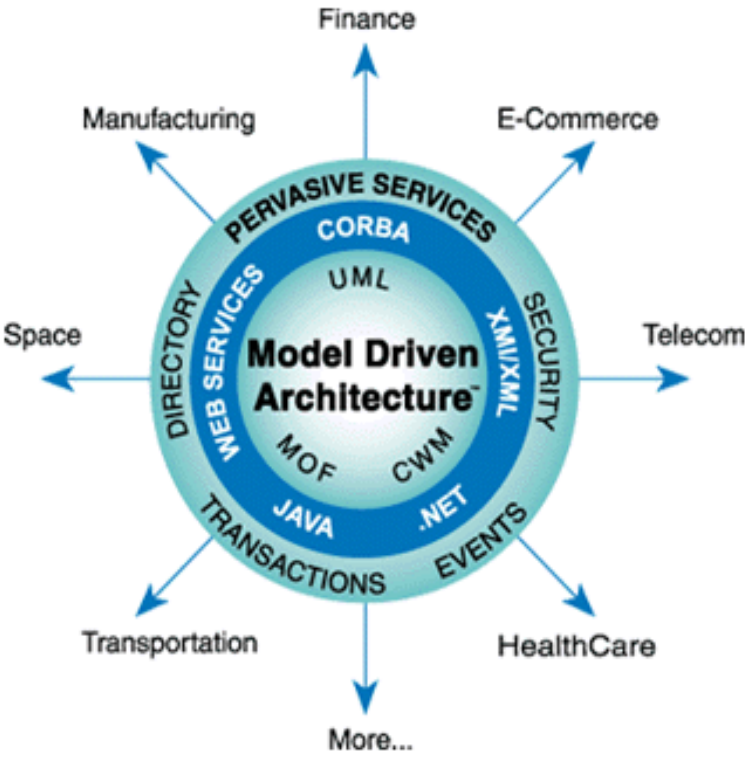
The Model Driven Architecture

Dennis Wagelaar
Viviane Jonckers
Software Languages Lab

Vrije Universiteit Brussel

Special session:
The Model Driven Architecture (MDA)

- MDA origins, goals
- MDA concepts
  - Platform, CIM, PIM, PSM...
  - Models and meta-models
  - The role of UML in MDA
  - Model transformation
- MDA applied
  - Case study
- Further reading



From http://www.omg.org/mda/

MDA origins:
Programming vs. software engineering

"The sooner you start, the longer it takes."

Fred Brooks Jr, *The Mythical Man Month, 1975.*

- Good Design Matters
  - Skimp on Requirements Capture
    ⇒ Design the wrong system
    - Unfamiliar application domains
    - Multiple stakeholders
  - Skimp on Design
    ⇒ Inconsistent implementation
    - Different programmers think different

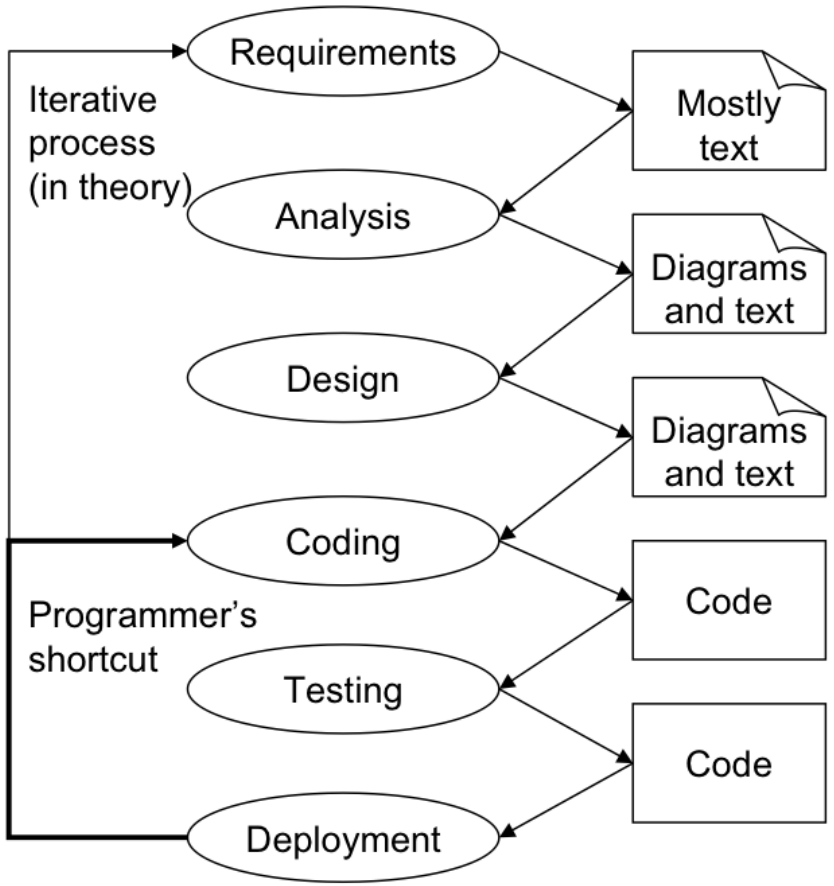
MDA origins:
The role of modelling

- Models of a system...
  - ...help communicate the properties of that system amongst its stakeholders (software engineers, customers, ...)
  - ...are used for Requirements Capture as well as Design
- Modelling languages:
  - Object oriented modelling languages
    - OMT (Rumbaugh), Booch, Yourdon and now the Unified Modeling Language (UML)
  - Domain-specific modelling languages
    - Matlab Simulink, LabView, MetaCase, SysML, ...

MDA origins:
The missing link in modelling

- Models of a system...
  - ...are manually implemented in a programming language
  - ...serve only as documentation and provide no guarantee on the implemented system
  - ...gradually fall out of sync with the code and no longer truthfully represent the implemented system

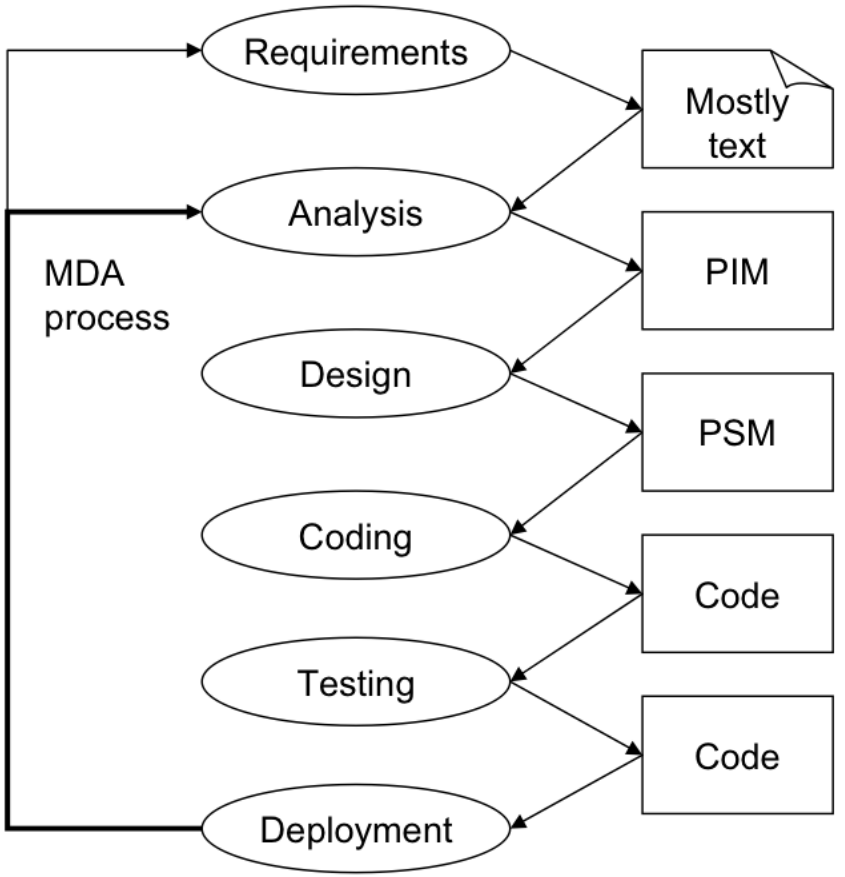
From [Kleppe et al., 2003]



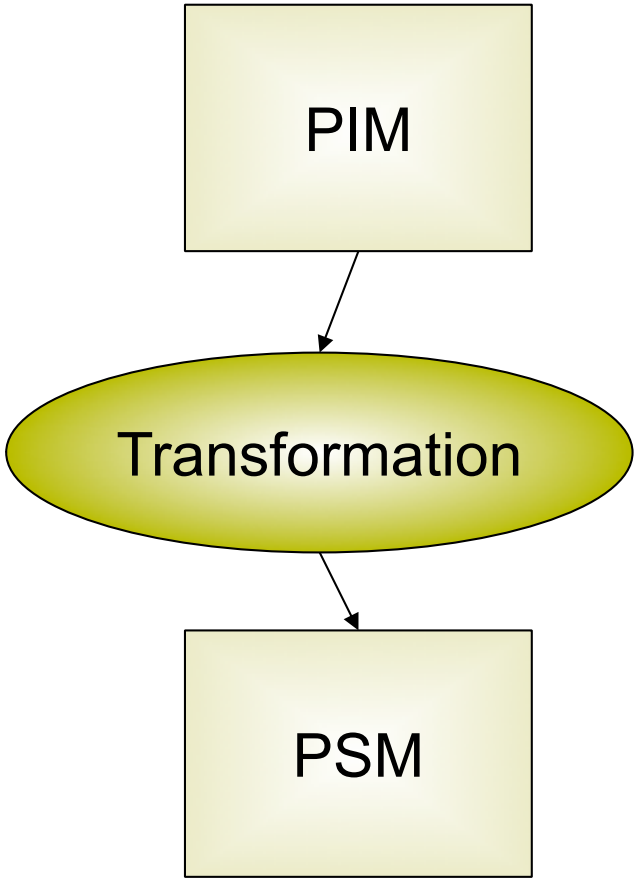
MDA goals:
Models as part of the automation chain

- Models of a system...
  - ...should be structured and machine readable
  - ...should be well-defined and correspond to the implemented system
  - ...should survive system evolution
    - Maintenance
    - Platform changes

From [Kleppe et al., 2003]



MDA goals:
The MDA pattern



- Design a Platform Independent Model (PIM)
- Automatically transform to a Platform Specific Model (PSM)
- Repeat until you reach code
  - "Platform Independent" is relative:
    - Example: independent from J2SE, J2EE and J2ME, but specific to Java
    - Example: independent from OOP, RDBMS, but specific to data modelling

MDA goals:
Envisioned benefits

- Productivity
  - Shift development focus from PSM to PIM
  - Reusable PIM-to-PSM transformations
- Portability
  - PIMs are portable to multiple platforms
- Maintenance and Documentation
  - Automatic transformations keep (derived) artefacts in sync

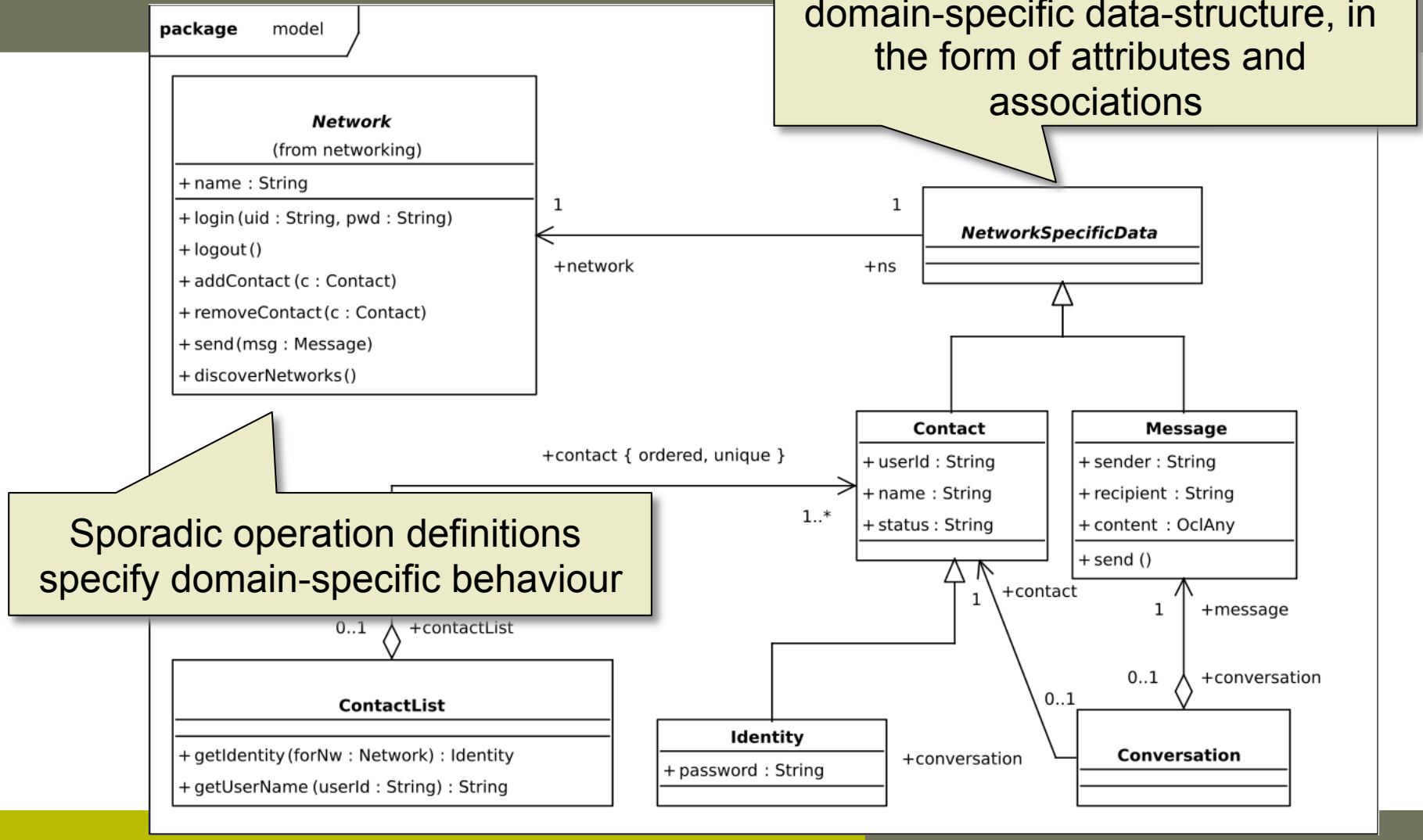
MDA concepts:
Platform

- What exactly is meant by "platform" in the MDA?
  - "A *platform* is a set of subsystems and technologies that provide a coherent set of functionality through interfaces and specified usage patterns, which any application supported by that platform can use without concern for the details of how the functionality provided by the platform is implemented." [Miller and Mukerji, 2003]
  - Loose interpretation: "The combination of hardware and software features on which the system under development depends" → ***Platforms are about dependencies***
  - Examples: Java, J2EE, Hibernate, Linux, i386, AWT, glibc, ...

MDA concepts:
CIM, PIM, PSM and PM

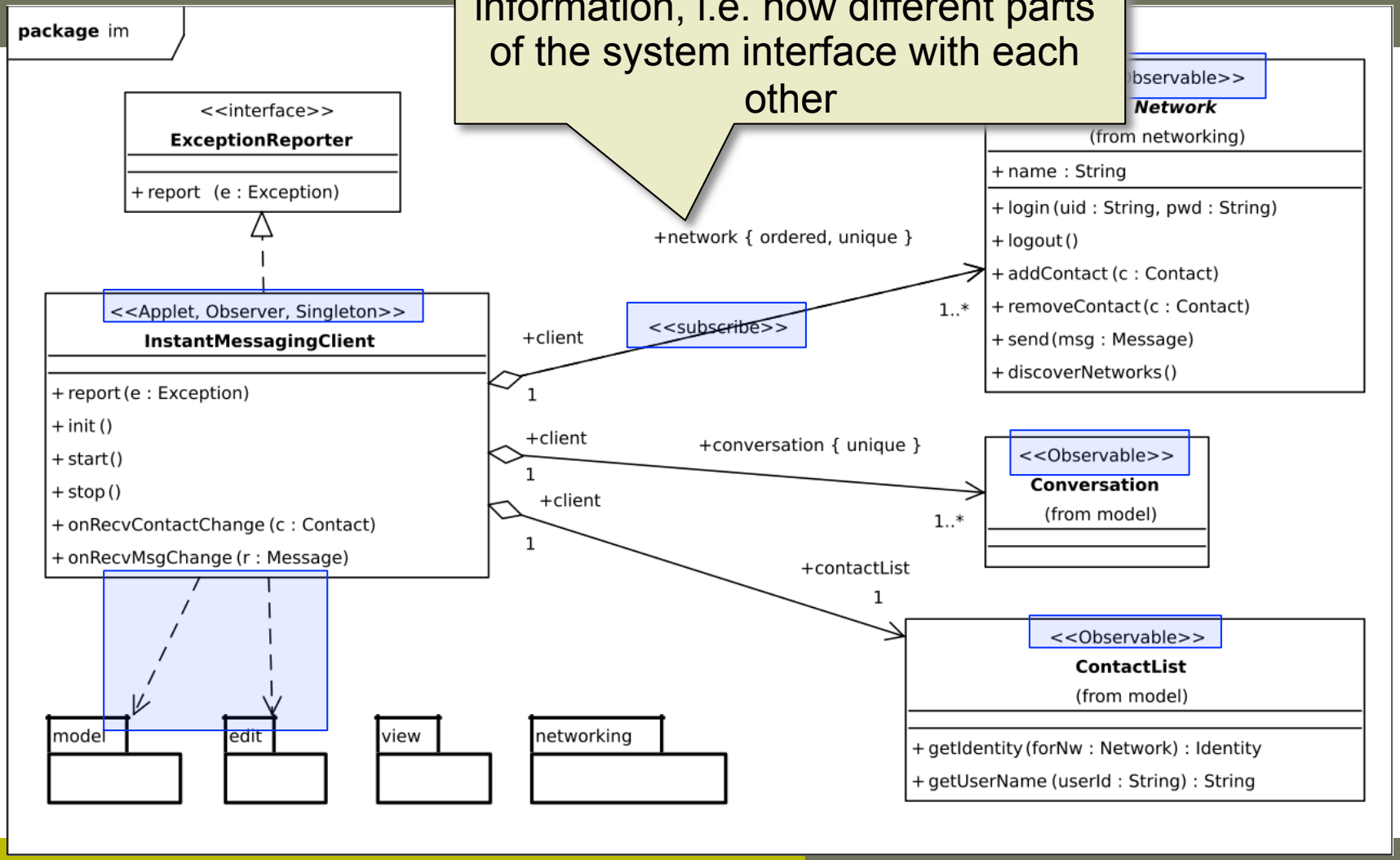
- The MDA predefines certain kinds of models:
  - **Computation Independent Model (CIM):** A conceptual, data-centric model of the system's *application domain*
  - **Platform Independent Model (PIM):** An architectural model that includes interfacing information between parts of the system, and may include behavioural specifications
  - **Platform Specific Model (PSM):** PIM including technical details with dependencies on a specific platform
  - **Platform Model (PM):** Model describing platform features that we can interface with

MDA concepts:
CIM example

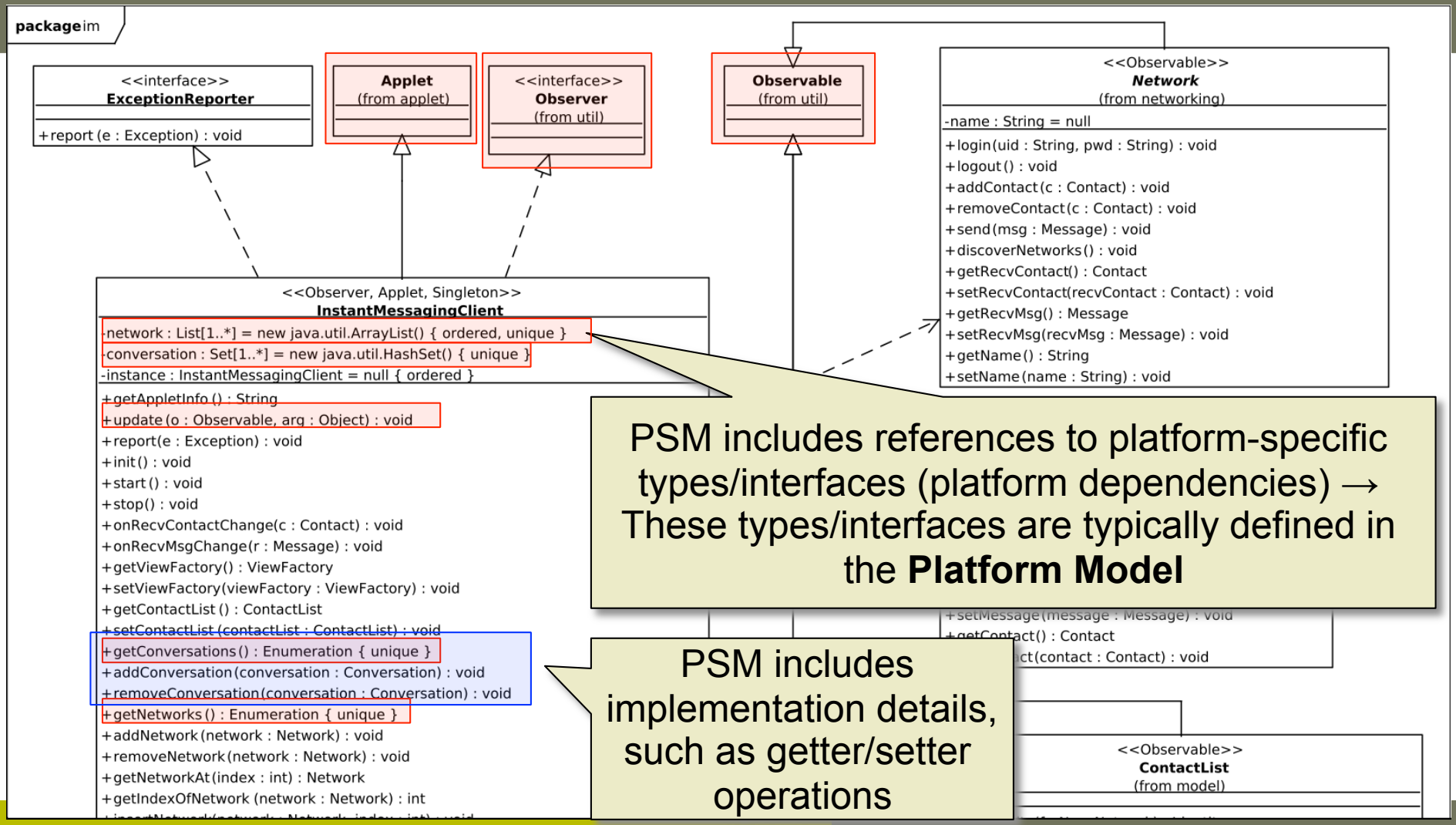


MDA concepts:
PIM example

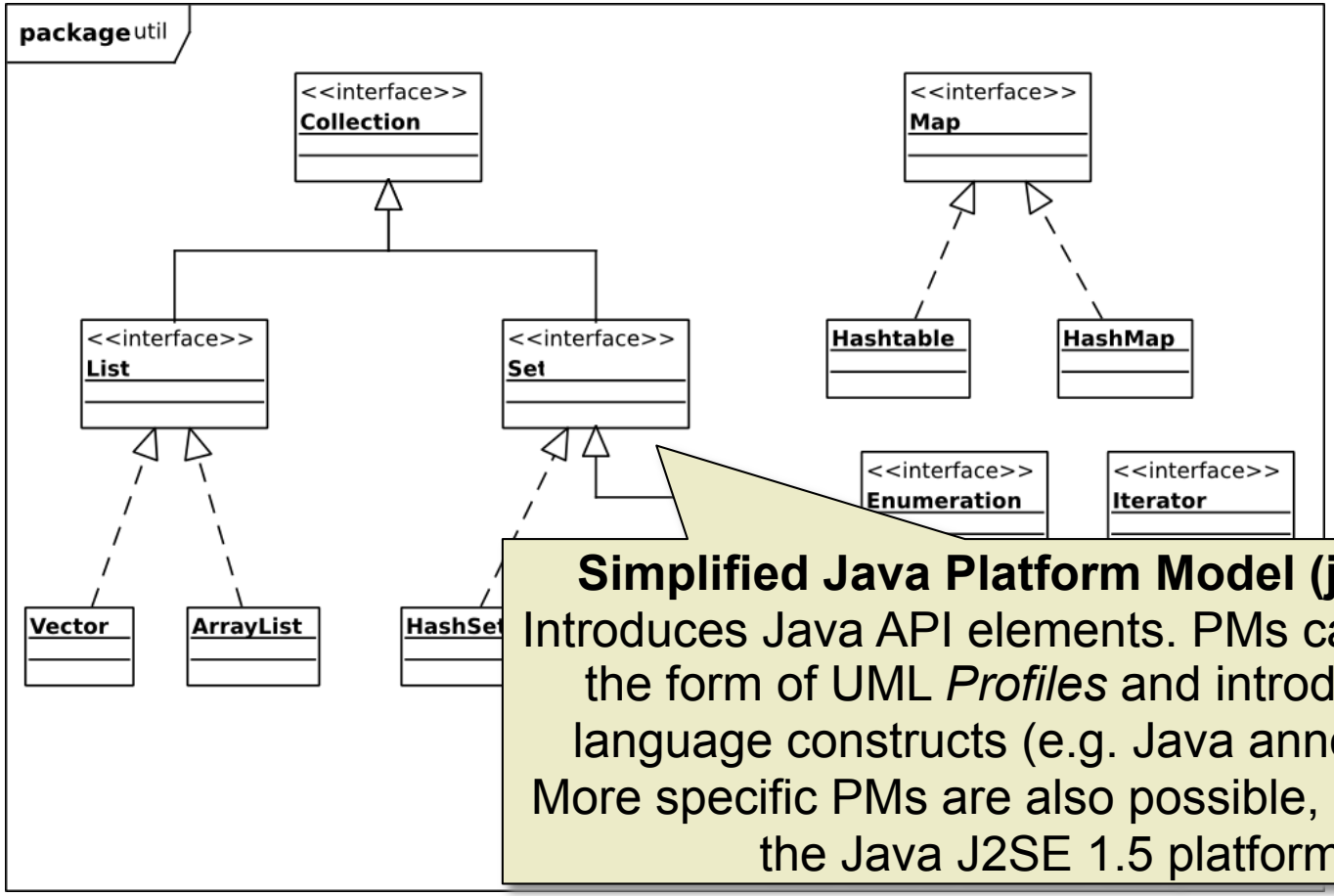
PIM includes architectural information, i.e. how different parts of the system interface with each other



MDA concepts:
PSM example



MDA concepts:
PM example



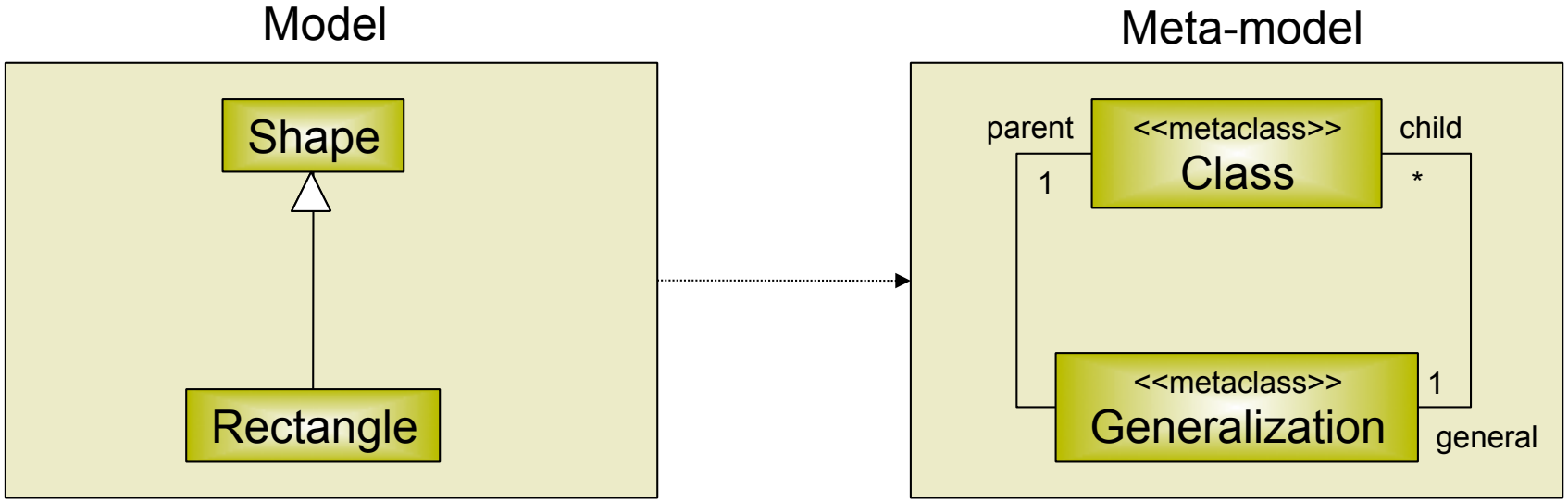
Simplified Java Platform Model (java.util): Introduces Java API elements. PMs can also take the form of UML *Profiles* and introduce new language constructs (e.g. Java annotations). More specific PMs are also possible, such as for the Java J2SE 1.5 platform.

MDA concepts:
Models and meta-models

- Models in the MDA are structured and machine readable
- The structural rules are defined in a **meta-model: "Model about models"**
  - Meta-models often expressed in the **Meta Object Facility (MOF)** language, which is a simplified UML class diagram language
  - Can also be formally defined using **(type) graphs**
  - Goal: Describe the abstract syntax of a modelling language

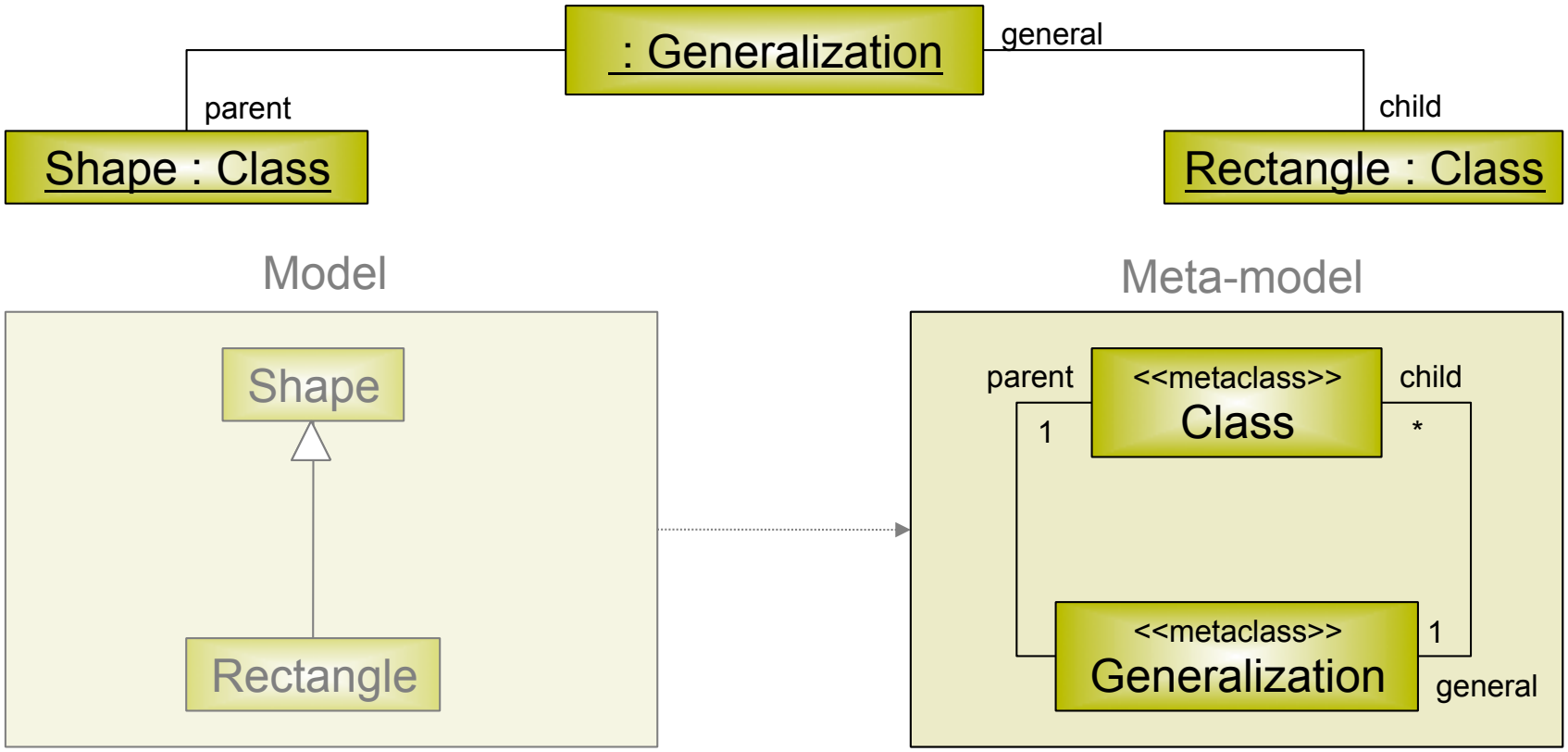
MDA concepts:
Models and meta-models

- **Models** conform to a **meta-model** if they follow the syntactic rules of that meta-model
- **Model elements** are instances of **meta-classes** defined in the meta-model

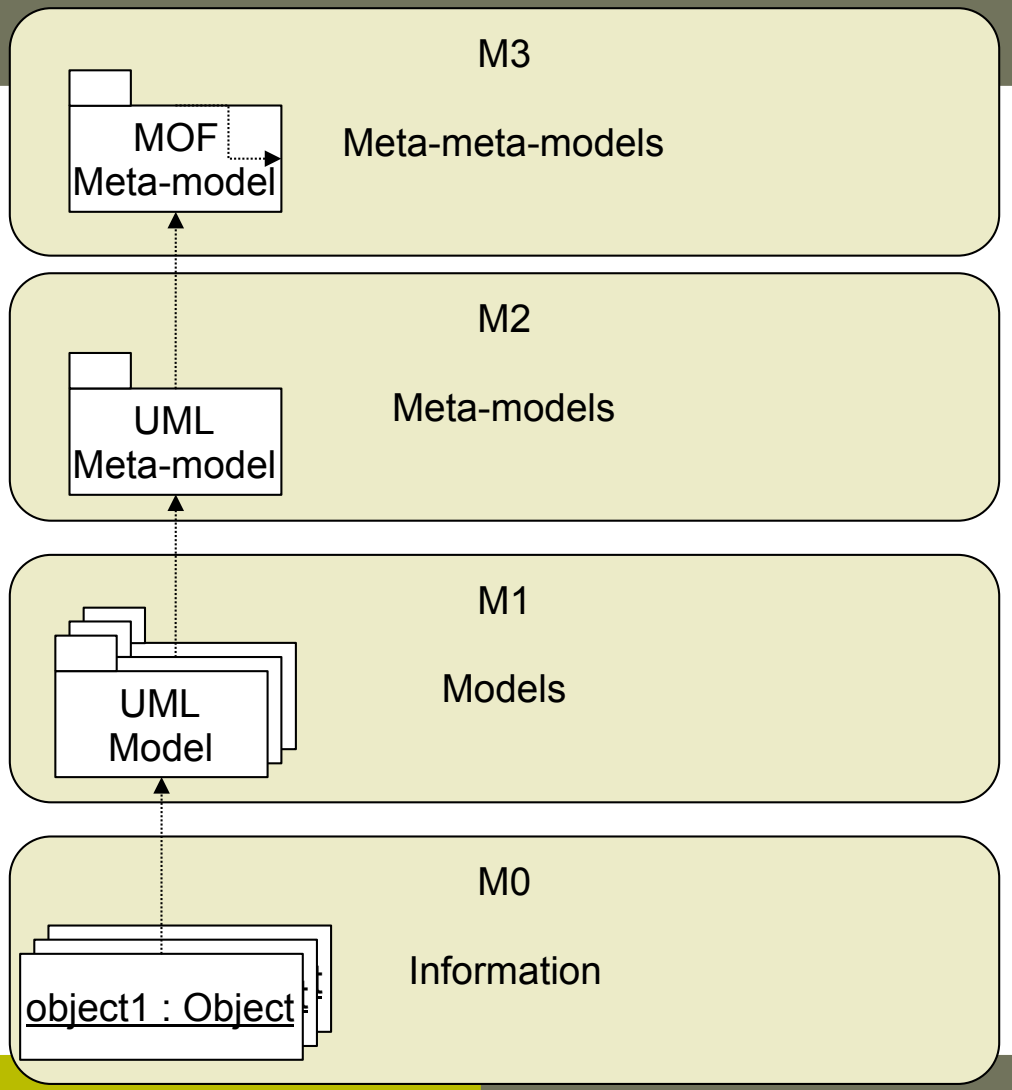


MDA concepts:
Models and meta-models

- Object representation of a model:

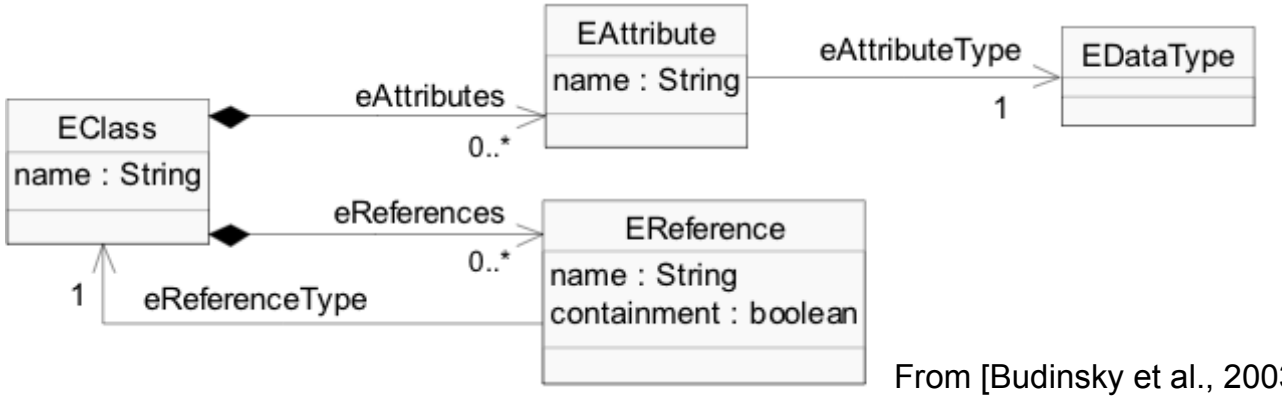


Models and meta-models:
Meta-model stack



Models and meta-models:
Eclipse Ecore meta-modelling language

- Ecore is a simplified version of MOF
  - Available as an Eclipse plug-in
- Meta-reflective
  - Ecore conforms to Ecore
  - EClass is an instance of EClass



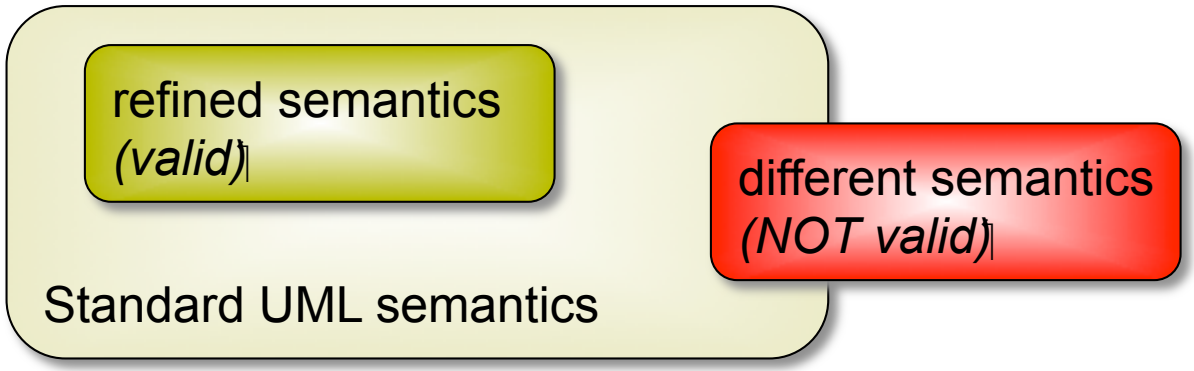
From [Budinsky et al., 2003]

Models and meta-models:
The role of the UML in the MDA

- The UML is intended as a general-purpose, OO modelling language
  - *Unified but not universal*
    - Not the ideal language for each application domain, but rather a compromise
  - Serves as a basis for extension and reuse
    - One can build domain-specific languages (DSLs) as UML extensions, but DSLs can also be defined as stand-alone languages
      - UML-RT vs. SysML

Models and meta-models:
The role of the UML in the MDA

- The UML can be extended via its MOF meta-model, but it also defines its own extension mechanism: **Profiles**
  - Profiles can only add language *refinements to the UML, while staying within the predefined UML semantics*



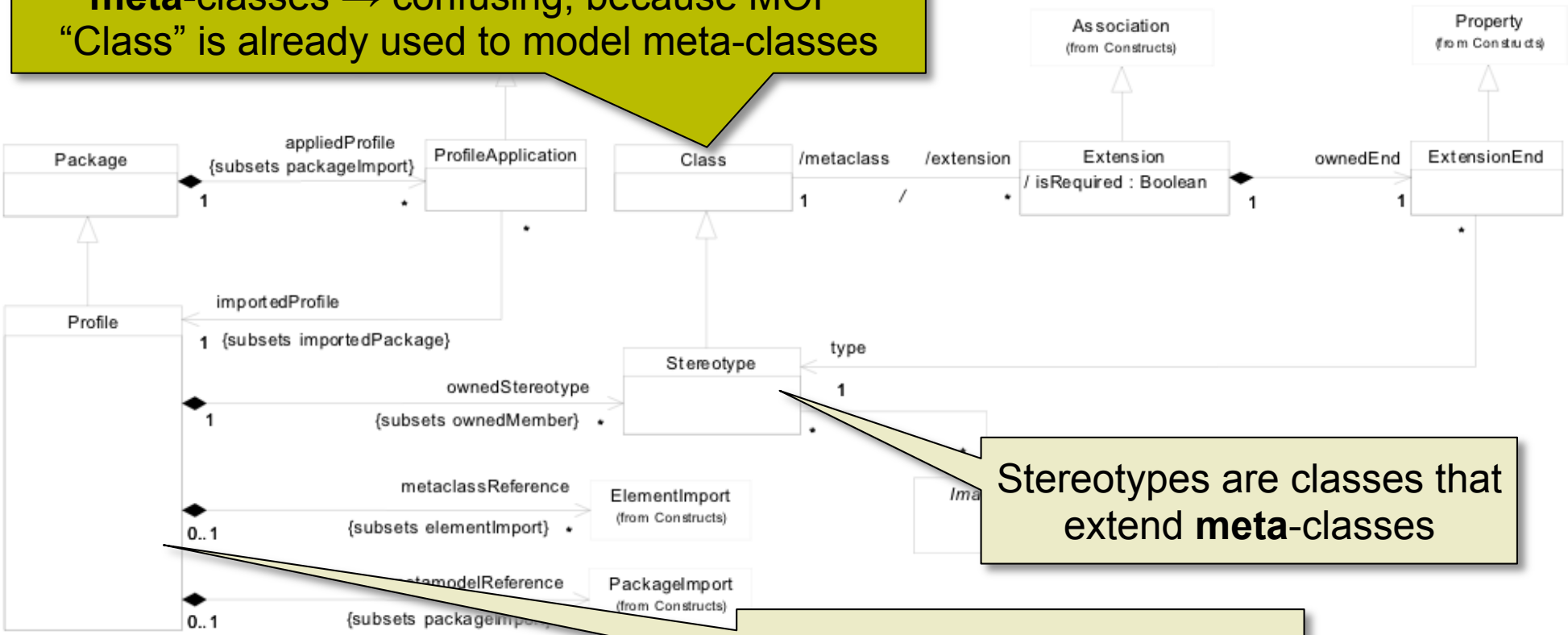
Adapted from OMG's UML tutorial

The role of the UML in the MDA: UML Profiles

- Profiles (UML 2.1) consist of
  - Stereotypes
    - Used to refine semantics of UML meta-classes
    - Can have attributes that allow for adding extra information on stereotyped model elements
  - Constraints
    - Used to narrow down allowed expressions
    - Often expressed in the Object Constraint Language (OCL)

The role of the UML in the MDA:
UML meta-model: Profiles

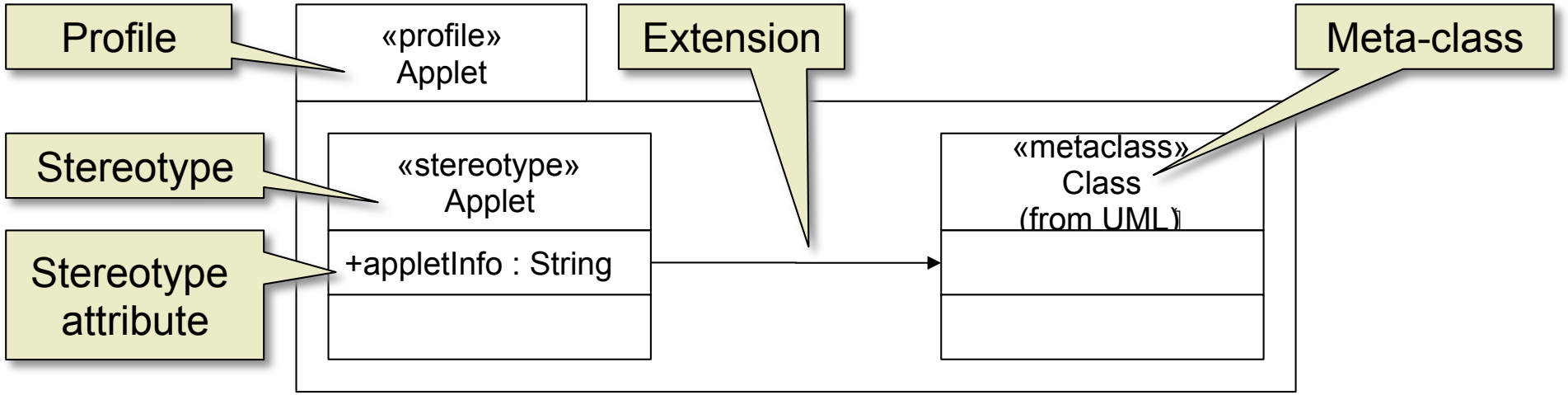
UML “Class” is used to model classes as well as **meta**-classes → confusing, because MOF “Class” is already used to model meta-classes



Stereotypes are classes that extend **meta**-classes

Profiles are Packages that can also contain stereotypes

The role of the UML in the MDA:
UML Profile example

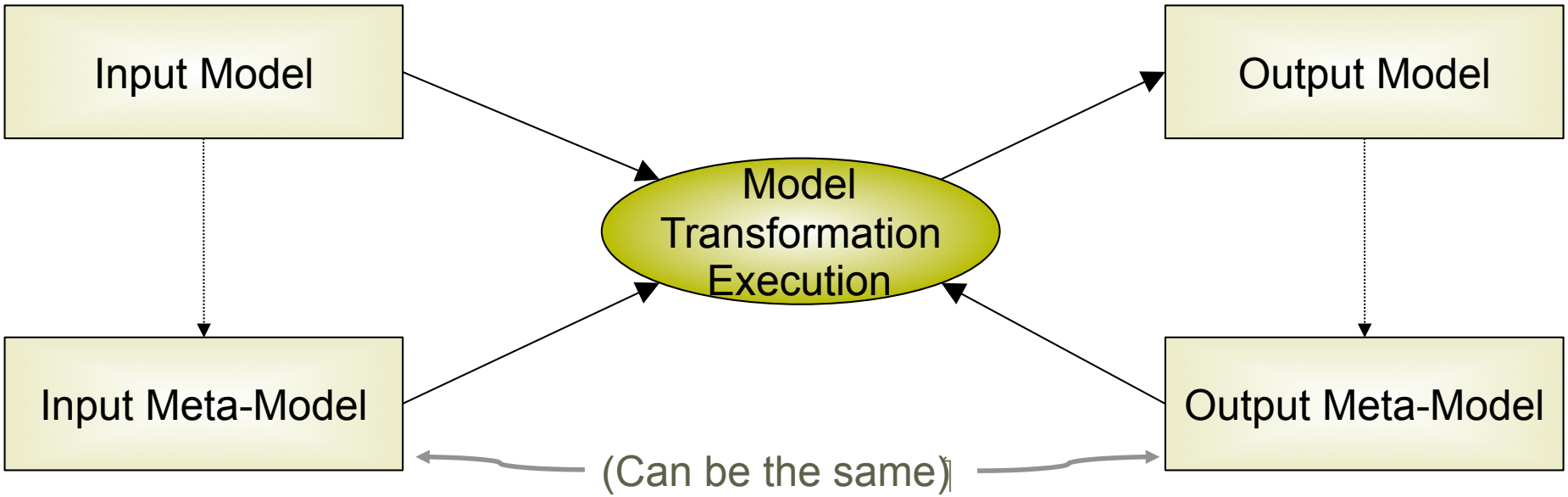


MDA concepts:
Model transformation

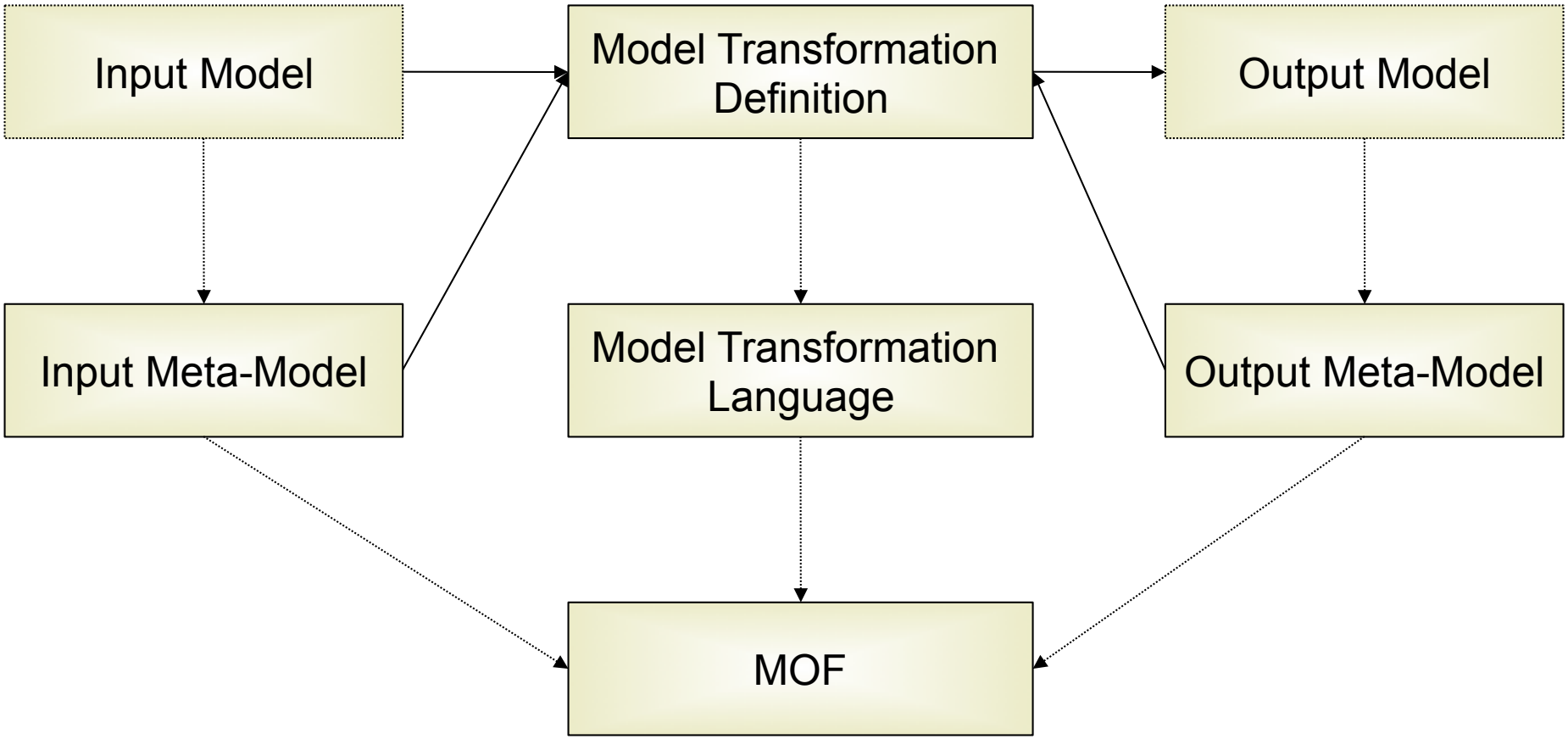
- Model transformation plays a central role in the MDA
  - Generate derived MDA elements, such as PSMs, and code
- The term "model transformation" is ambiguous and can mean all of the following:
  - **Model transformation definition** refers to an expression in a model transformation language that can be executed by a transformation engine
  - **Model transformation execution** refers to the application on a model transformation definition to a specific set of input models

MDA concepts:
Model transformation

- Model transformations use meta-models to express which model elements it transforms



Model transformation:
Transformations as models



Model transformation: Scenarios

	Horizontal	Vertical
Endogenous	*Refactoring*	*Refinement*
Exogenous	*Language migration*	*Code generation*

Adapted from [Mens and Van Gorp, 2005]

- PIM-to-PSM transformations are *vertical* transformations
  - Can be *endogenous (=same input/output language)*
  - *Can be exogenous (=different input/output language)*

Model transformation: Languages

For an overview of model transformation languages, see [Czarnecki and Helsen, 2006]

- Several model transformation languages exist:
  - MOF-based
    - QVT (Relations, Operational mappings, Core)
    - ATL, Tefkat, Kermeta, BOTL, UMLX, MOLA, ...
  - Graph-based
    - Graph grammars (AGG, Atom3, GreAT, Viatra)
    - Triple graph grammars (Fujaba)
- We use ATL in our examples
  - good tool support
  - can deal with MOF-based languages, such as UML

Model transformation:
Example

- Class-to-Relational
  - Transforms class diagrams into relational schemas
    - Source meta-model Class is a simplification of class diagrams
    - Target meta-model Relational is a simplification of relational schemas
    - Expressed in ATL
- Example courtesy from Eclipse OMCW:
  - http://www.eclipse.org/gmt/omcw/resources/

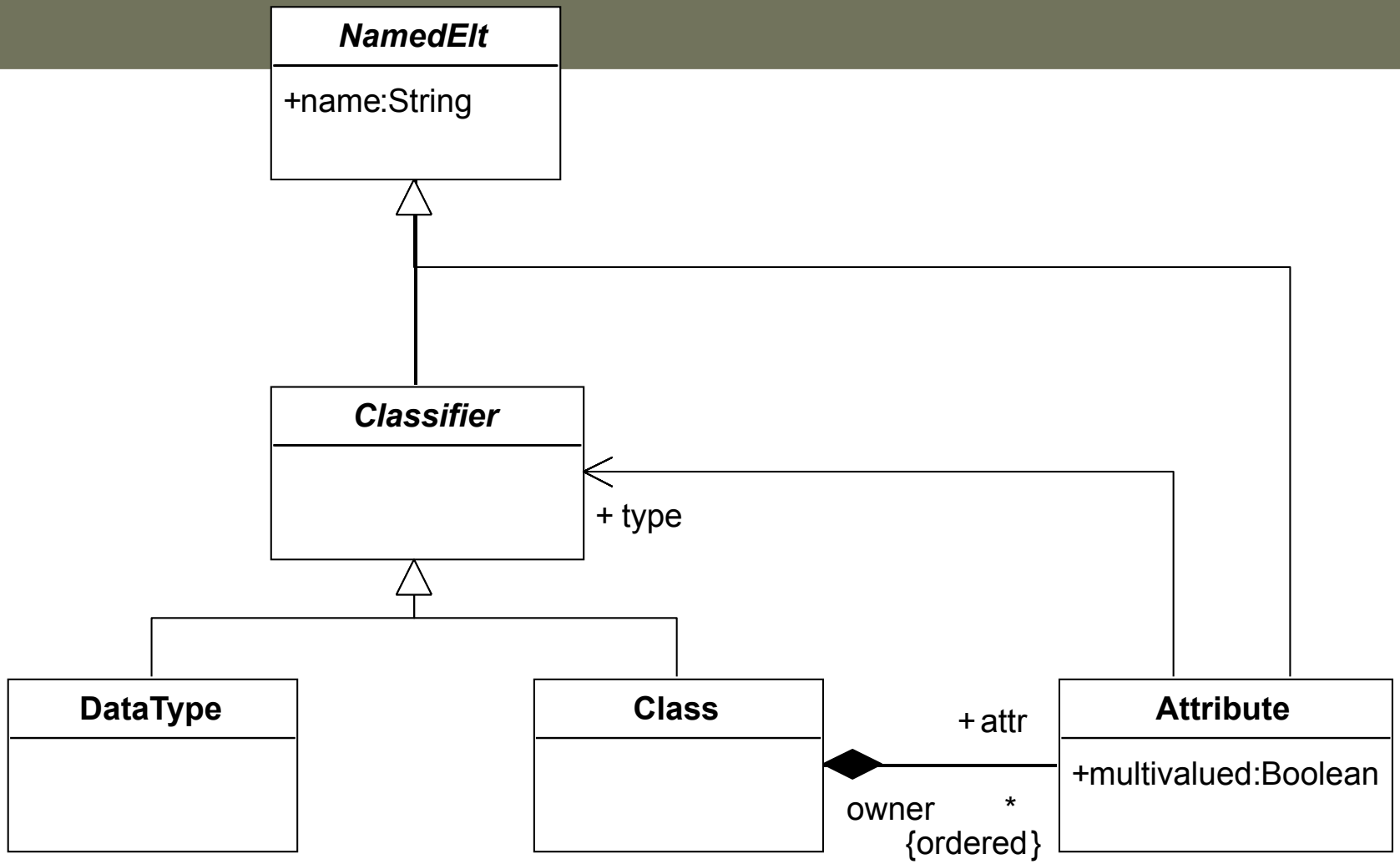
Model transformation:
Class-to-Relational example

- An ATL transformation module...
  - ...creates one or more output models using one or more input models
  - ...consists of a number of transformation rules

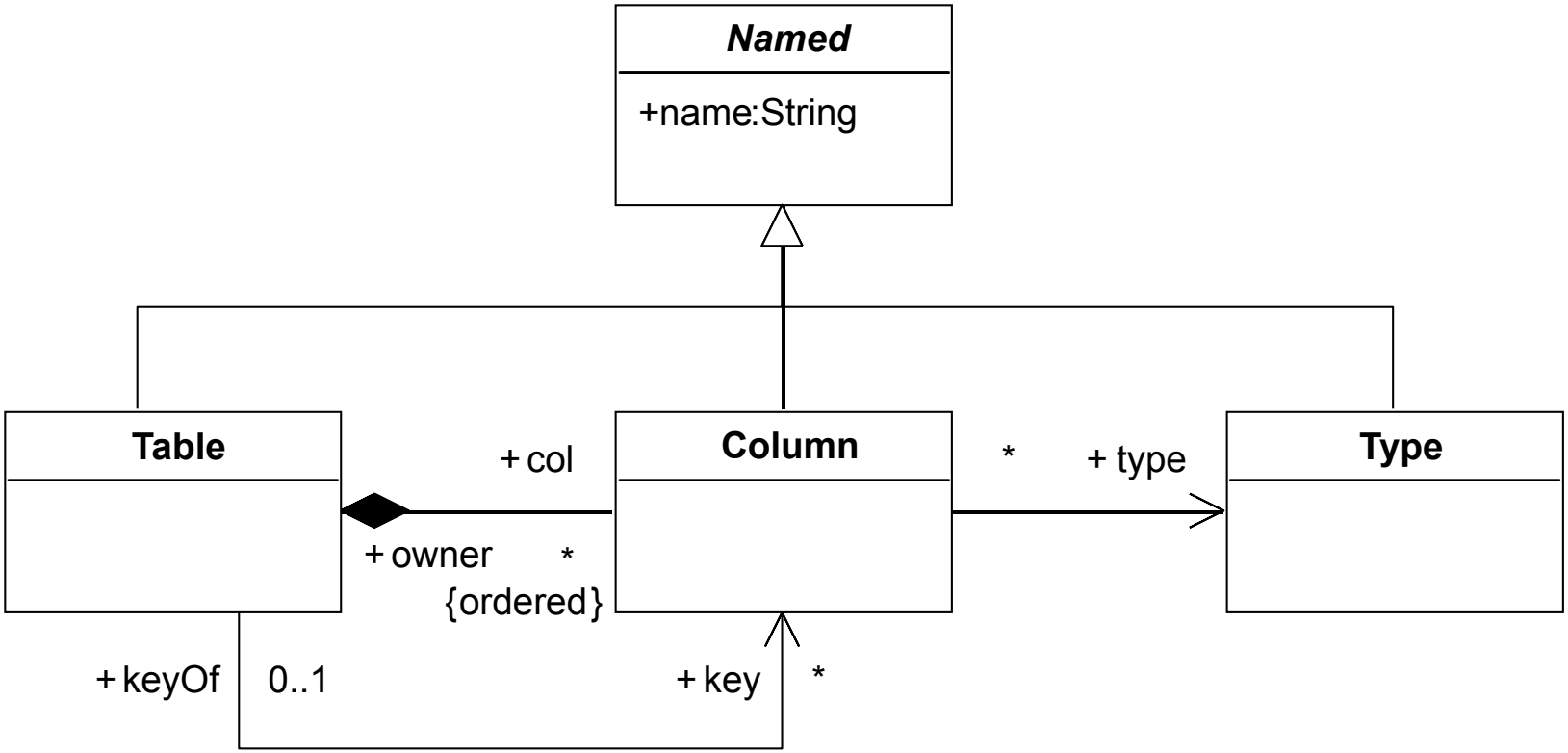
```
module Class2Relational;
create Mout : Relational from Min : Class;

rule Class2Table { ... }
rule SingleValuedAttribute2Column { ... }
rule MultiValuedAttribute2Column { ... }
```

Class-to-Relational example:
Class meta-model



Class-to-Relational example:
Relational meta-model



Class-to-Relational example: Class2Table transformation rule

- A Table is created for each Class
- The name of the Table is the name of the Class:
- The columns of the table correspond to the single-valued attributes of the class

```
rule Class2Table {
   from
      c : Class!Class
   to
      t : Relational!Table (
        name <- c.name,
        col <- c.attr->select(e|not e.multiValued)
      )
}
```

a simple binding

complex navigation

attributes are automatically resolved into columns (triggered on '<-')

Class-to-Relational example: SingleValuedAttribute2Column rule

- A Column is created for each single-valued Attribute:

```
rule SingleValuedAttribute2Column {
  from   -- the guard is used for selection
    a : Class!Attribute (not a.multiValued)
  to
    c : Relational!Column (name <- a.name)
}
```

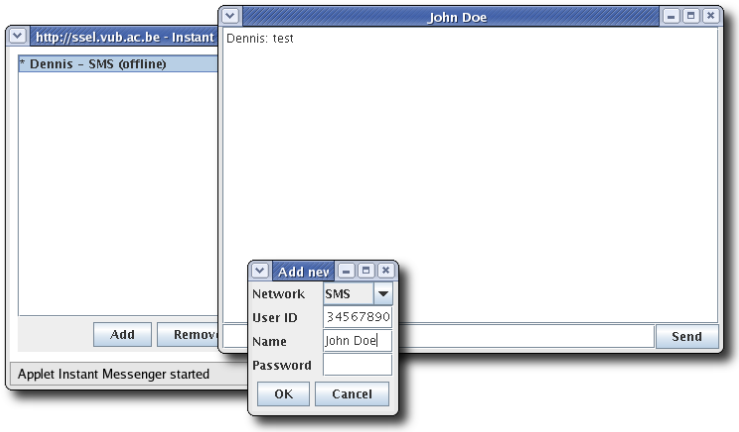
Class-to-Relational example: MultiValuedAttribute2Column rule

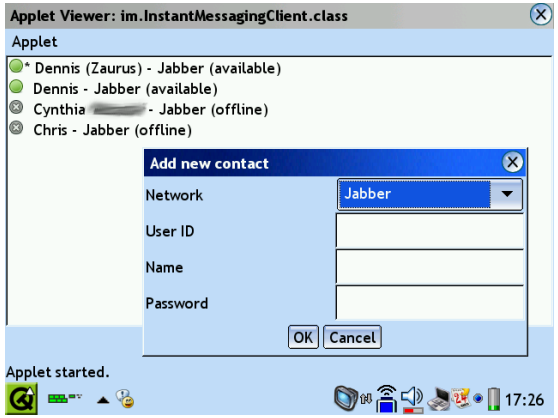
- A Table with two columns is created for each multi-valued Attribute
- The identifier of the table is created from the names of the class owner of the Attribute and the name of the attribute
- The columns get the names 'Id' and the name of the attribute and will store id/value pairs

```
rule MultiValuedAttribute2Column {
  from
    a : Class!Attribute (a.multiValued)
  to
    t : Relational!Table (
          name <- a.owner.name + '_' + a.name,
          col <- Sequence {id, value}),
    id : Relational!Column (name <- 'Id'),
    value : Relational!Column (name <- a.name)
}
```

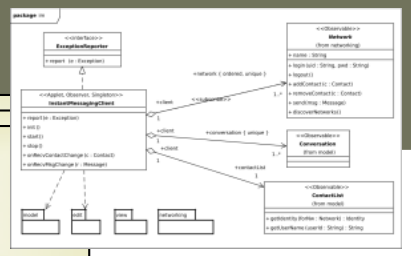
MDA applied:
Case study

- Instant messaging client
  - 11 PIM-to-PSM refinement transformations
  - One core PIM and 7 optional feature PIMs
  - Targets various Java client platforms





MDA applied:
Case study overview



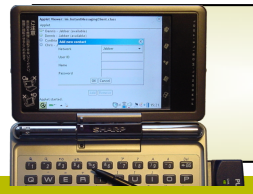
Step-wise refinement transformations generate intermediate PSMs

PIMs

MTE_1 ... MTE_n → PSM_1

MTE_1 ... MTE_n → PSM_2

MTE_1 ... MTE_n → PSM_3



Different chains of refinement transfor-mations result in PSMs for different platforms

The Model Driven Architecture: Summary 1/2

- MDA origins:
  - Good Design Matters: modelling is an important tool for that
  - The connection between model and code was weak
    - No guaranteed commonalities between model and code
    - Evolving code gradually falls out out sync with model
- MDA goals:
  - Automatic transformation from PIMs to PSMs
  - Envisioned benefits: Productivity, Portability, Interoperability, Maintenance and Documentation

The Model Driven Architecture: Summary 2/2

- MDA concepts:
  - Platform, CIM, PIM, PSM, PM
  - Model and meta-models
  - The role of the UML in the MDA: profiles and stereotypes
  - Model transformation: scenarios, languages & example

Further reading:
Books & Papers

- A. Kleppe, J. Warmer, W. Bast, *MDA Explained: The Model Driven Architecture : Practice and Promise* (2003)
  http://books.google.be/books?vid=ISBN032119442X
- S.J. Mellor, K. Scott, A. Uhl, D. Weise, *MDA Distilled: Principles of Model-Driven Architecture (2004)*
  http://my.safaribooksonline.com/0201788918
- *S.J. Mellor, M.J. Balcer, Executable UML: A Foundation for Model-Driven Architecture (2002)*
  http://www.executableumlbook.com/
- *J. Miller, J. Mukerji, MDA Guide. Object Management Group, Inc., Version 1.0.1, omg/ 03-06-01 (2003)*
  http://www.omg.org/cgi-bin/apps/doc?omg/03-06-01.pdf
- K. Czarnecki, S. Helsen, *Feature-based survey of model transformation approaches. IBM Systems Journal* ***45****(3), pp. 621—645, July 2006.*
  http://www.research.ibm.com/journal/sj/453/czarnecki.html
- *T. Mens, P. Van Gorp, A Taxonomy of Model Transformation. Electr. Notes Theor. Comput. Sci.* ***152, pp. 125—142, 2006.***
  ftp://ftp.umh.ac.be/pub/ftp_infofs/2005/GraMOT-taxonomy.pdf

Further reading:
Websites

- Open Model CourseWare:
  http://www.eclipse.org/gmt/omcw/resources/
- ModelWare Project (Videos):
  http://www.modelware-ist.org/
- Eclipse Modeling Framework (EMF):
  http://www.eclipse.org/modeling/emf/
- ATLAS Transformation Language (ATL):
  http://www.eclipse.org/m2m/atl/
- Planet MDE (links to portals, conferences and tools):
  http://planet-mde.org/
- MDE research within SSEL:
  http://ssel.vub.ac.be/ssel/research/mdd