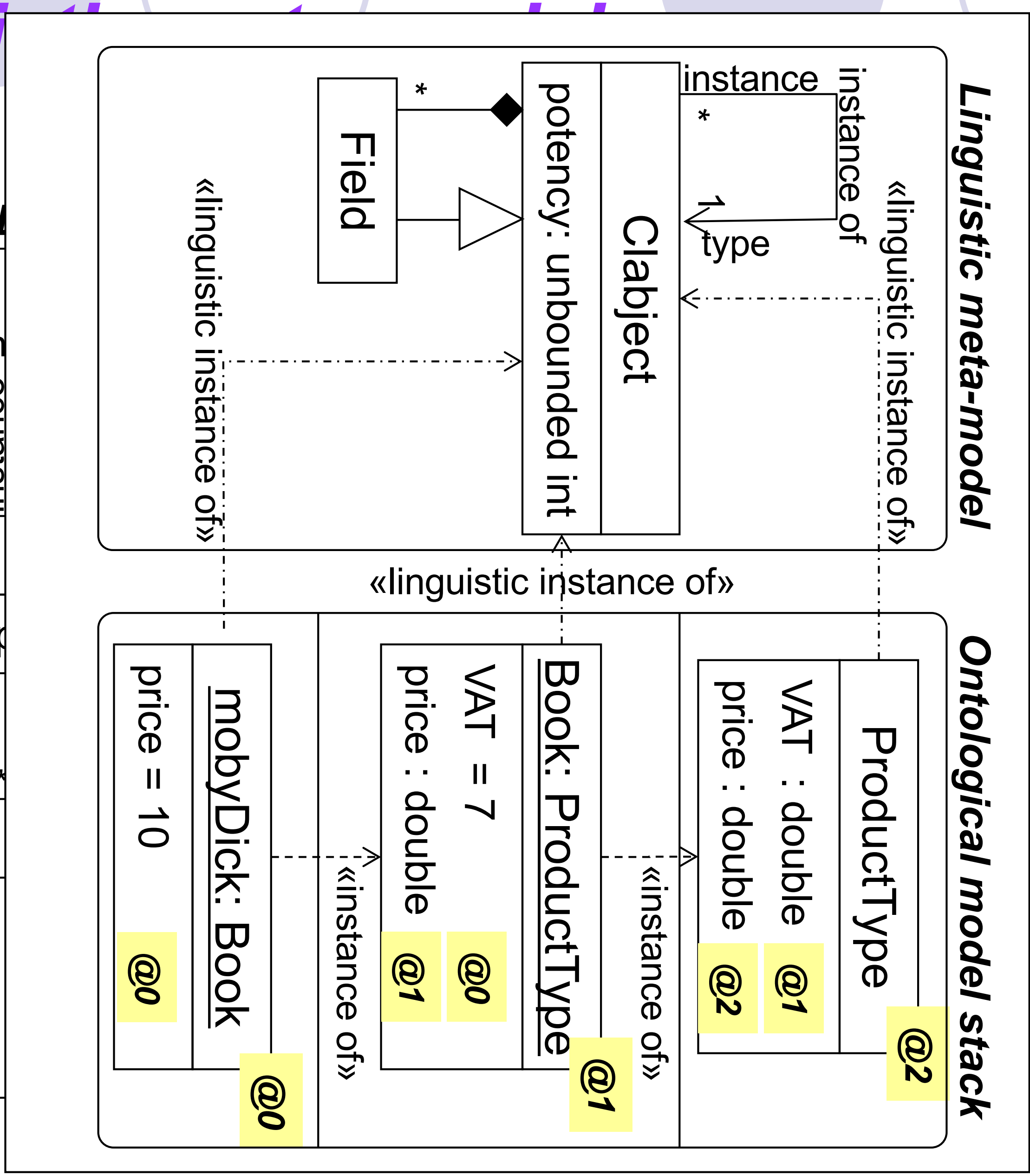
Deep characterisation: potency



Atkinson, C., Kühne, T.: Rearchitecting the UML infrastructure. ACM Trans. Model. Comput. Simul. 12(4), 290–321 (2002)

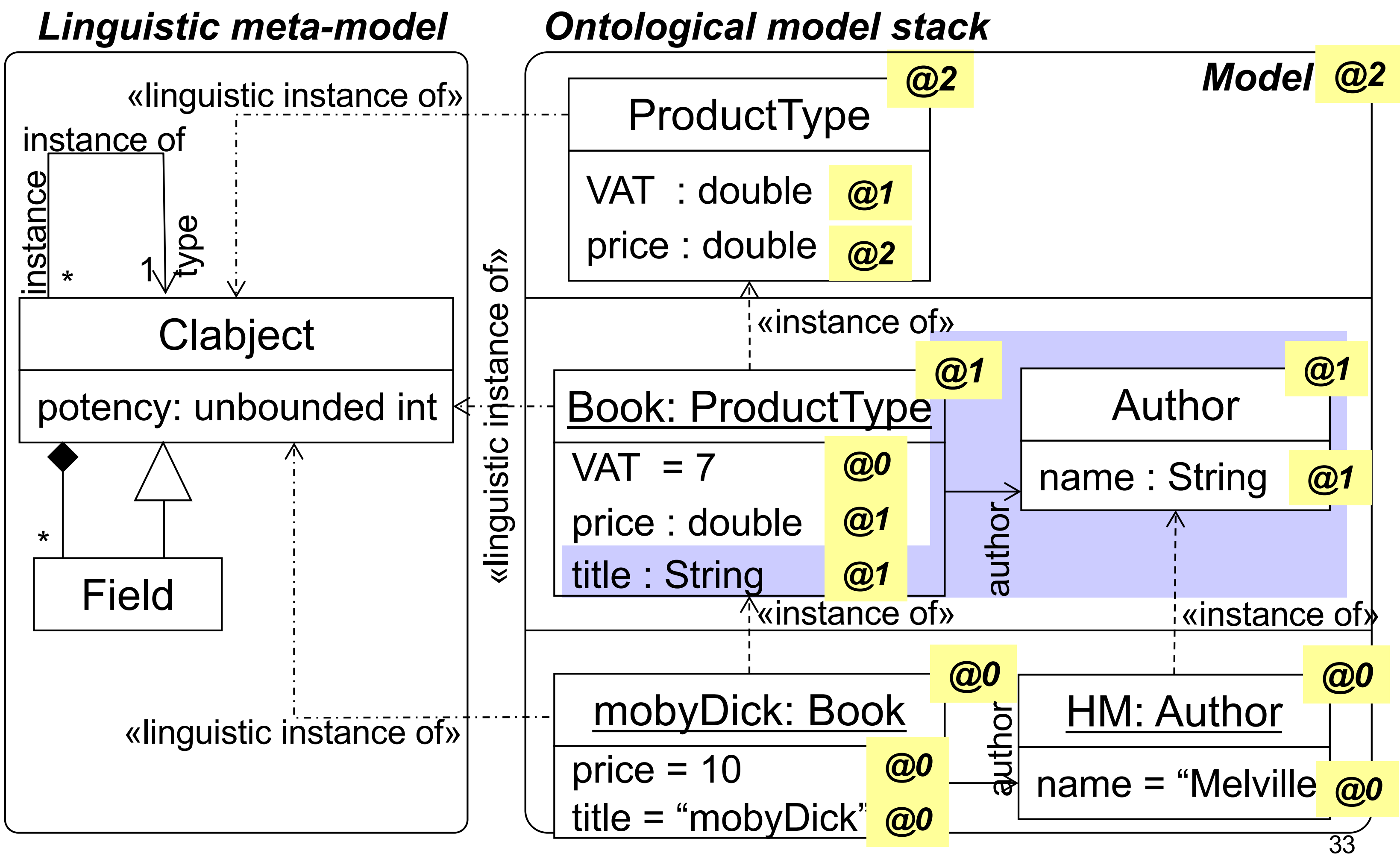
Multi-level meta-modelling

Lingui



Multi-level meta-modelling

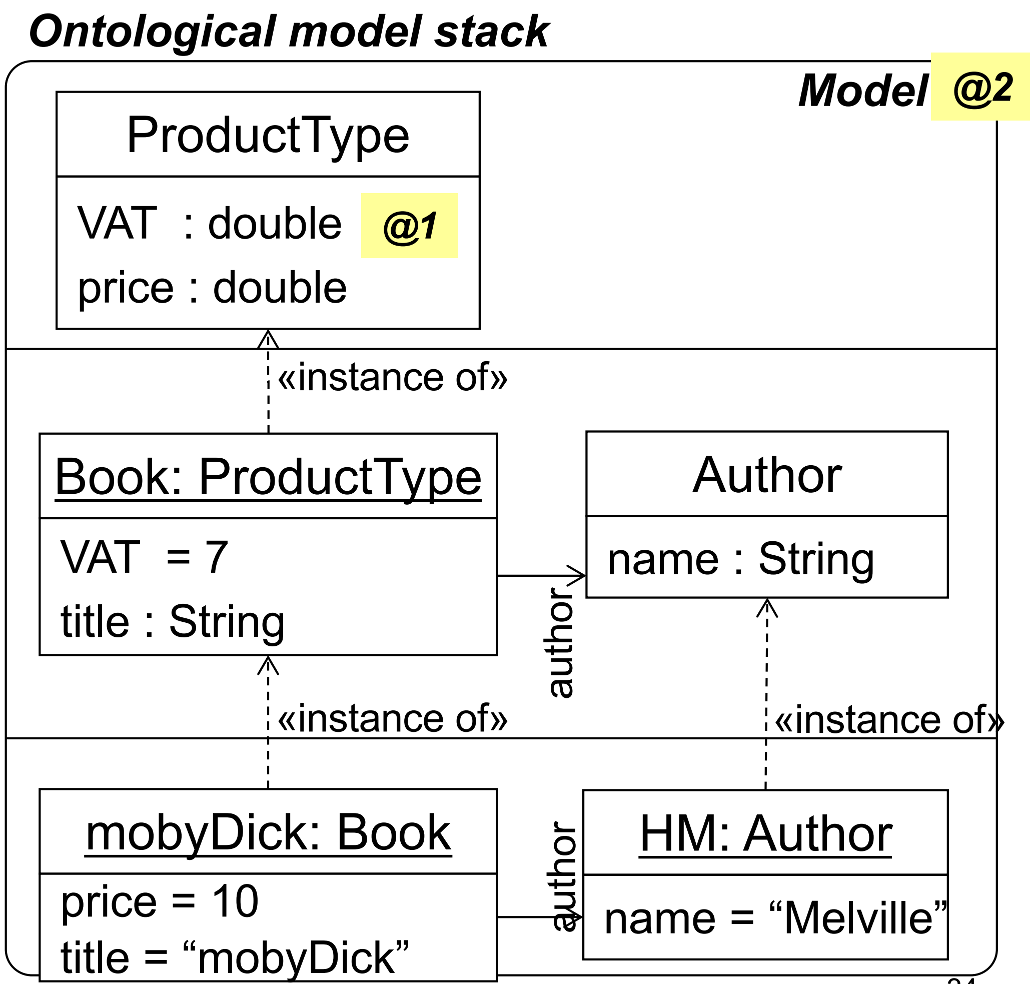
Linguistic extension



Multi-level meta-modelling

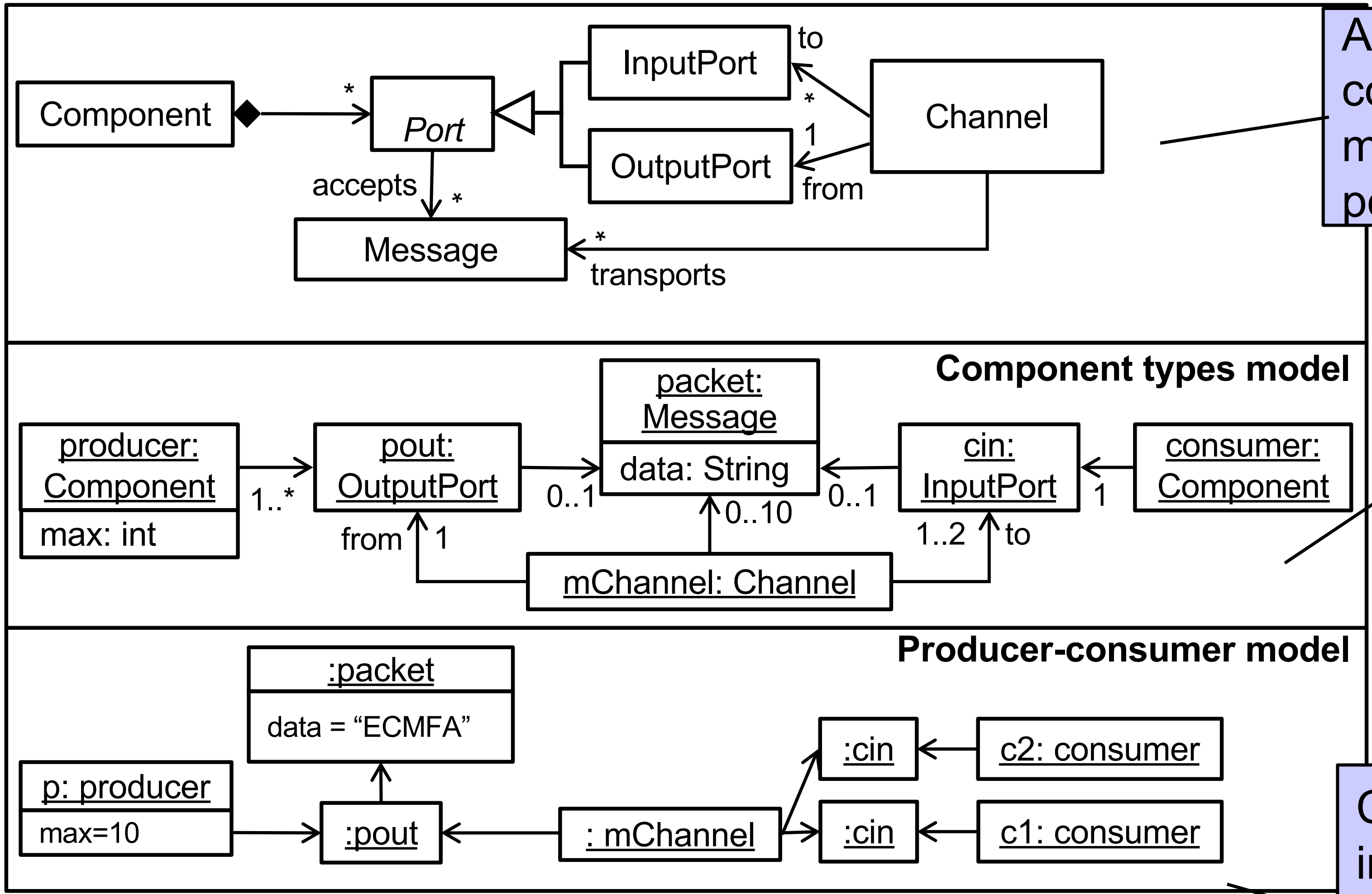
metaDepth: http://metadepth.org/

```
Model Store@2{
  Node ProductType{
    VAT@1 : double = 7.5;
    price : double = 10;
  }
}

Store Library{
  ProductType Book{
    VAT    = 7;
    title  : String;
    author : Author;
  }
  Node Author{
    name : String;
  }
}

Library MyLibrary{
  Book mobyDick{
    price  = 10;
    title = "mobyDick";
    author = HM;
  }
  Author HM{
    name = "Herman Melville";
  }
}
```



Multi-level meta-modelling

Example: component-based modelling



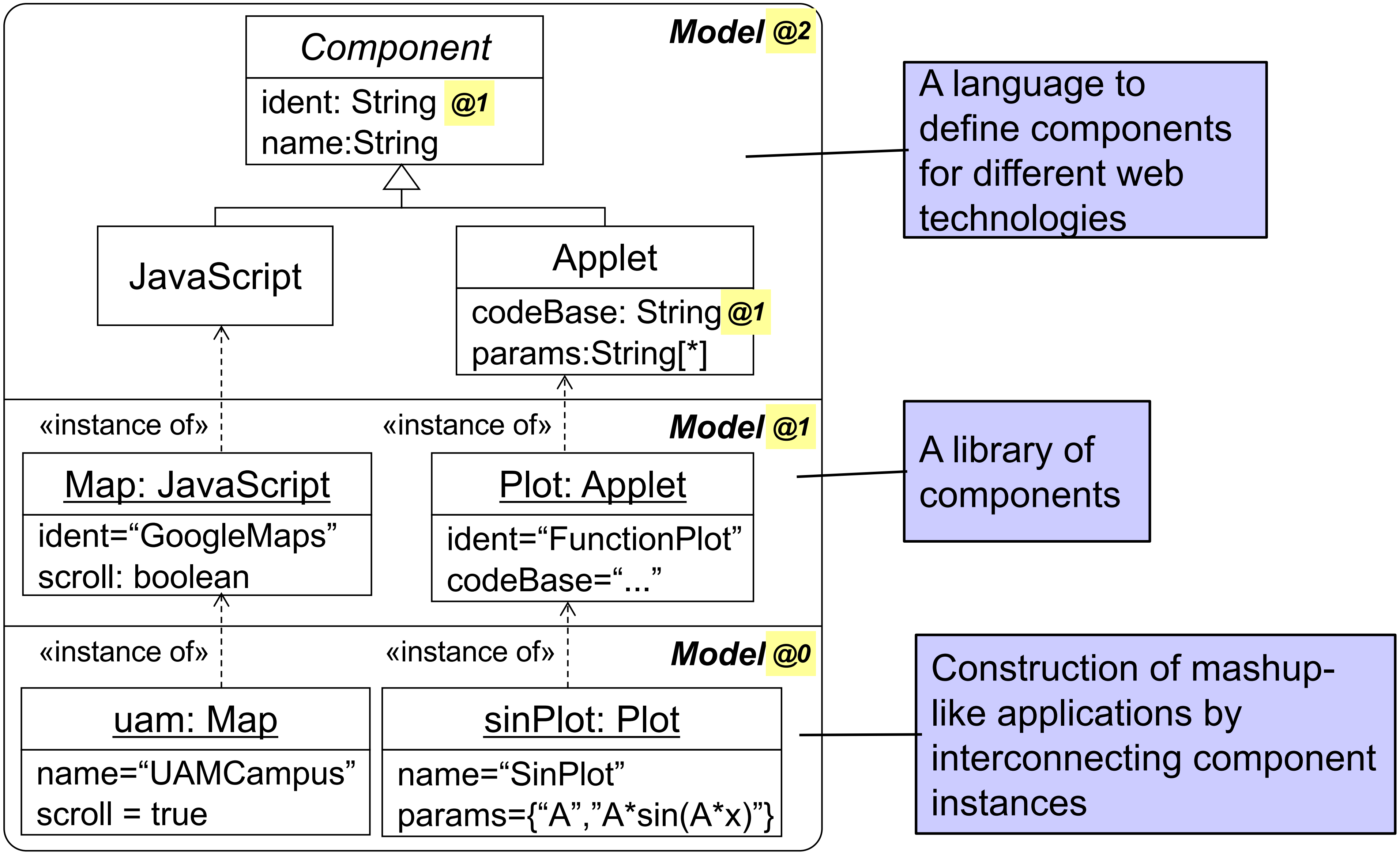
A language to define component types, message types, ports, channels.

Several component and message types can be defined. Ports may have a cardinality.

Components can be instantiated, with the number of ports conforming to the cardinality.

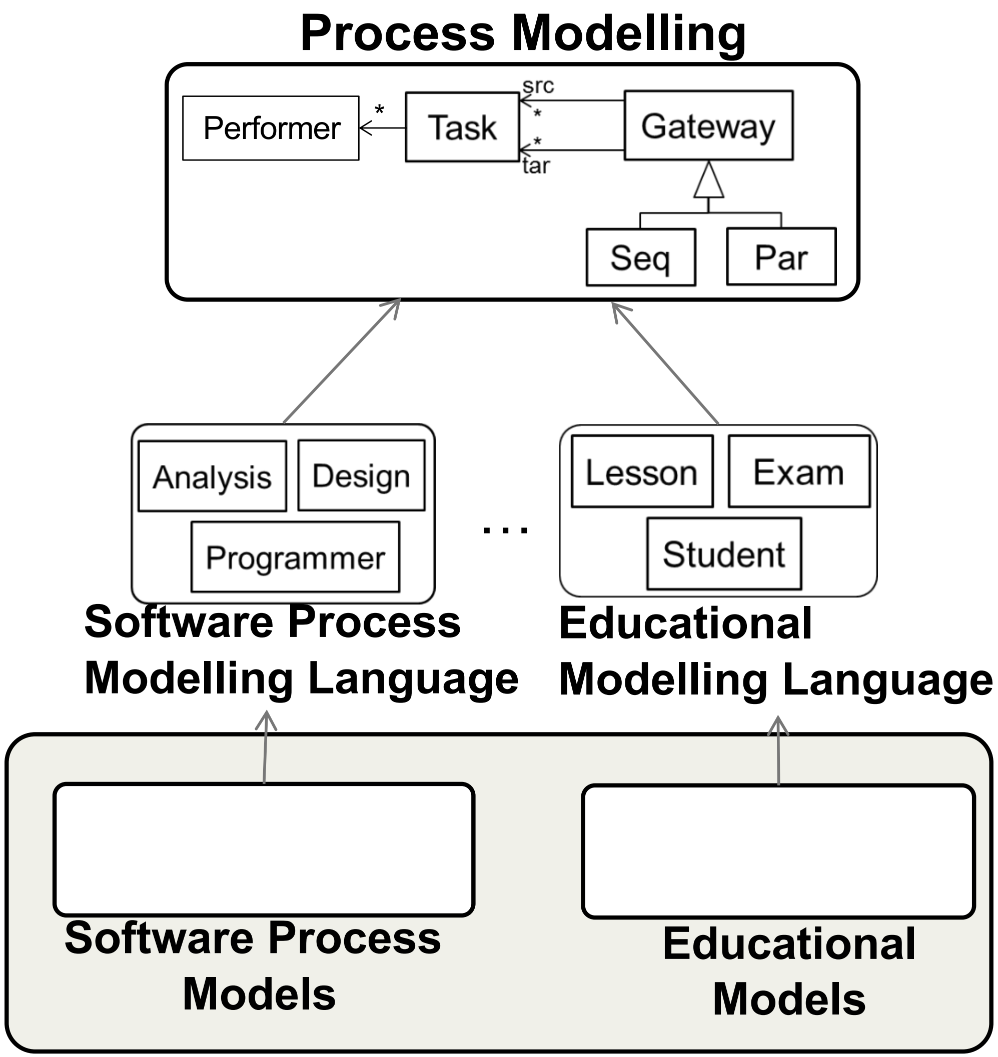
Multi-level meta-modelling

Example: web components



Multi-level meta-modelling

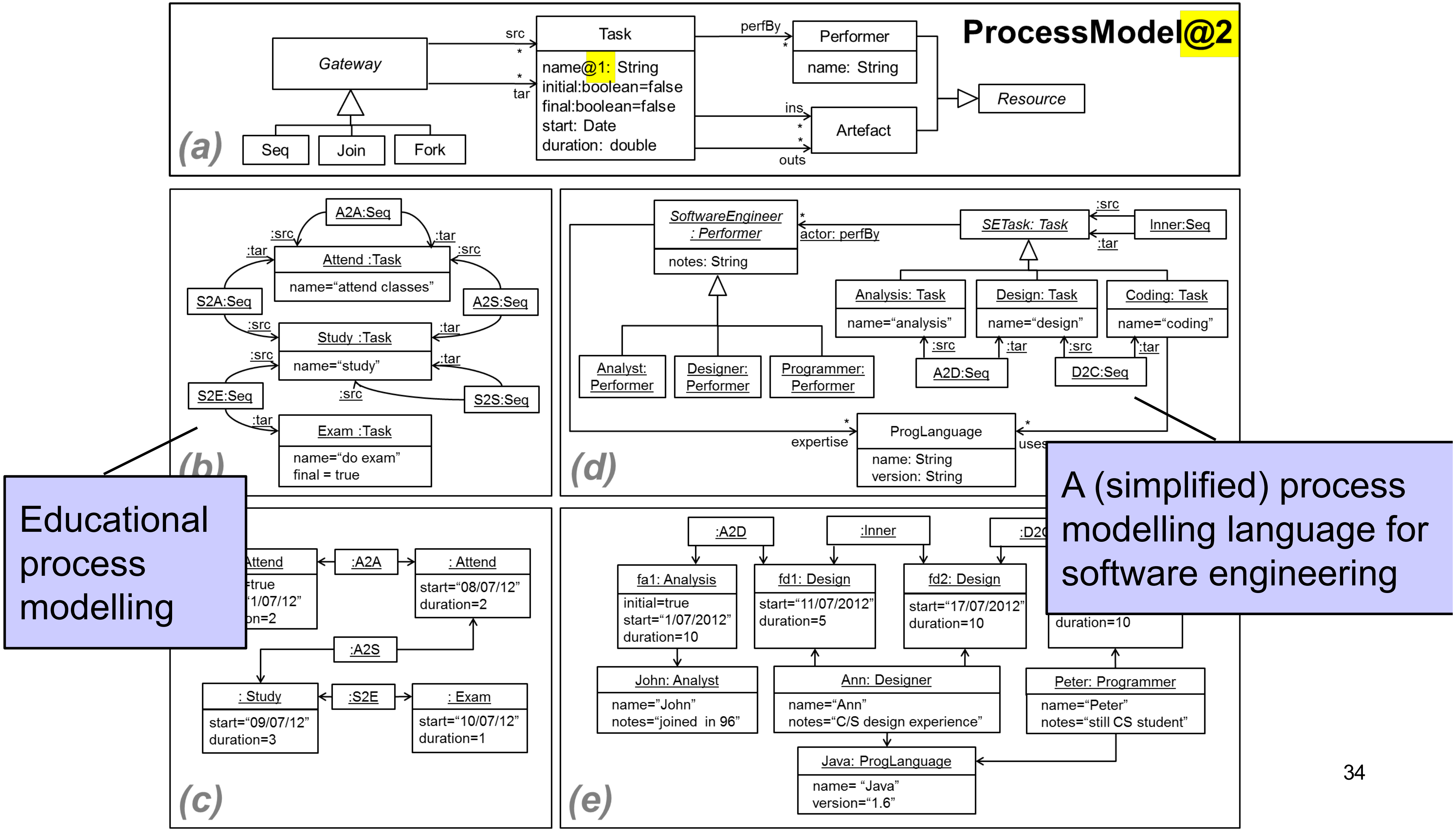
Families of modelling languages



- We can use multi-level modelling to define families of related modelling languages.
- A top-level meta-model for process modelling, which can be used to define specialized languages for process modelling in: software engineering, education, logistics, production engineering, etc.

Multi-level meta-modelling

Families of modelling languages



Index

- Introduction
- Two-level meta-modelling
- Multi-level meta-modelling
- **Profiles** 
- Graph grammars
- Bibliography

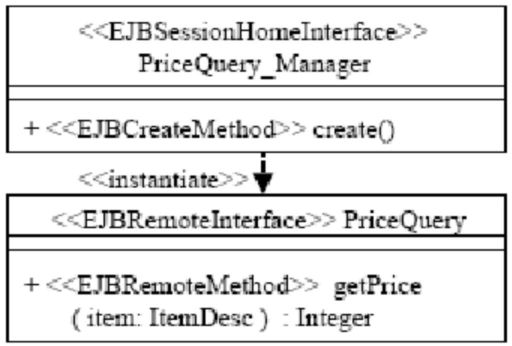
UML Profiles

- **Extension** mechanism to declare new constructions in UML, adapted to a specific domain.

- They define specialised meta-models for a domain, as a subset of the UML meta-model.

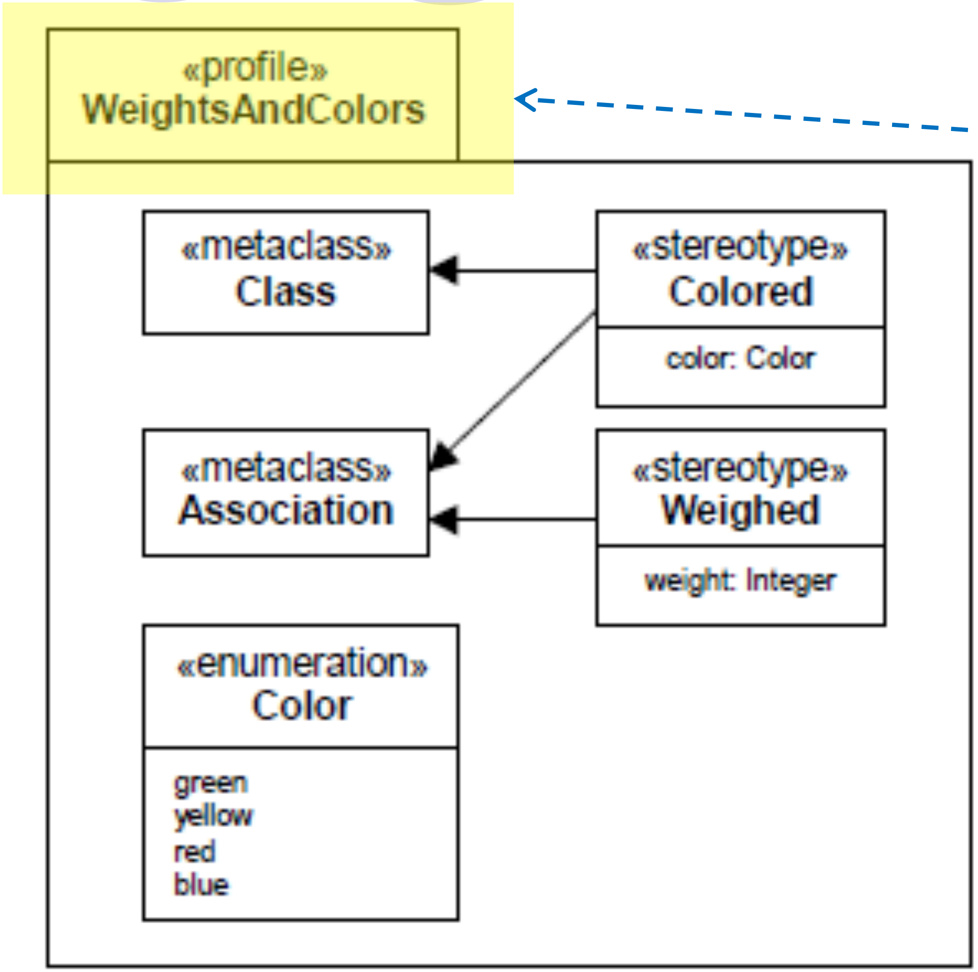
- A profile is defined by means of:
  - stereotypes, e.g. <<JavaClass>> in the EJB profile
  - constraints attached to the stereotype, expressed in OCL
  - tagged values (meta-attributes) attached to the stereotype

- Some examples: EJBs, Web Services, SysML, CORBA, MARTE…



UML Profiles

Defining a profile

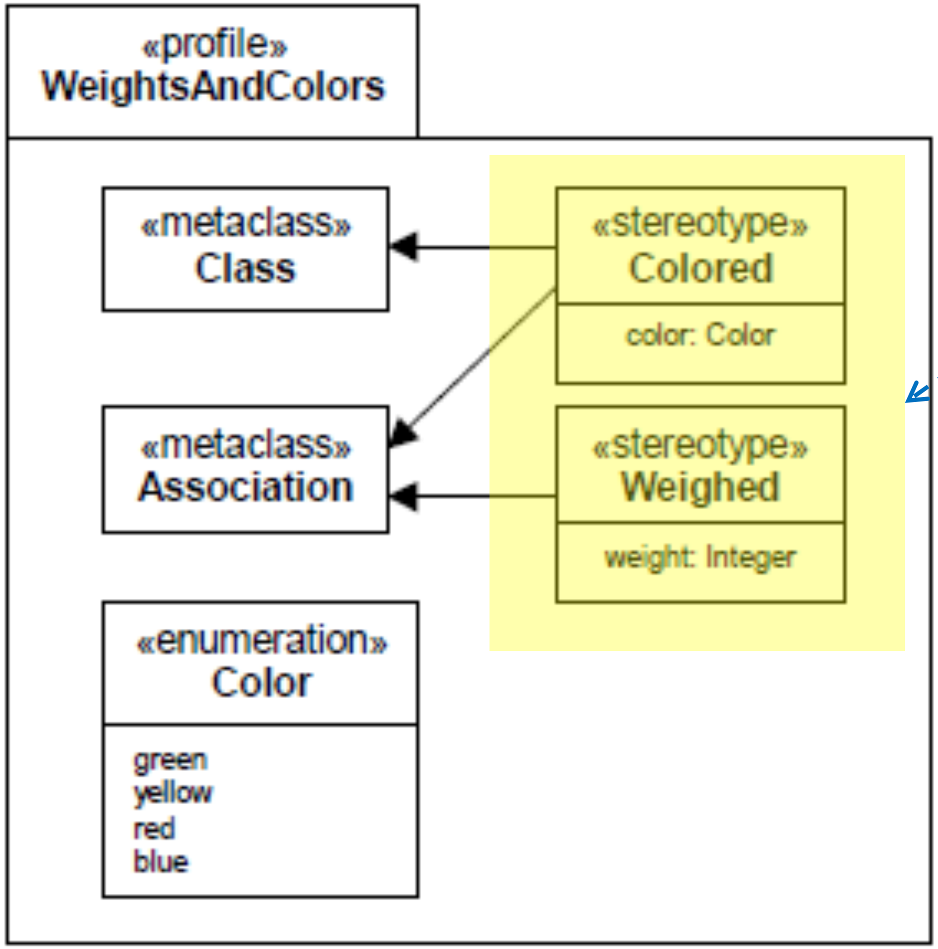


A profile is defined in a UML package with the stereotype <<profile>>.

context UML::InfrastructureLibrary::Core::Constructs::Association
inv : self.isStereotyped("Colored") **implies**
self.connection->**forAll**(isStereotyped("Colored") **implies** color=self.color)

UML Profiles

Defining a profile



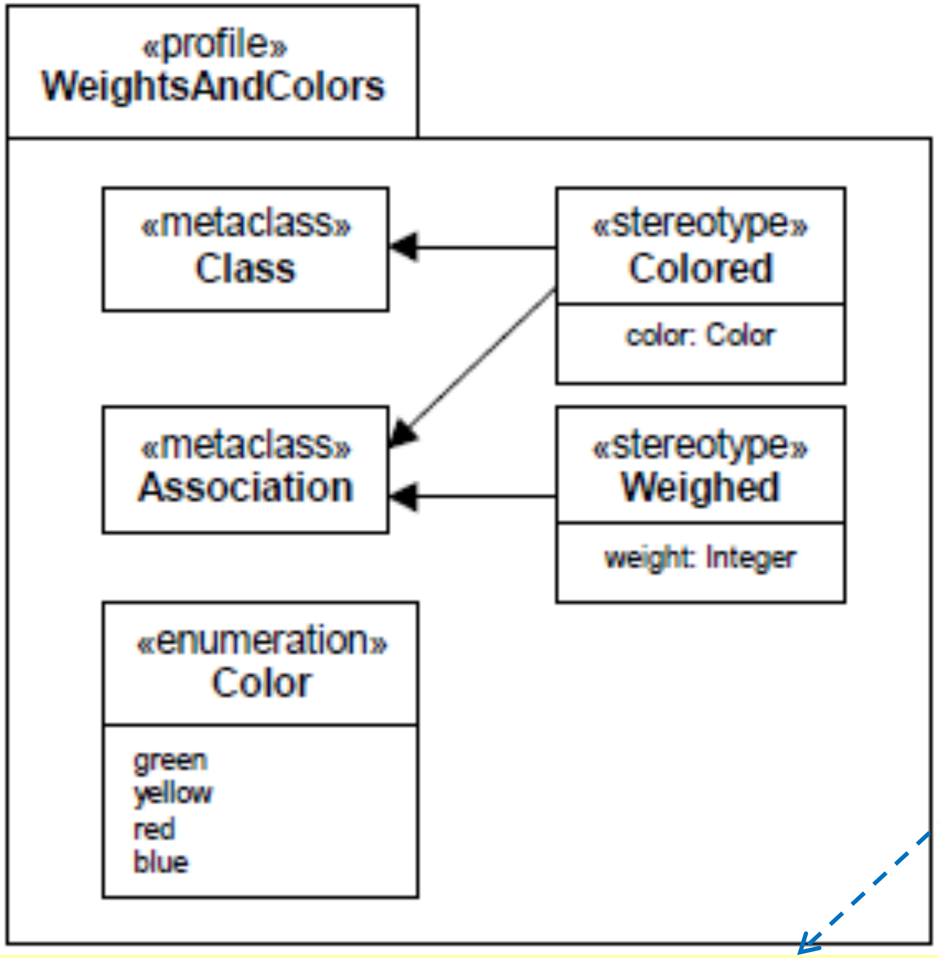
Stereotypes represent domain concepts.

They have a name, and point to the UML meta-model elements for which we are defining the stereotype.

context UML::InfrastructureLibrary::Core::Constructs::Association
inv : self.isStereotyped("Colored") **implies**
self.connection->**forAll**(isStereotyped("Colored") **implies** color=self.color)

UML Profiles

Defining a profile

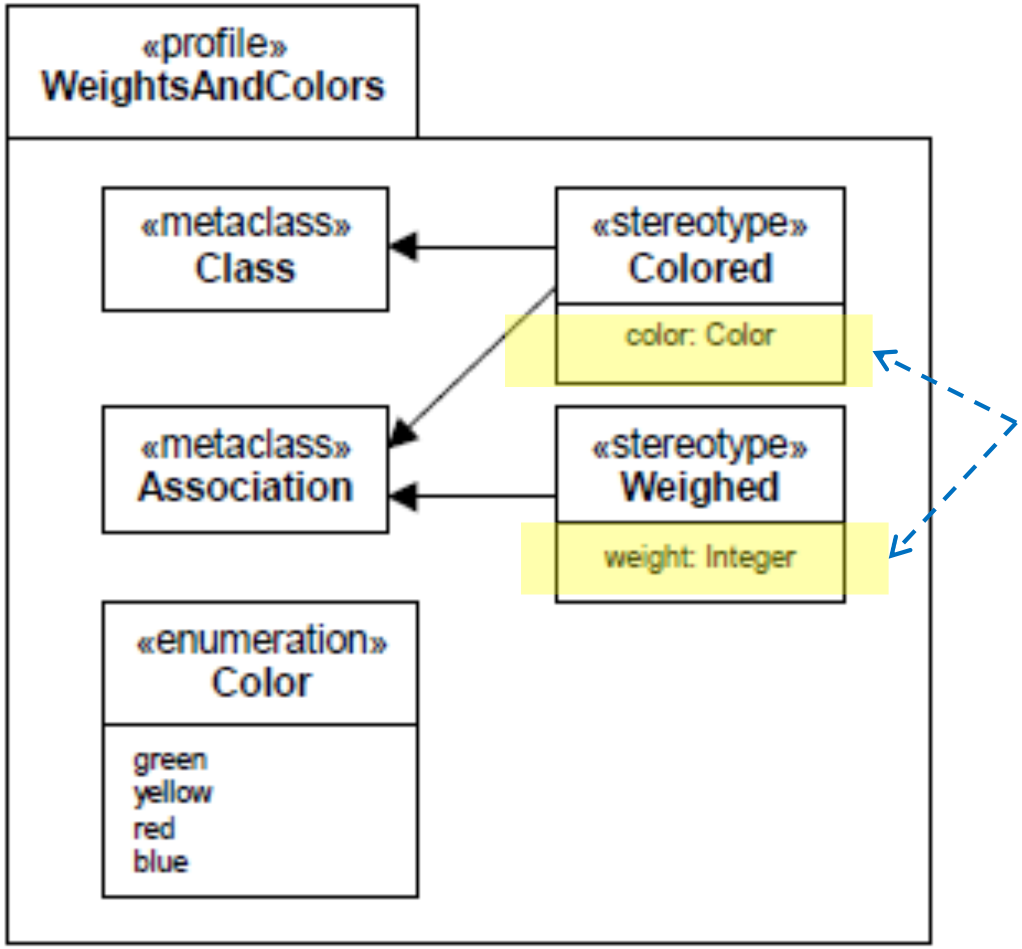


We can define constraints for the stereotyped elements, either in OCL or in natural language.

context UML::InfrastructureLibrary::Core::Constructs::Association
inv : self.isStereotyped(“Colored”) **implies**
self.connection->**forAll**(isStereotyped(“Colored”) **implies** color=self.color)

UML Profiles

Defining a profile



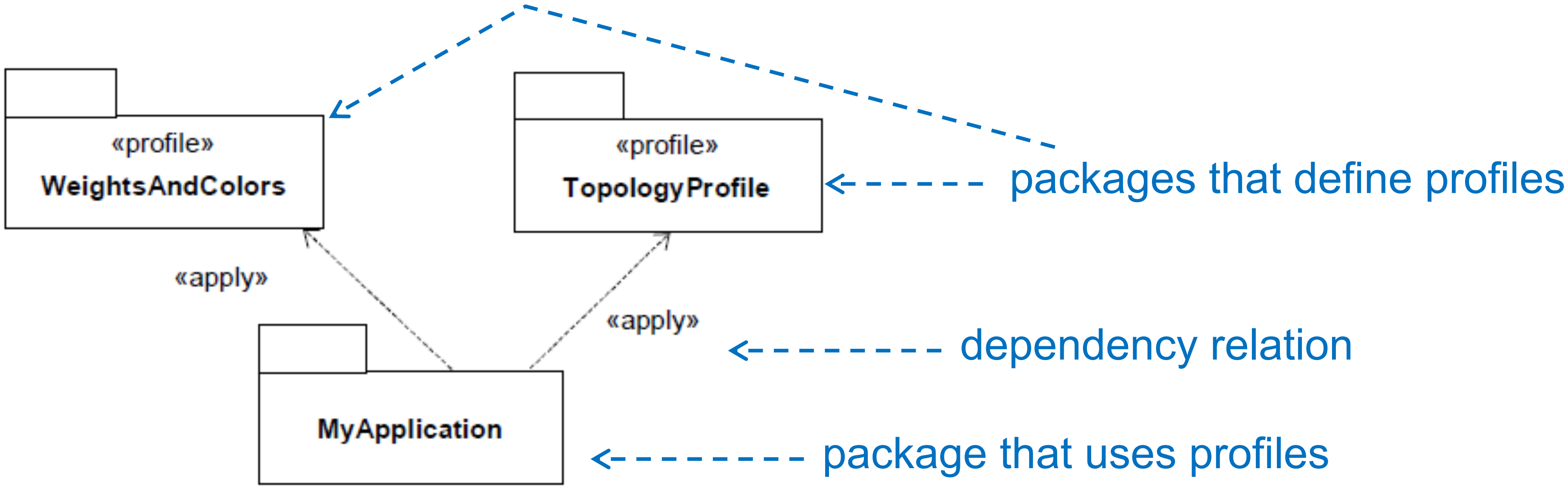
Tagged values are meta-attributes for the elements tagged by the stereotype.

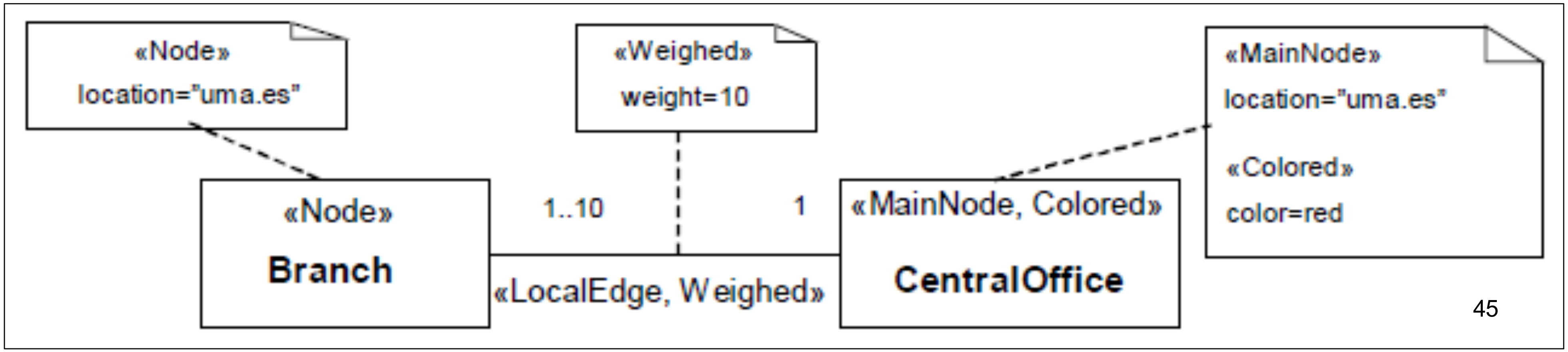
They have name and type.

context UML::InfrastructureLibrary::Core::Constructs::Association
inv : self.isStereotyped("Colored") **implies**
self.connection->**forAll**(isStereotyped("Colored") **implies** color=self.color)

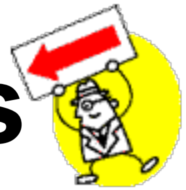
UML Profiles

Using a profile





Index

- Introduction.
- Two-level meta-modelling
- Multi-level meta-modelling
- Profiles
- **Graph grammars** 
- Bibliography

Graph transformation

- Models are graphs.
- A (multi-)graph can be defined as G=(N, E, s, t), where is a set of nodes, E is a set of edges, and con s, t:E$\rightarrow$N define the source and target nodes of edges.
- Nodes and edges can be typed, attributed.
- Formal techniques to manipulate graphs.
- Rules with left (LHS) and right hand side (RHS), both containing a graph.
- If there is an occurrence of the LHS in the host graph, we can substitute the occurrence by the RHS.

Graph transformation

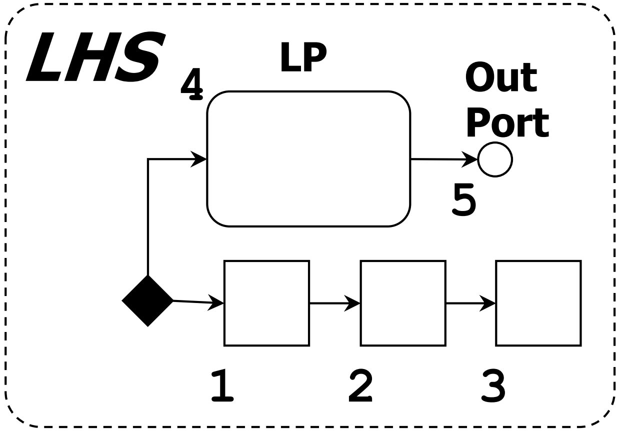
- Advantages: Visual, formal, declarative technique to express graph manipulations.

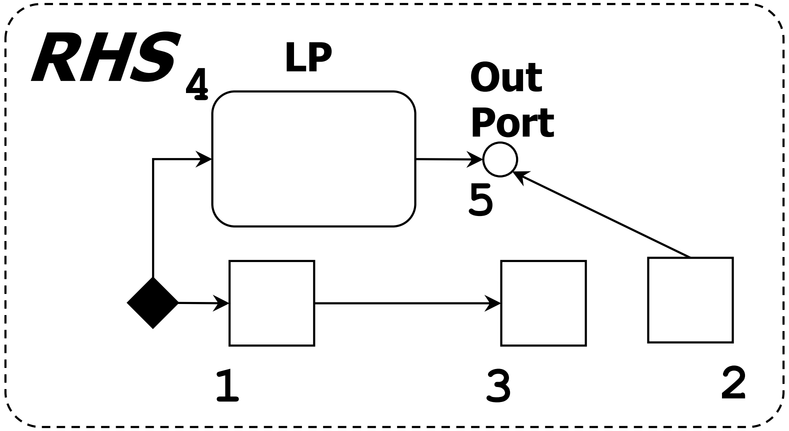
- Formal technique:
  - Based on category theory.
  - Analysis:
    - Termination (partially).
    - Confluence.
    - Dependencies/Conflicts/Concurrency between rules.

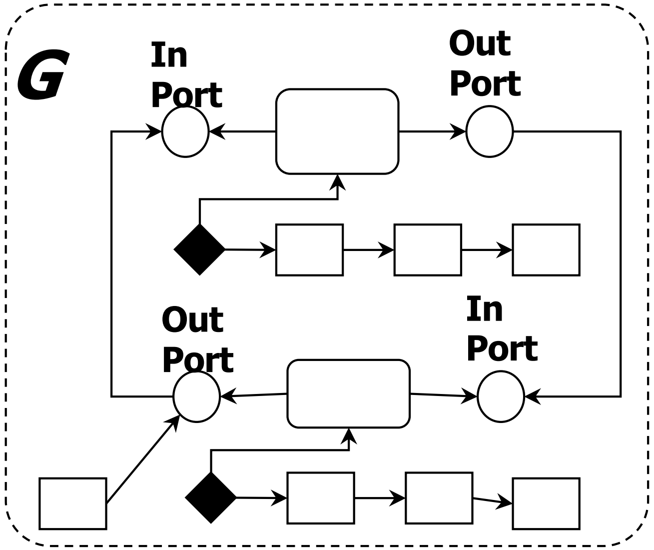
- Disadvantages:
  - Expressive power of rules (multi-objects, etc.)?
  - Execution control flow?

Graph transformation

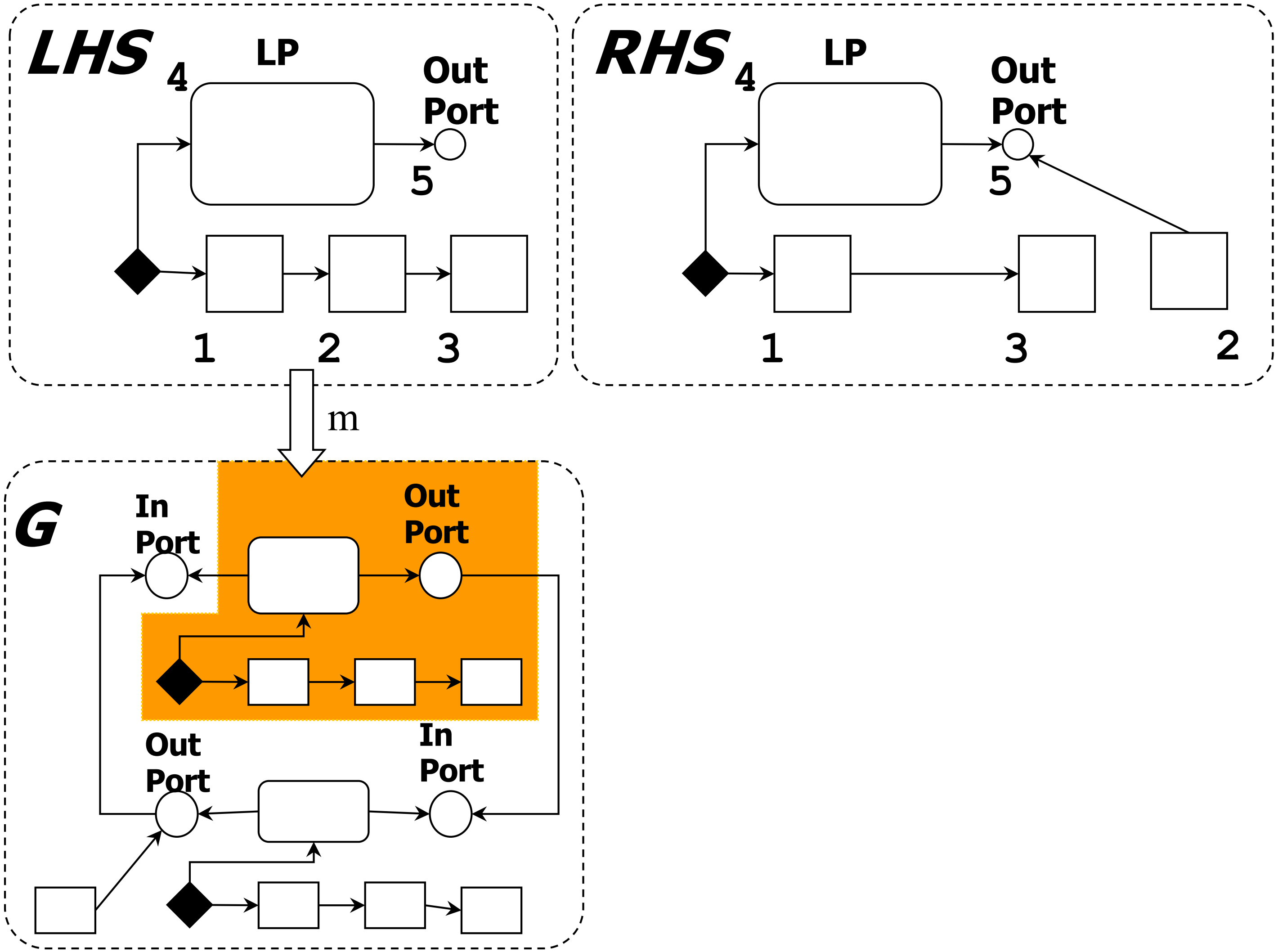
Example





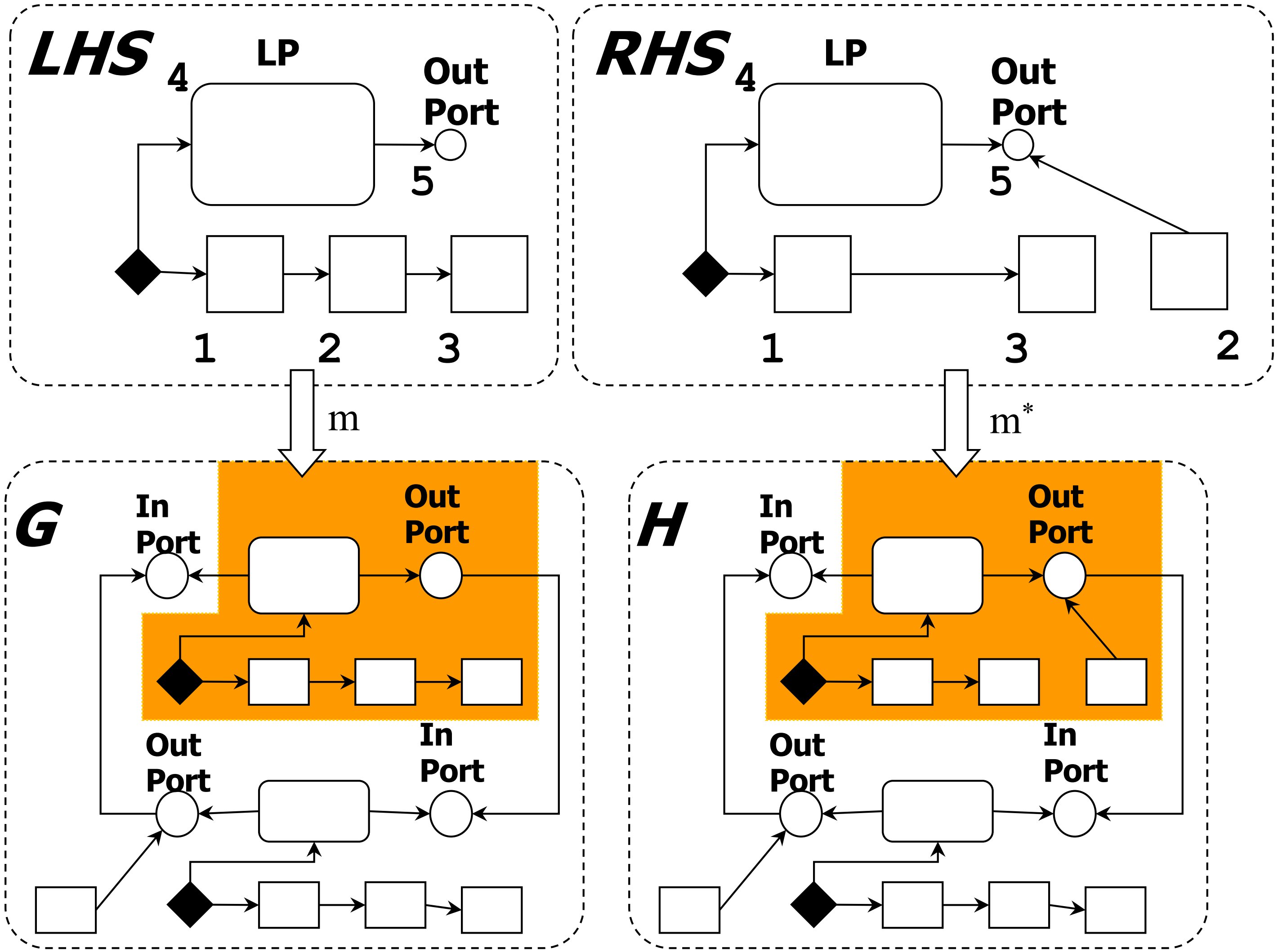


Graph transformation
Example



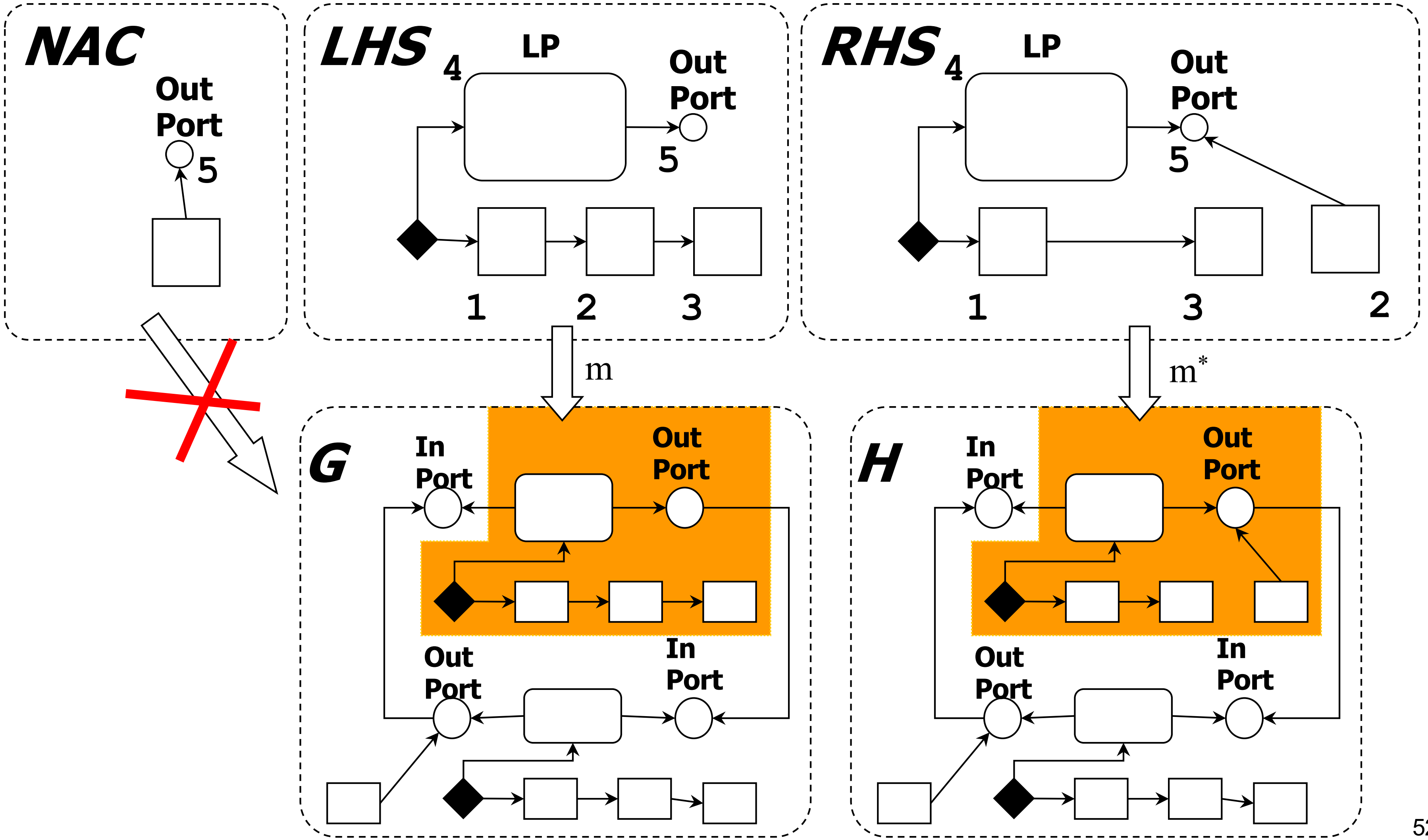
Graph transformation

Example



Graph transformation

Example, Negative Application Conditions



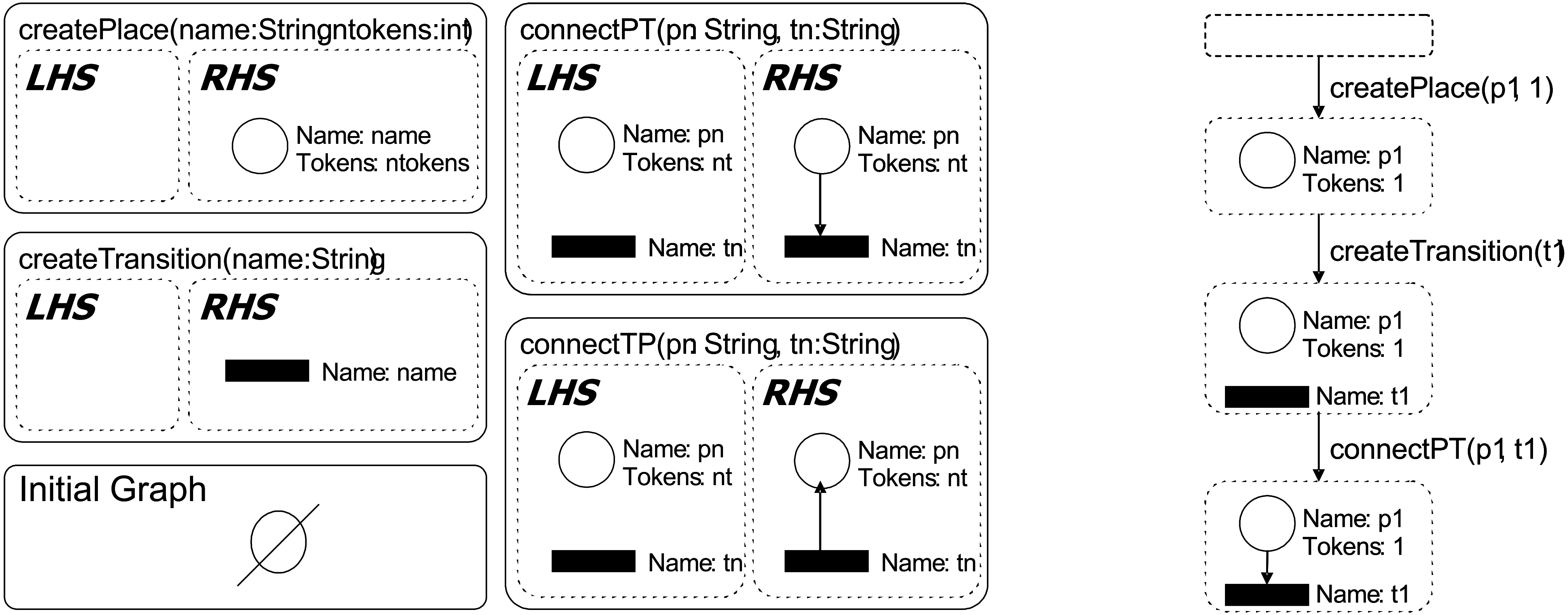
Graph transformation

Creation grammars

- A graph grammar consists of a set of rules and an initial graph: $GG=\{\{p_1,\ldots,p_n\}, G_0\}$.
- A grammar describes a language: The grammar semantics are all graphs that can be derived by applying the grammar rules in 0 or more steps.
- Do we need additional constraints (like we use OCL in meta-modelling)?
  - Application conditions for rules.
  - Graph constraints.

Graph transformation

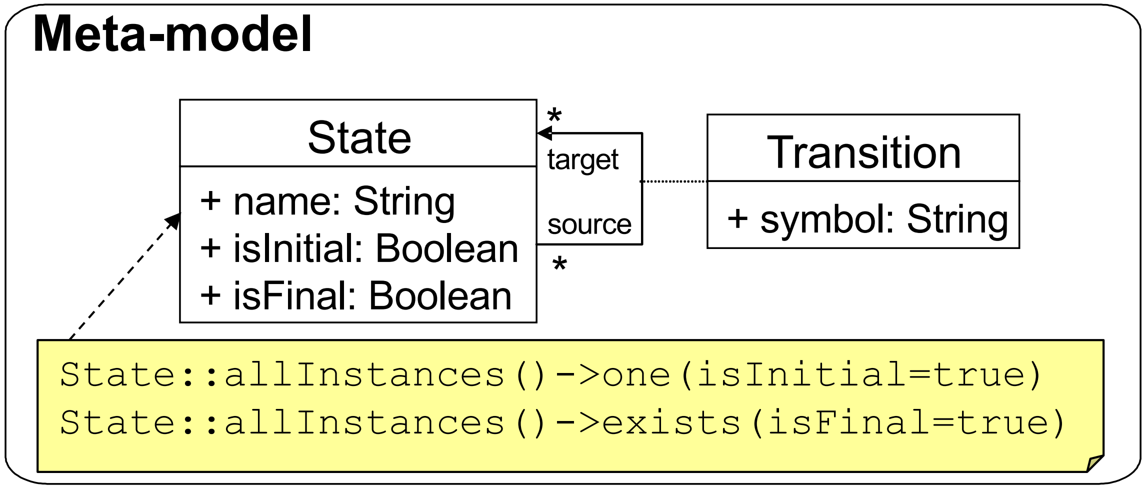
Example, Petri nets



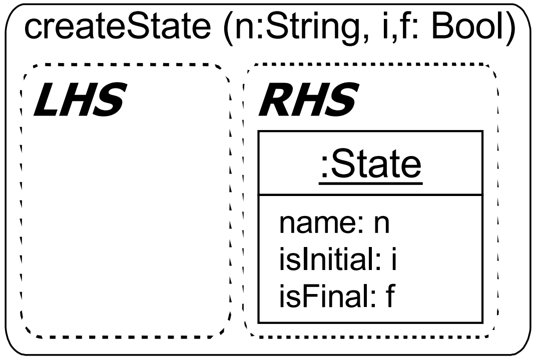
Graph transformation

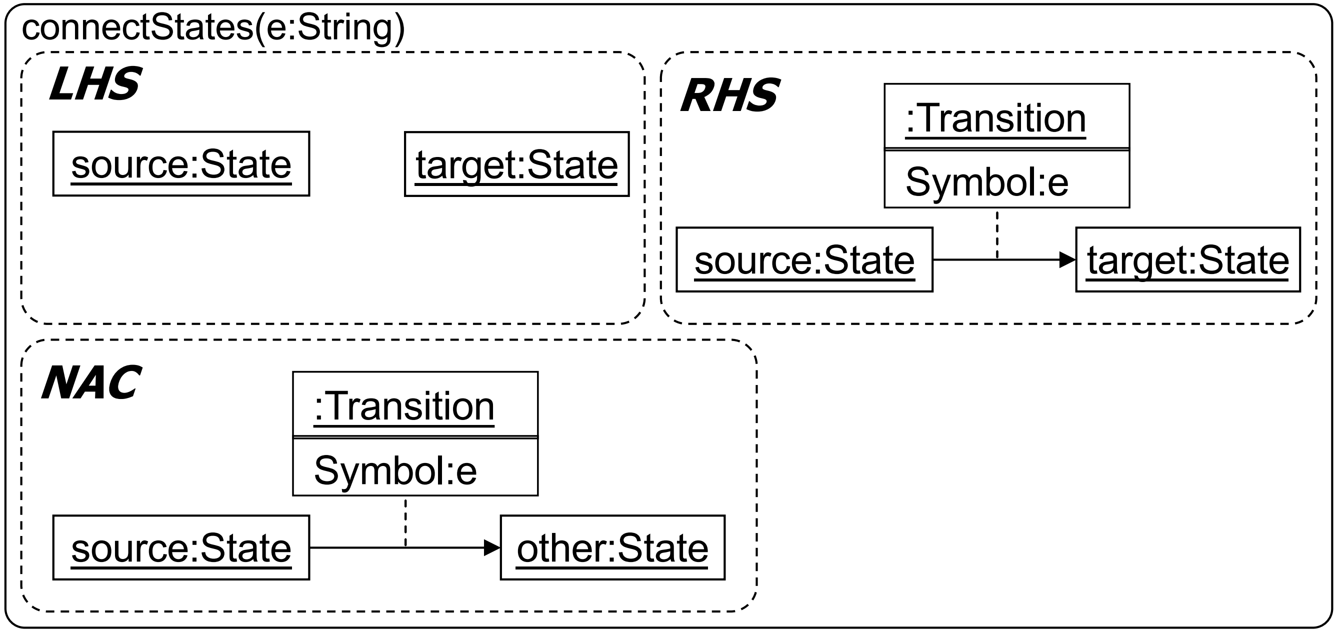
Exercise, Finite automata

- Build a creation grammar for deterministic finite automata, which generates the same language as the following meta-model:



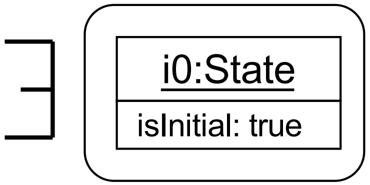
Graph transformation
Exercise, Finite automata



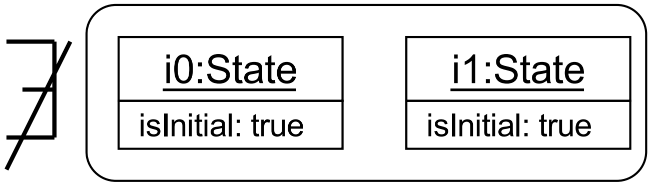


Additional graph constraints:

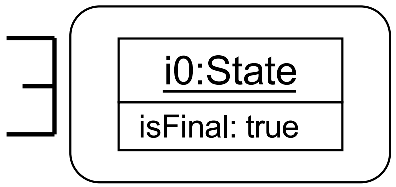
One initial



Unique initial



Some final



Bibliography

- Two-level meta-modelling:
  - "*Model-driven software development*". 2006. M. Völter, T. Stahl. 2006. Willey.
  - "*EMF: Eclipse Modeling Framework*". 2008. D. Steinberg, F. Budinsky, M. Paternostro, E. Merks. Addison Wesley Professional 2nd edition.

- Multi-level meta-modelling:
  - "*Rearchitecting the UML infraestructure*". 2002. C. Atkinson, T. Kühne. ACM Transactions on Modeling and Computer Simulation, Vol 12(4), pp.: 290-321.
  - "*When and how to use multi-level modelling*". 2014. J. de Lara, E. Guerra, J. Sánchez-Cuadrado. ACM Trans. Softw. Eng. Methodol. 24(2): 12:1-12:46.

Bibliography

- Profiles:
  - “*An introduction to UML profiles*”. 2004. A. Vallecillo, L. Fuentes. Novatica, pp: 6-13.
- Graph transformation:
  - “*Fundamentals of algebraic graph transformation*”. 2006. H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. Springer.