



IBM Rational Software

Domain-Specific Modeling Languages and UML Profiles – An Introduction

Bran Selic
IBM Canada
bselic@ca.ibm.com

Overview

- **A little bit of MOF**
- **Domain-specific modeling languages (DSML)**
- **The UML profile mechanism**
- **UML profile designer's guide**

The Meta-Levels Game (1)

Is this a pipe?

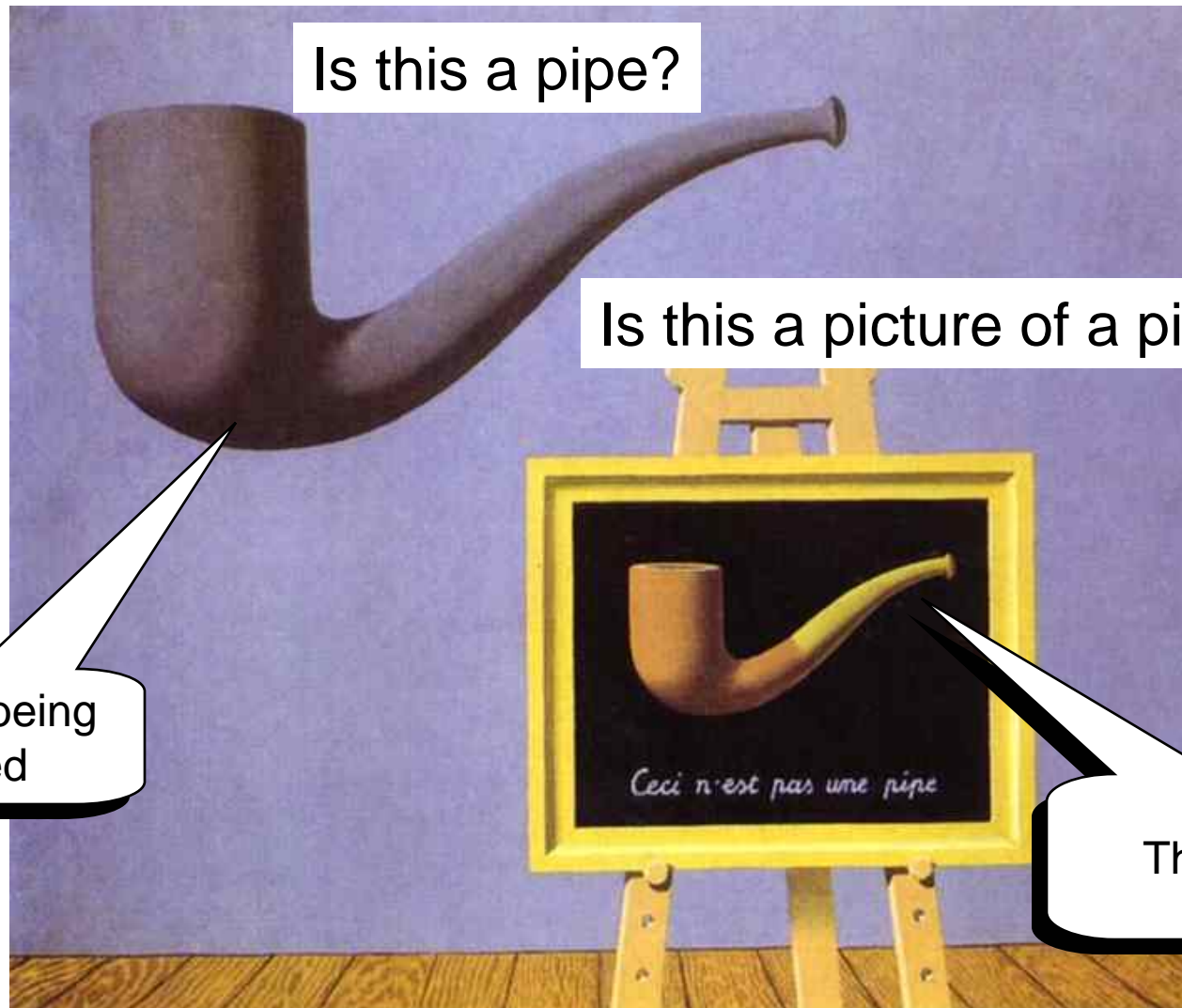


The Meta-Levels Game (2)

No, this is a picture of a pipe



The Meta-Levels Game (3)



Is this a pipe?

Is this a picture of a pipe?

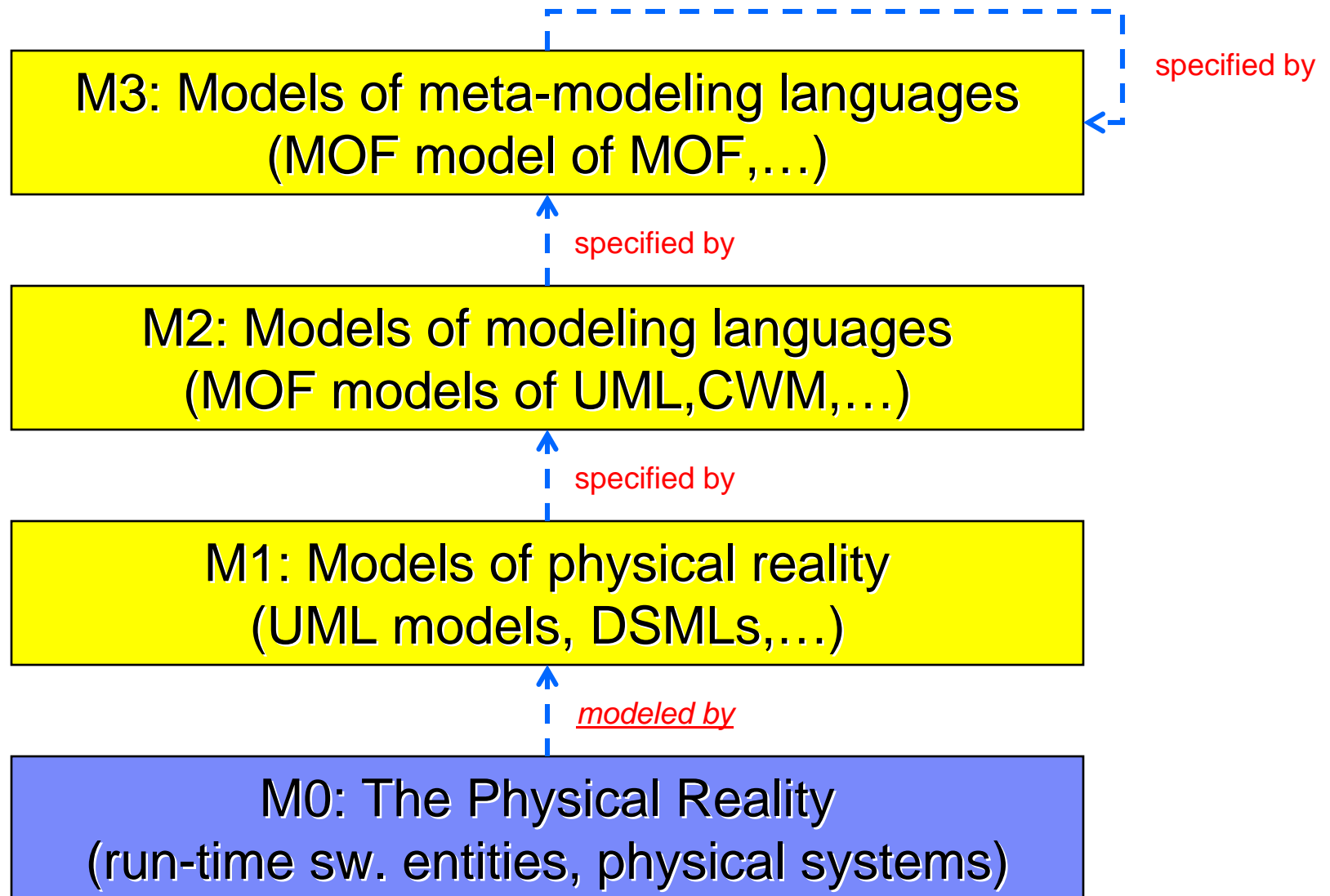
The thing being modeled

The model

The Meta-Levels Game: Conclusions

- **Meta-levels can be very confusing**
 - It is easy to confuse a specification/model with the actual thing that is being modeled
 - ...especially in informal colloquial usage
- **Meta-levels can be extended infinitely**
 - picture (model) of a picture (model) of a picture (model)...
- **There is a singularity at the origin (level 0)**
 - Physical reality is not a model
 - All levels above this level represent models

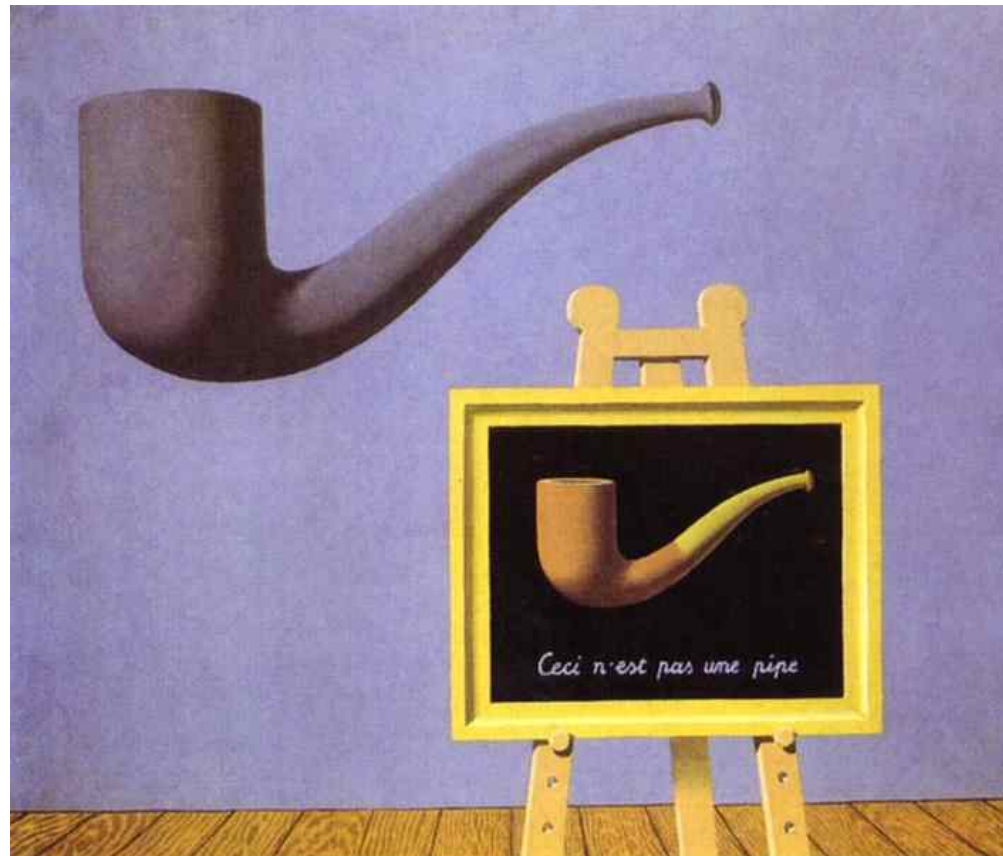
The OMG 4-Level Architecture



WARNING: The Infamous “Instance of” Term

- **This term is best avoided since it is used for a number of subtly different concepts**
 1. M_n -level entities are “instances of” their $M_{(n+1)}$ level specs
 - e.g., UML Association is an instance of a MOF Class
 2. An M_n -level model of an M_n -level class instance
 - e.g., a box in an object diagram labeled x : ClassQ
 3. A Java class is an “instance of” its UML model class
 - M1 to M0 relationships
 4. A Java object is an “instance of” its corresponding Java class
 - no modeling involved (similar to 1.)
 5. An attribute/variable that is an “instance of” of some type/class
 - = VAR x : ClassQ // x is an “instance of” ClassQ
- **If you want to use the term, then qualify it!**

Do Meta Levels “Really” Exist?



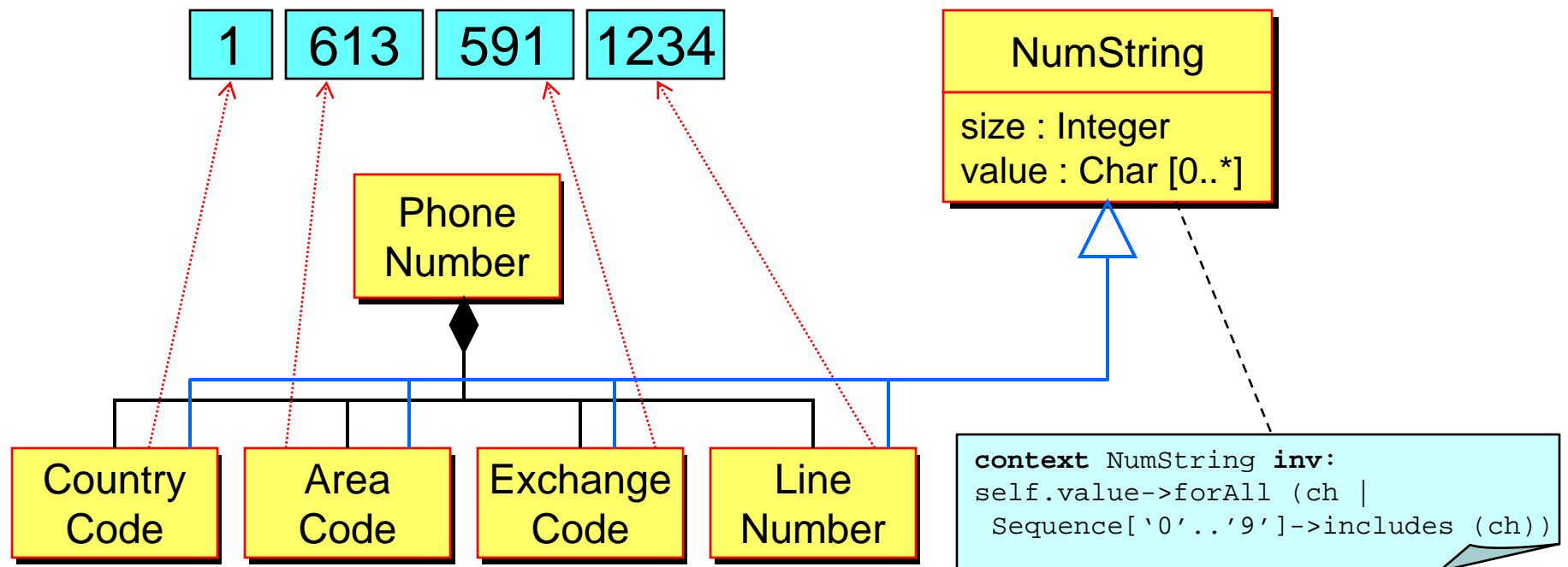
- There is no physical manifestation of a meta level (no hidden dimensions)
- The concept of meta levels is an intellectual aid to help us reason about the relationship between models and the things they model
 - It is possible to mix objects at different meta levels – key to reflective systems

MetaObject Facility (MOF)

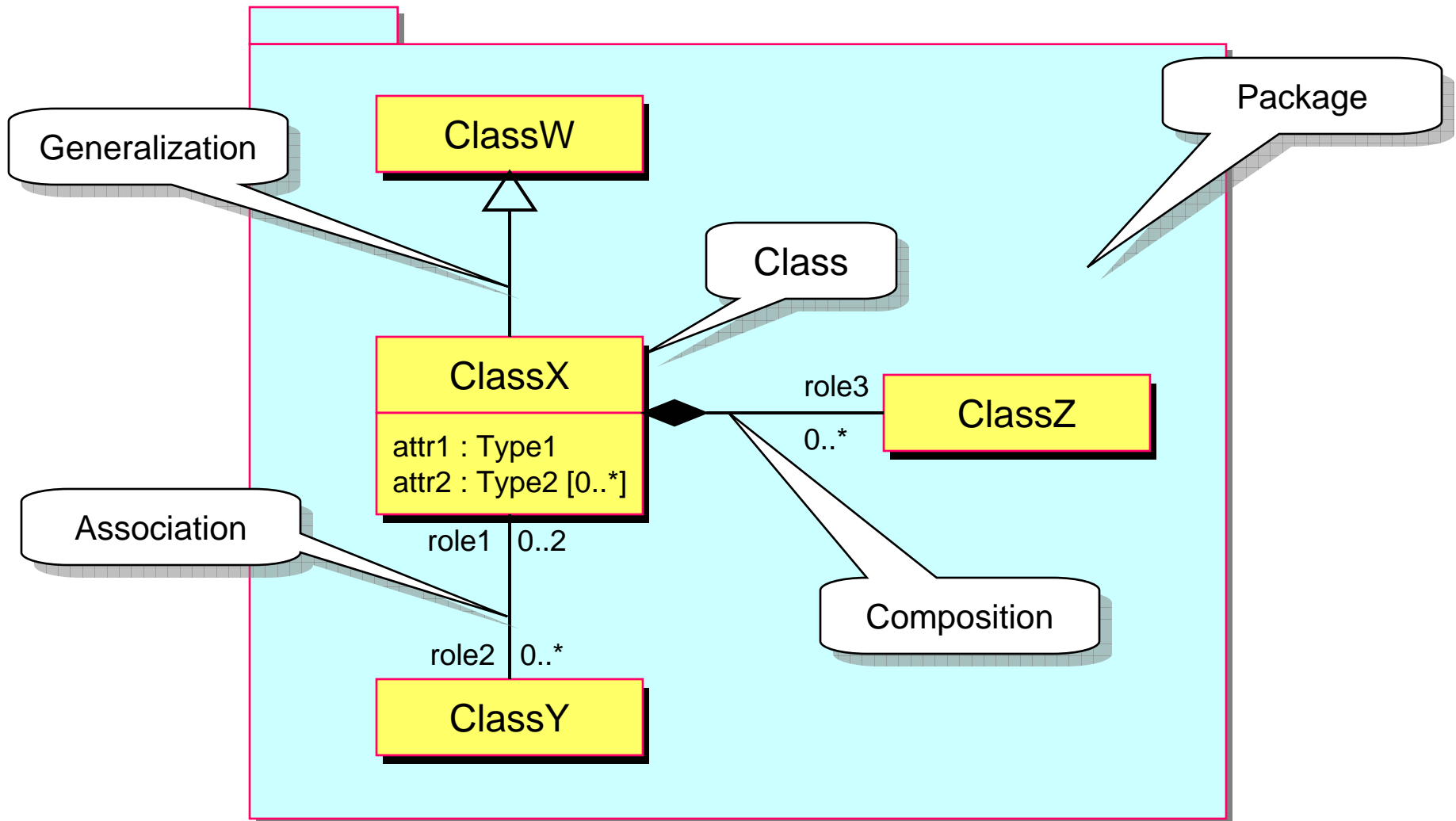
- **An OMG standard domain-specific modeling language (DSML) for defining metadata**
 - E.g. abstract syntax of a modeling language (well-formedness rules)
- **MOF domain**
 - Models stored in model repositories
 - Mostly static – minimal behavior (a few operations)
- **Technology independent**
 - Includes mappings to key technologies (XML [XMI], Java [JMI])
- **Self defining**
 - To avoid infinite regression of meta levels

Metadata

- A specification of the (mostly syntactic) structure of data
- Example (MOF)
 - Combination of Object Constraint Language (OCL) and a subset of UML concepts

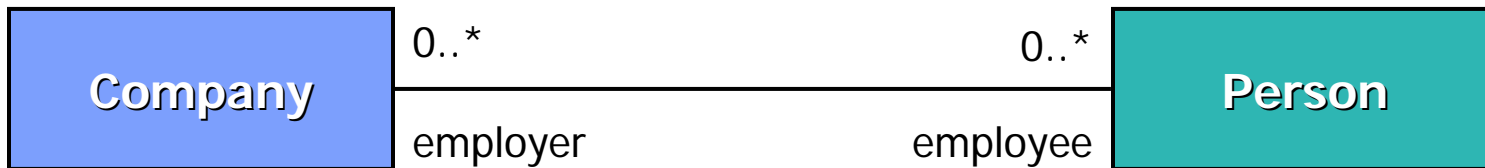


Key MOF Tools: Basic Concepts

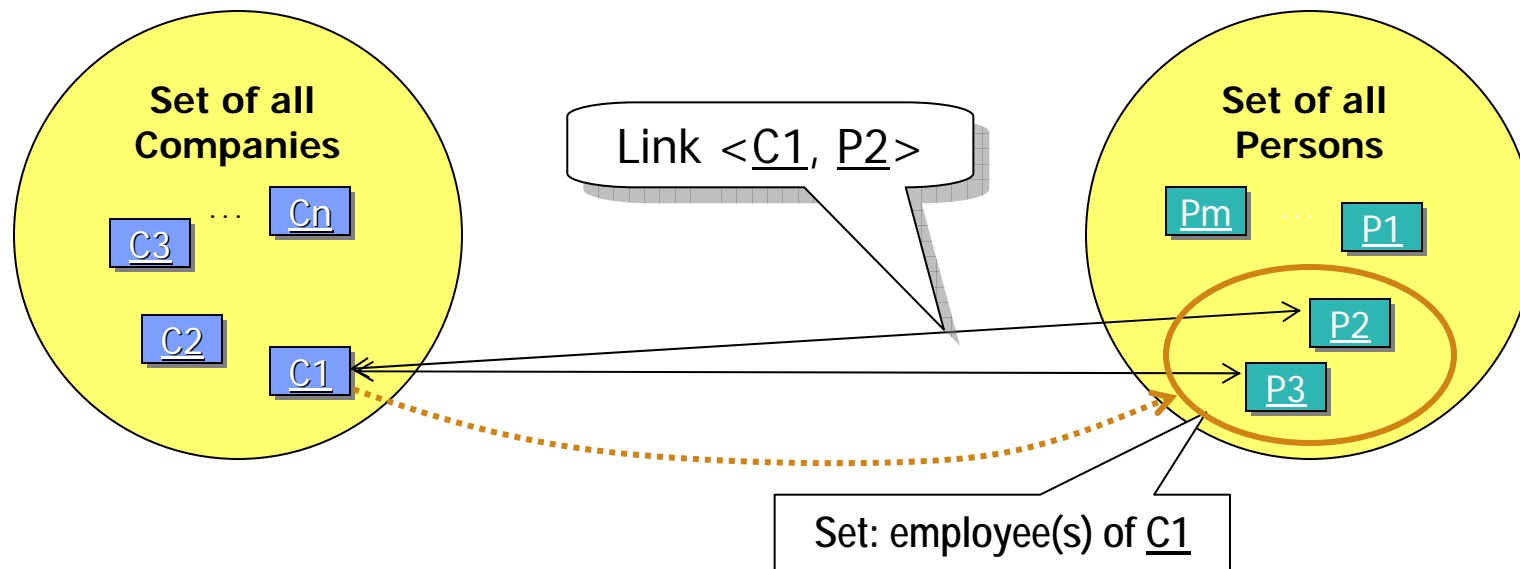


Class and Association Semantics

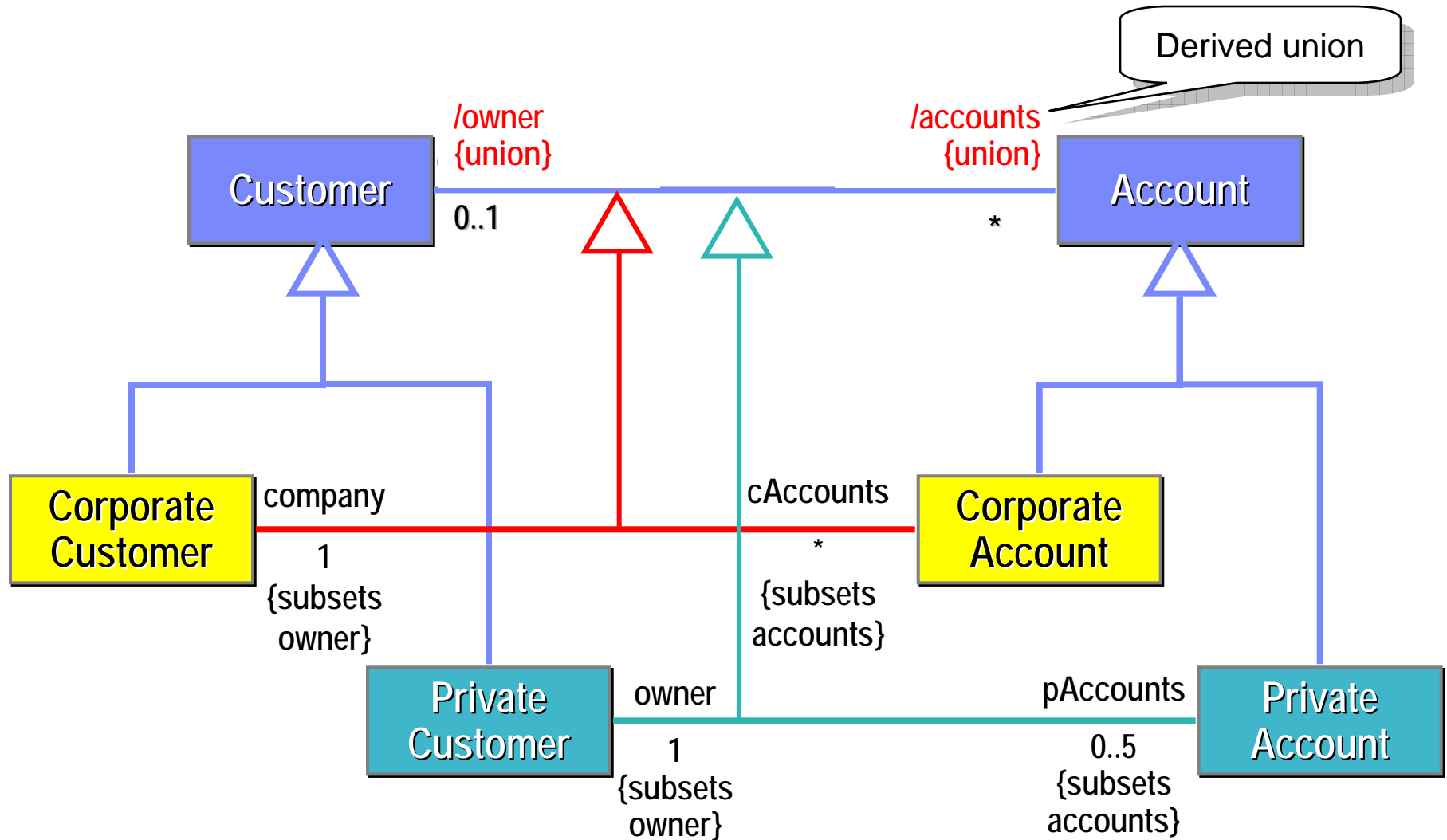
- Represent relationships between instances of classes



Formal (set theoretic) interpretation:

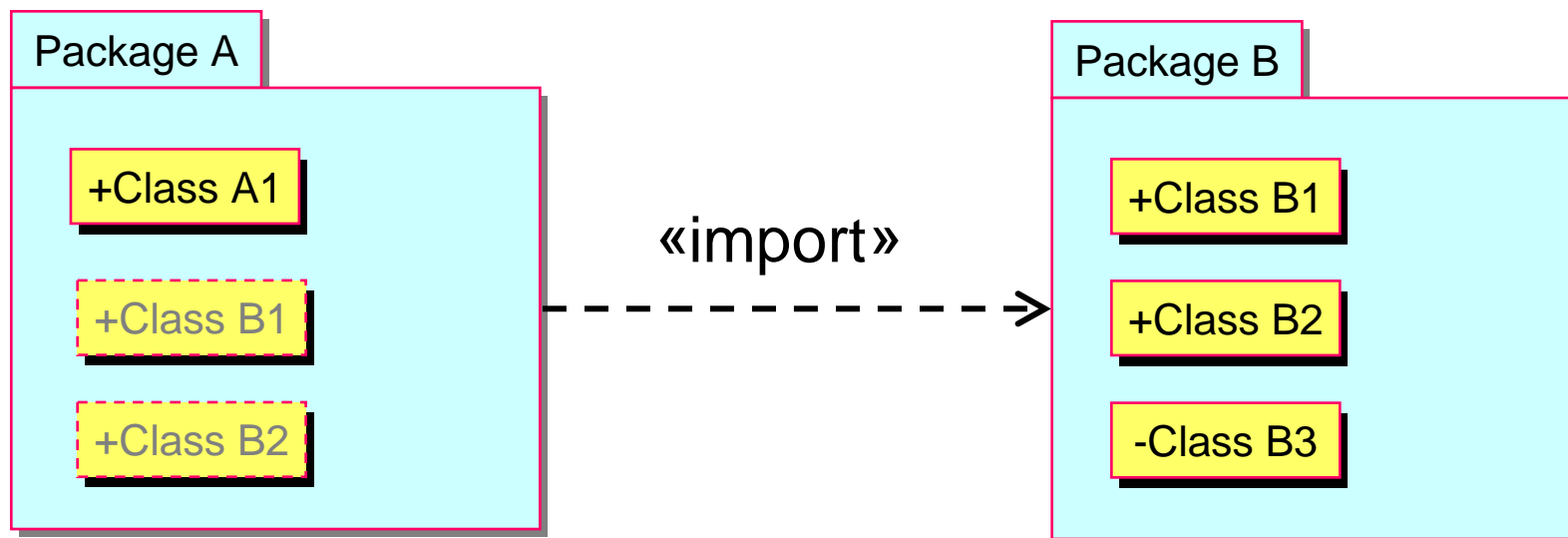


Key MOF Tools: Association Specialization



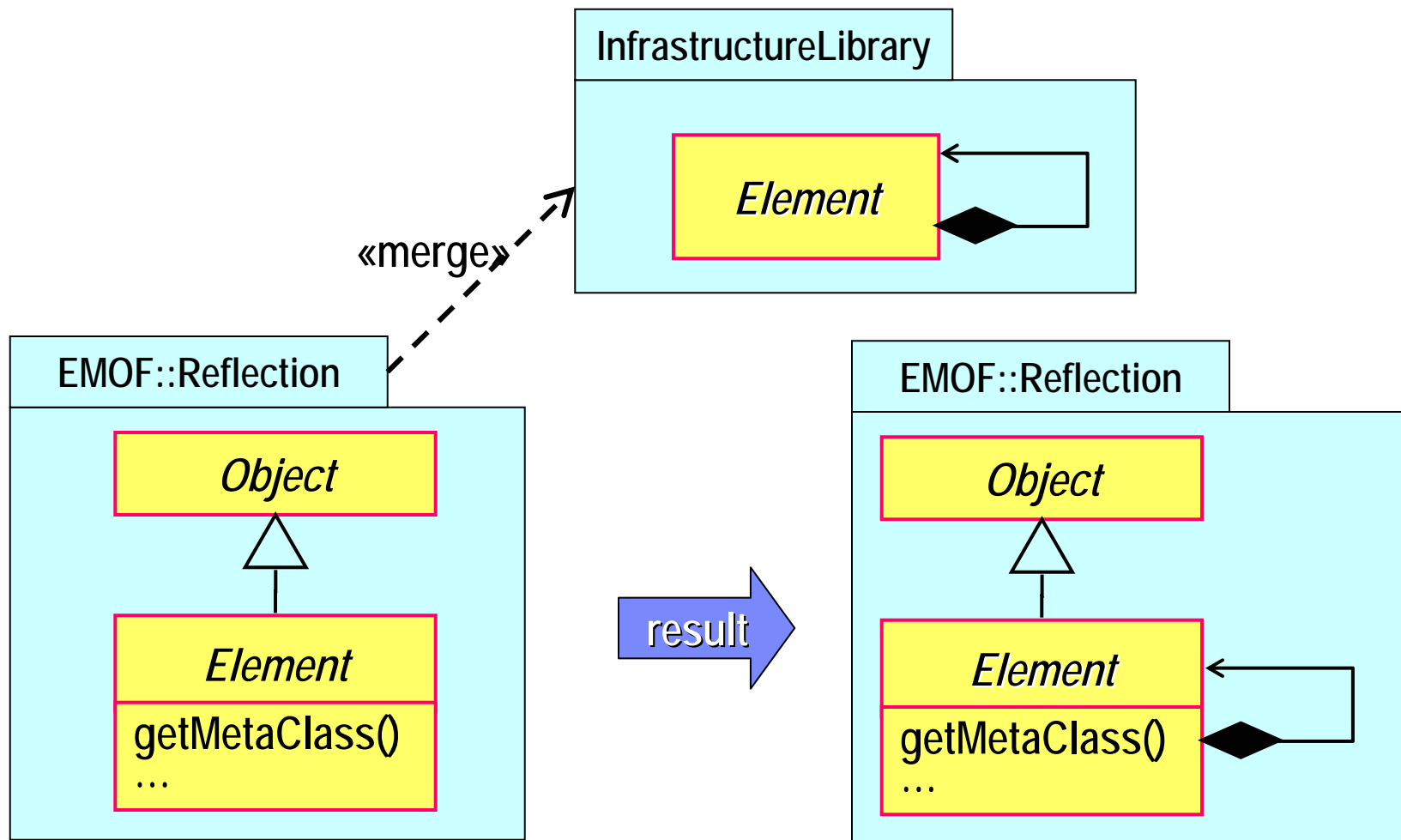
Key MOF Tools: Package and Element Import

- **MOF packages are namespaces**
 - Each element has a unique name and a visibility
 - It is possible to *import* publicly visible elements (classes, associations, etc.) from other packages
 - It is also possible to import individual elements (element import)



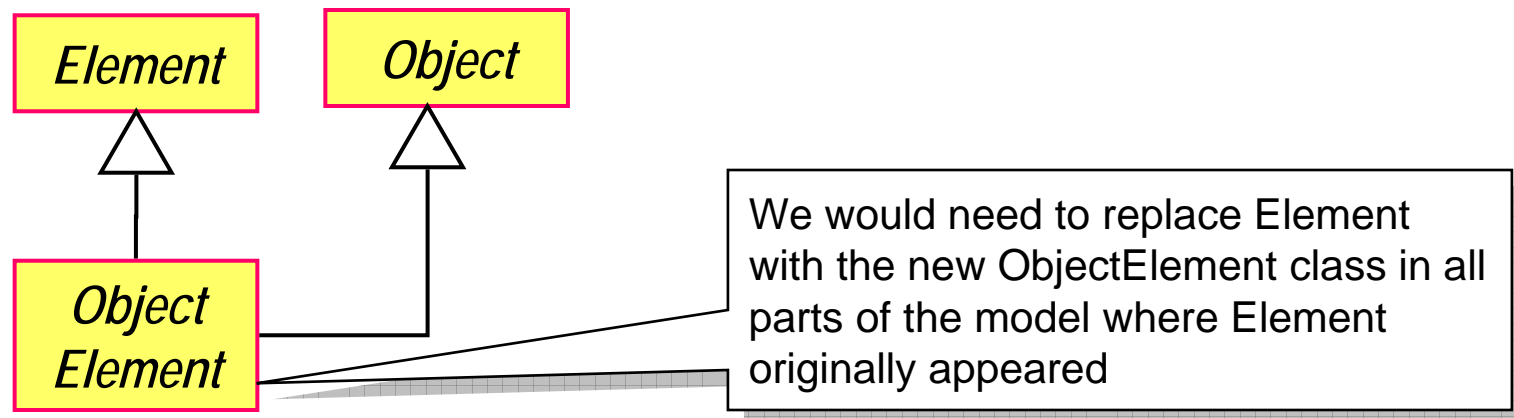
Key MOF Tools: Package Merge

- **Allows selective incremental concept extension**



Package Merge Semantics

- **A graphically-specified operation on packages and their contents**
 - Extends definition and scope of a concept through an increment
- **Why not just use subclassing?**
 - The additional specifications apply to the original concept wherever it was used



Summary

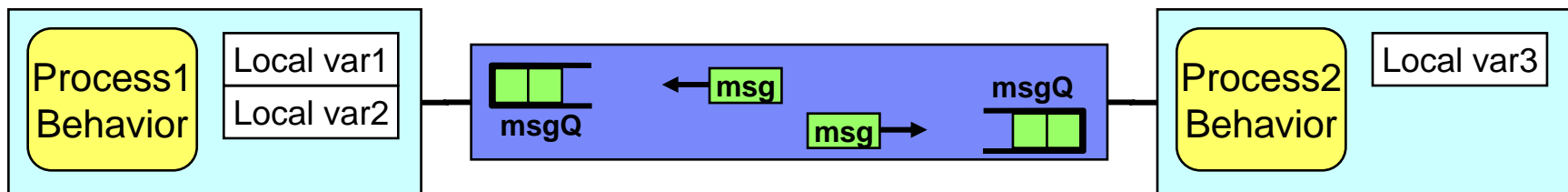
- **MOF is a key standard in the MDA lineup**
 - Used to define metadata: the structure of data
 - Particularly for defining the abstract syntax of modeling languages
 - ...including OMG standard languages such as UML, CWM
 - Also used to define specific technology mappings to XML (XMI) and Java (JMI)
- **MOF is also the basis for a set of related standards**
 - MOF Queries, Views, and Transformations
 - MOF Versioning and development lifecycle

Bibliography

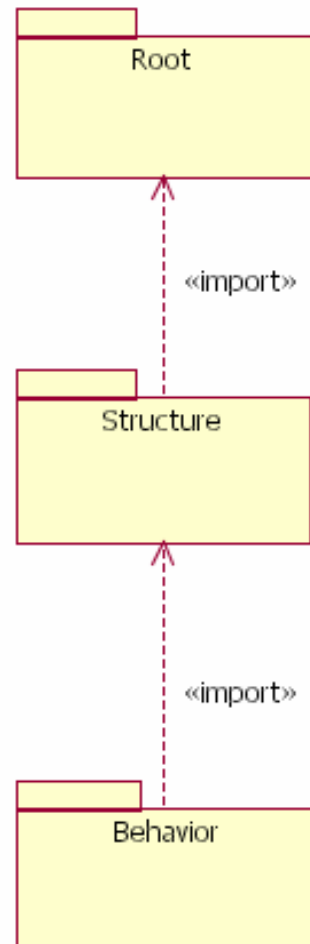
- **Meta Object facility (MOF) Core Specification – Version 2.0**
 - <http://www.omg.org/cgi-bin/doc?formal/2006-01-01>
- **Object Constraint Language (OCL) – Version 2**
 - <http://www.omg.org/cgi-bin/doc?formal/2006-05-01>
- **The UML 2.1 Infrastructure – Version 2.1**
 - <http://www.omg.org/cgi-bin/doc?ptc/2006-04-03>
- **MOF 2.0/ XMI Mapping Specification, v. 2.1**
 - <http://www.omg.org/cgi-bin/doc?formal/2005-09-01>
- **JSR-000040 Java Metadata Interface API Specification 1.0 Final Release**
 - <http://jcp.org/aboutJava/communityprocess/final/jsr040/index.htm>
- **A. Evans, et al., “Applied Metamodeling: A Foundation for Language-Driven Development,”**
 - A good book on metamodeling in general
 - <http://www.xactium.com/>

Exercise

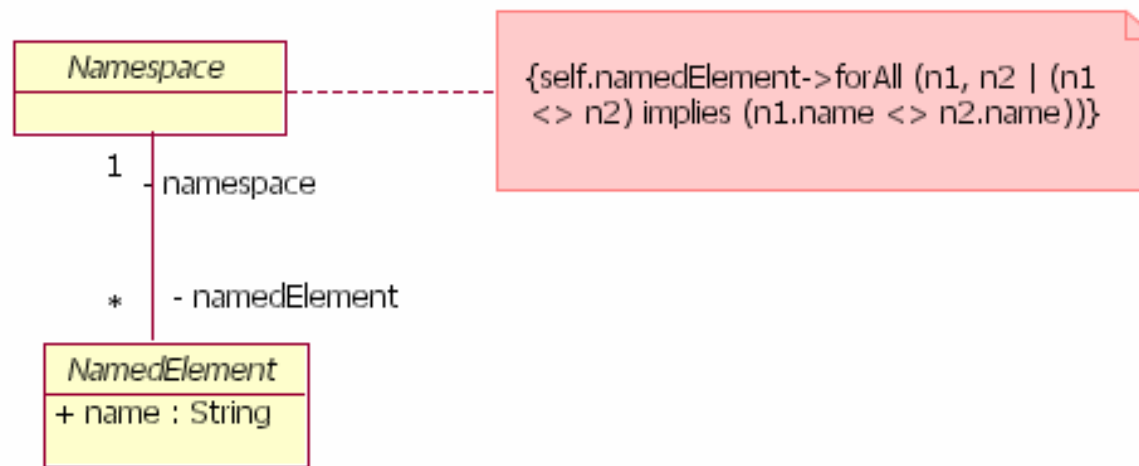
- **Define a metamodel for a CSP-like language**
 - Network of communicating processes interconnected by bidirectional channels
 - Communication by asynchronous message passing
 - Messages consist of typed data objects
 - A process can have local variables for storing data objects and messages



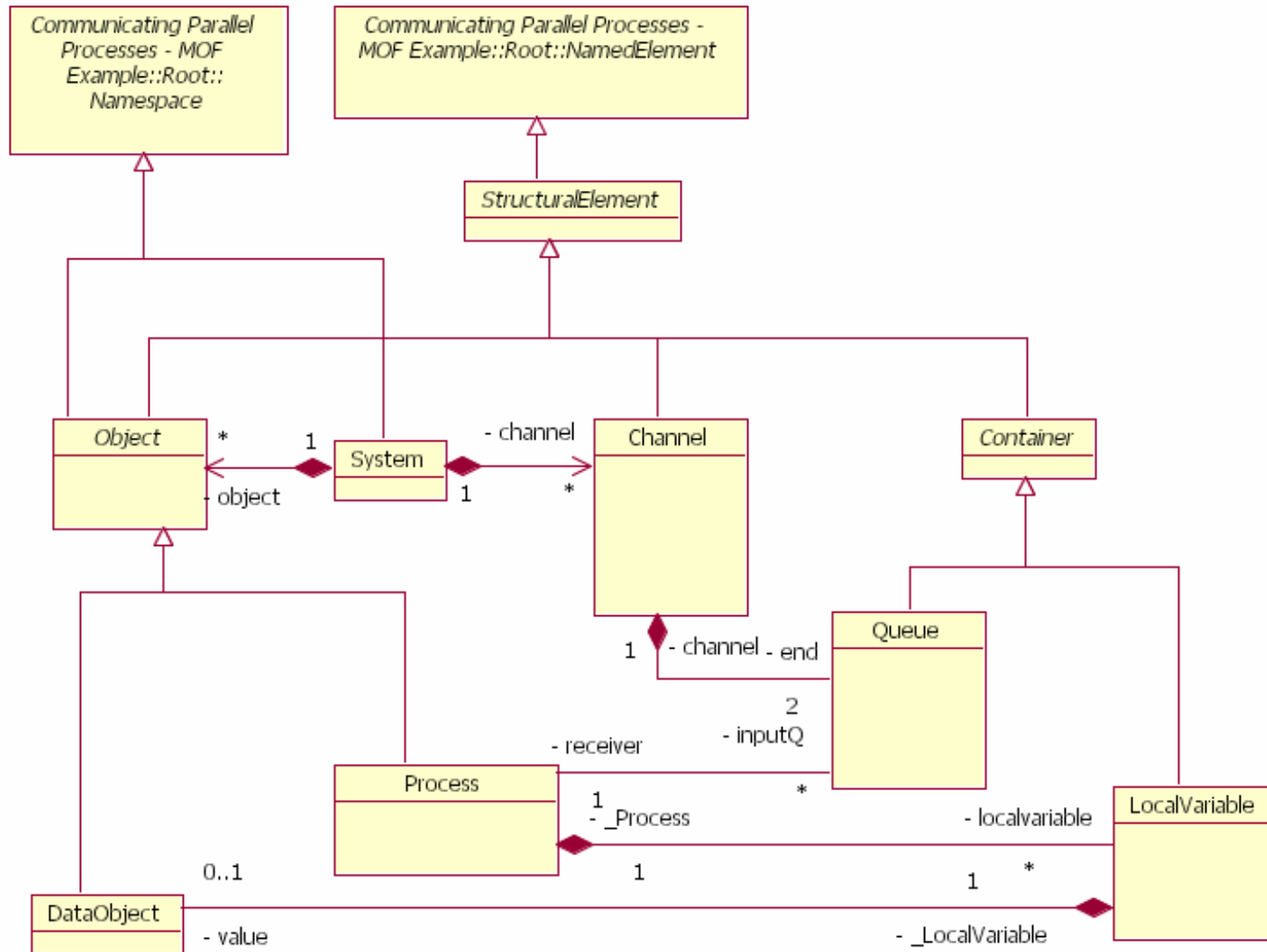
Exercise: A Solution – The Package Hierarchy



Exercise: A Solution – The Root Package



Exercise: A Solution – The Structure Package



Overview

- A little bit of MOF
- **Domain-specific modeling languages (DSML)**
- The UML profile mechanism
- UML profile designer's guide
- Example

Specialization – a Characteristic of Our Time

- **Constant branching of domains into ever-more specialized sub-domains**
 - E.g.: Computer programming emerged as a sub-discipline of engineering into a complex of diverse (and often overlapping) disciplines
 - Domain concepts become more and more refined where it becomes critical to be able to clearly differentiate seemingly subtle semantic distinctions
 - E.g., concept of “computer memory”
 - Virtual/physical, primary/secondary, cache/main, transient/persistent, read-write/read-only, dynamic/static, ...
- **Clearly, this trend needs to be reflected in the computer languages used to specify applications in various domains**

Specialized Computer Languages

- **Literally hundreds of domain-specific programming languages have been defined over the past 50 years**
 - Fortran: for scientific applications
 - COBOL for “data processing” applications
 - Lisp for AI applications
 - etc.
- **Some trends**
 - Many of the original languages are still around
 - More often than not, highly-specialized domains still tend to use general-purpose languages with specialized domain-specific program libraries instead of domain-specific programming languages
 - In fact, the trend towards defining new domain-specific programming languages seems to be diminishing
 - Why?

Basic Criteria for Success of a Computer Language

- **Technical validity**: absence of major design flaws and constraints (ease of writing correct programs)
- **Expressiveness**: ability to succinctly specify the necessary domain concepts
- **Simplicity**: absence of complexity (eases learning)
- **Efficiency**: potential to minimize space and performance overheads
- **Familiarity**: proximity to widely-available skills sets
- **Interoperability**: language compatible with other technologies
- **Support**: availability of the infrastructure required for effective exploitation
 - Availability of tools (editors, compilers, debuggers, static and dynamic analyzers, build tools, version control tools, merge/diff tools, etc.)
 - Effectiveness of tools
 - Availability of program libraries
 - Availability of skilled practitioners
 - Availability of textbooks and training courses
 - Capacity for evolution and maintenance

Some Important Conclusions

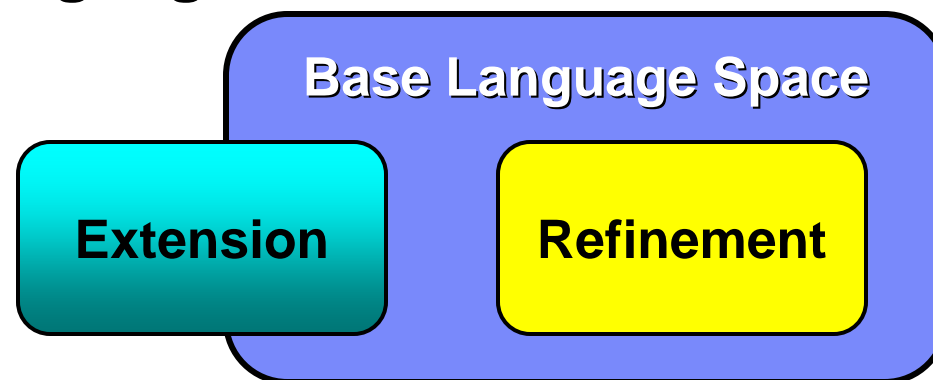
- **Designing a useful domain-specific computer language (modeling or programming) is hard**
 - In addition to domain expertise, it requires language design expertise
 - There is no established comprehensive theory of modeling language design to guide the designer
- **If the support infrastructure is inadequate, the language may not be viable**
 - Despite potential technical excellence

Domain-Specific Modeling Languages (DSML)

- **Modeling languages intended for a specific application domain**
- **Three different approaches to defining a DSML**
 - Define a new language: from scratch
 - Extend an existing language: add new domain-specific concepts to an existing language
 - Refine an existing language: specialize the concepts of an existing language

The “Semantic Space” of a Computer Language

- **Semantic space** = the set of all valid programs that can be specified with a given computer language
- **Refinement**: subsets the semantic space of the base language
 - Enables reuse of base-language tools
- **Extension**: intersects the semantic space of the base language



Comparing the Approaches

- **New language**
 - Pro: potential for maximum expressive power
 - Con: Requires language design skills
 - Con: No language support infrastructure (tools, etc.)
- **Extension of an existing language**
 - Pro: requires less language design skills
 - Con: little or no reuse of language support infrastructure
- **Refinement of an existing language**
 - Pro: reuse of language support infrastructure
 - Pro: requires less language design skills
 - Con: expressive power constrained by base language

Overview

- **A little bit of MOF**
- **Domain-specific modeling languages (DSML)**
- **The UML profile mechanism**
- **UML profile designer's guide**

UML as a Platform for DSMLs

- **Designed as a “family of modeling languages”**
 - Contains a set of semantic variation points (SVPs) where the full semantics are either unspecified or ambiguous
 - SVP examples:
 - Precise type compatibility rules
 - Communications properties of communication links (delivery semantics, reliability, etc.)
 - Multi-tasking scheduling policies
 - Enables domain-specific customization
- **Open to both extension (“heavyweight” extension) and refinement (“lightweight” extension)**

Customizing UML

- **Heavyweight/extension**

- Requires adding new concepts (classes) and/or relationships (associations) to the UML metamodel using MOF
- Example: Adding a Petri-net behavioral formalism to UML

- **Lightweight/refinement**

- *Refinements must be formally consistent with base UML semantics and well-formedness rules!*
- Specified using the built-in UML extension mechanisms:
 - Profiles
 - Stereotypes
 - Constraints
 - Model libraries(*)

Example: Adding a Semaphore Concept to UML

- **Semaphore semantics:**

A specialized object that limits the number of concurrent accesses in a multithreaded environment. When that limit is reached, subsequent accesses are suspended until one of the accessing threads releases the semaphore, at which point the earliest suspended access is given access.

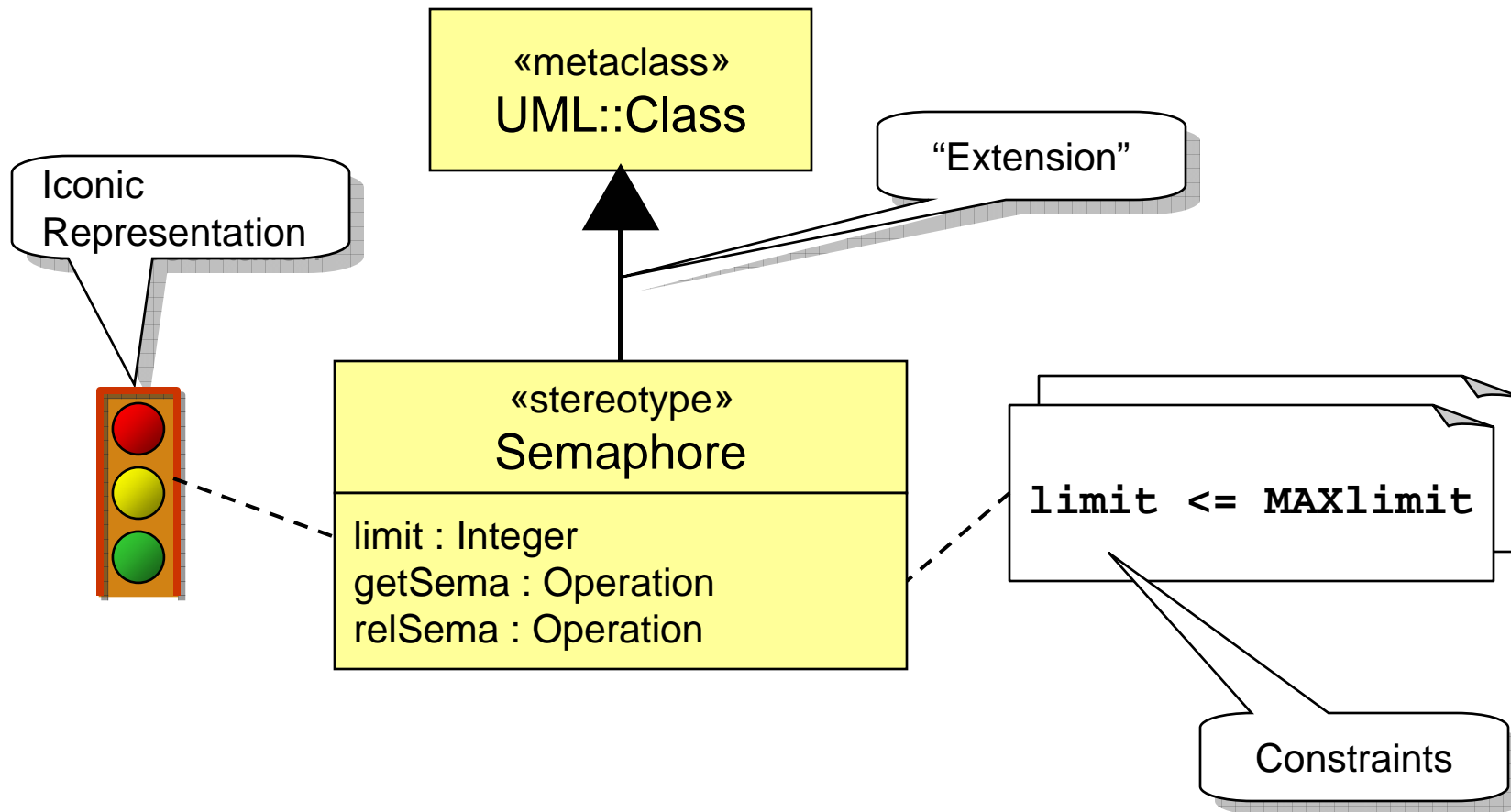
- **What is required is a special kind of object**

- Has all the general characteristics of UML objects
- ...but adds refinements

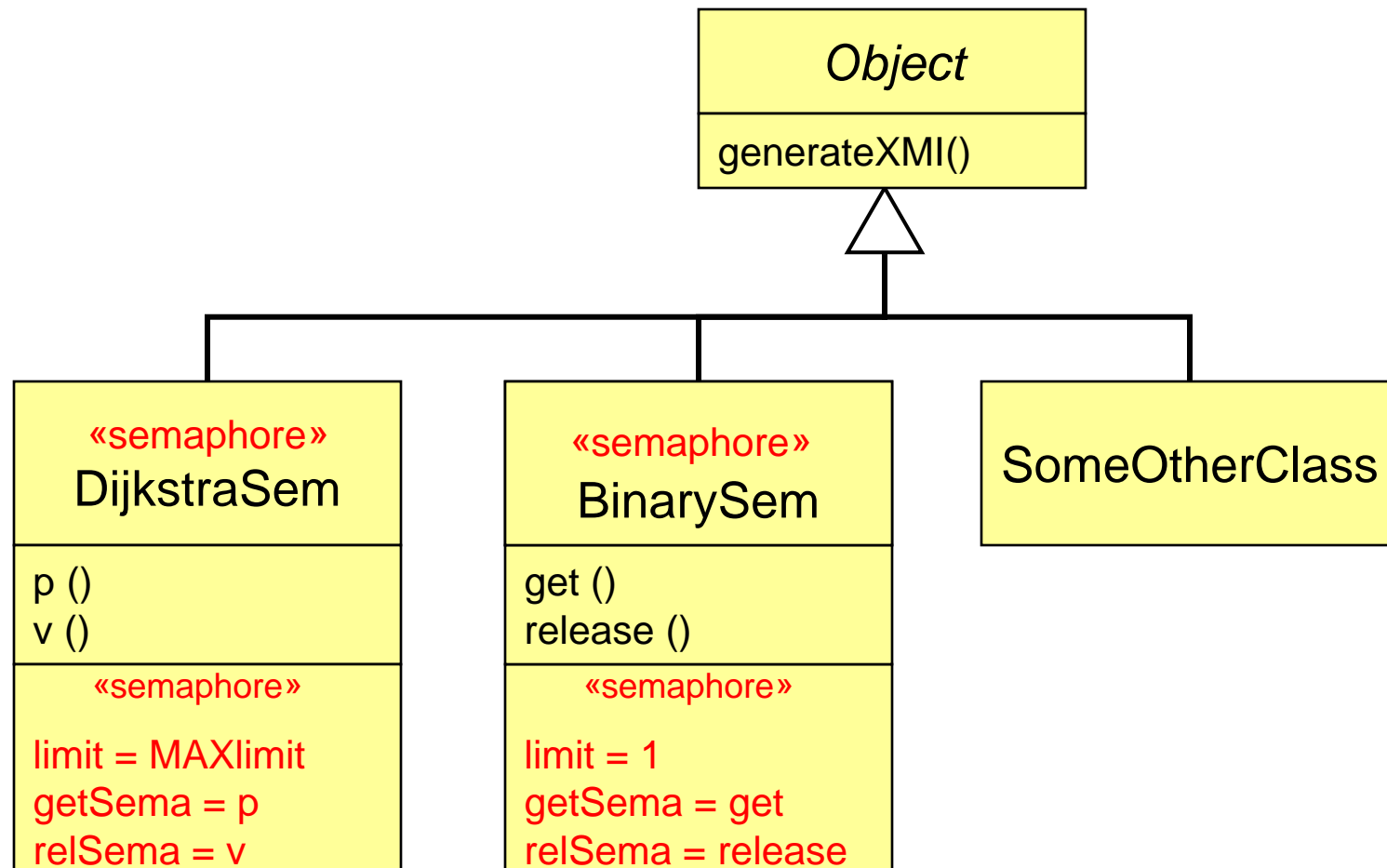
Example: The Semaphore Stereotype

- **Design choice: Refine the UML Class concept by**
 - “Attaching” semaphore semantics
 - Done informally as part of the stereotype definition
 - Adding constraints that capture semaphore semantics
 - E.g., when the maximum number of concurrent accesses is reached, subsequent access requests are queued in FIFO order
 - Adding characteristic attributes (e.g., concurrency limit)
 - Adding characteristic operations (getSemaphore (), releaseSemaphore ())
- **Create a new “subclass” of the original metaclass with the above refinements**
 - For technical reasons, this is done using special mechanisms instead of MOF Generalization (see slide [Why are Stereotypes Needed?](#))

Example: Graphical Definition of the Stereotype



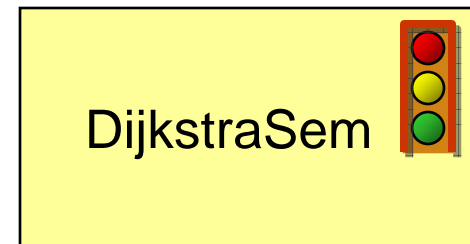
Example: Applying the Stereotype



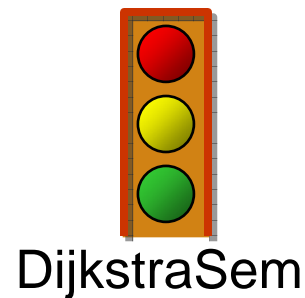
Example: Stereotype Representation Options



(a)



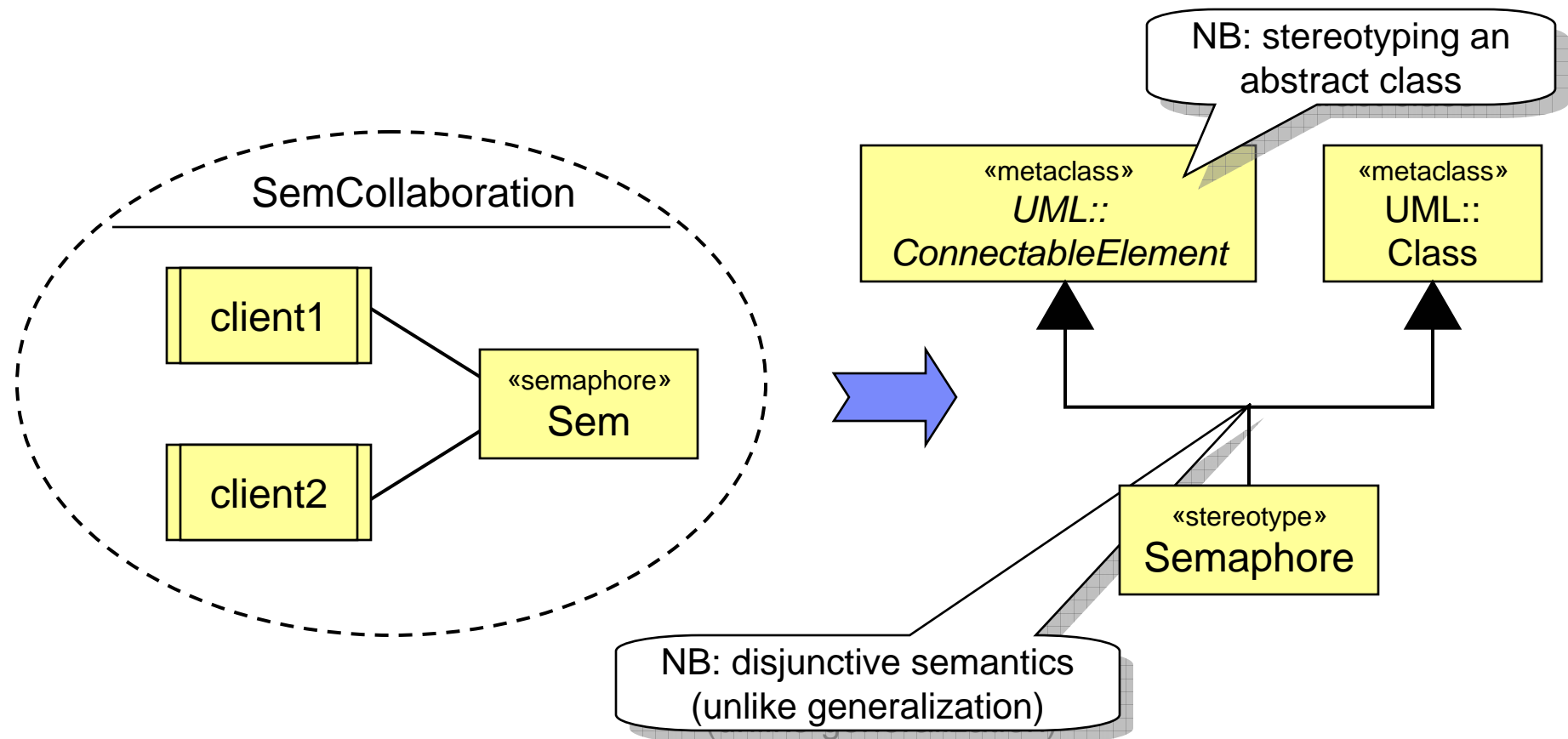
(b)



(c)

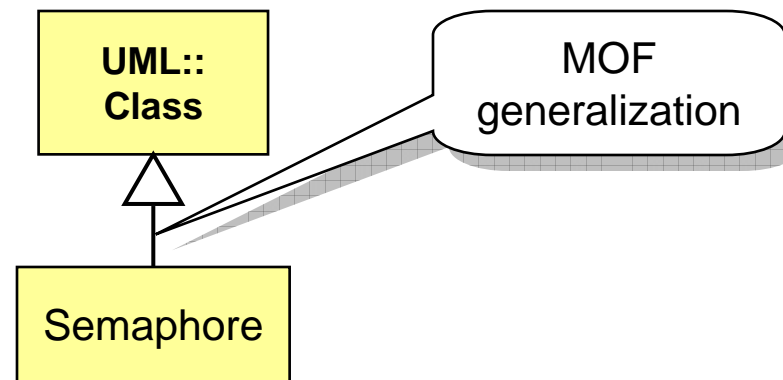
Example: Extending the Scope of a Stereotype

- It is common to associate a stereotype with different kinds of metamodel elements



Why are Stereotypes Needed?

- **Why not simply create a new metaclass?**

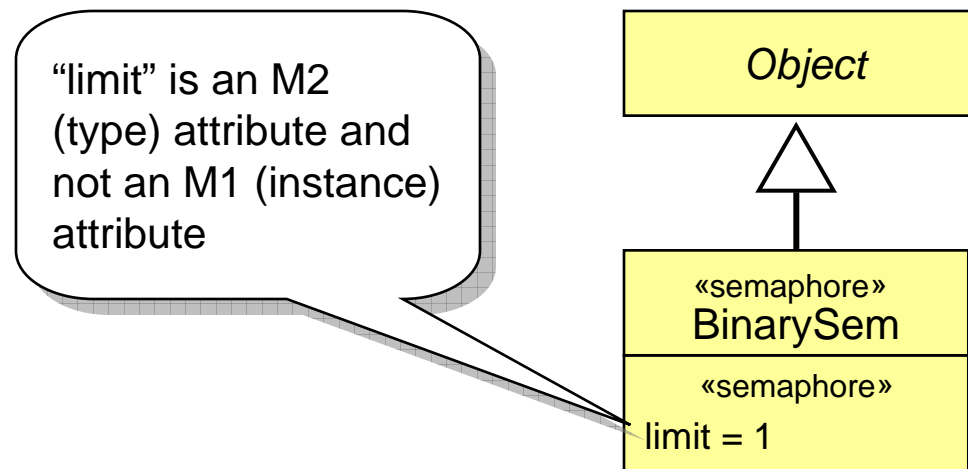


Rationale:

1. Not all modeling tools support meta-modeling \Rightarrow need to define (M2) extensions using (M1) models
2. Need for special semantics for the extensions:
 - multiple extensions for a single stereotype
 - extension of abstract classes (applicable to all subclasses)

Meta-Levels Revisited

- **NB: The attributes of a stereotype are meta-level (M2) attributes and not model-level (M1) attributes**
- **However, because each stereotype instance can have unique tagged values attached, the mechanism is often used to specify model-level characteristics**



The MOF Semantics of UML Extension

- How a stereotype is attached to its base class within a model repository:

“Base” metaclass

Stereotype

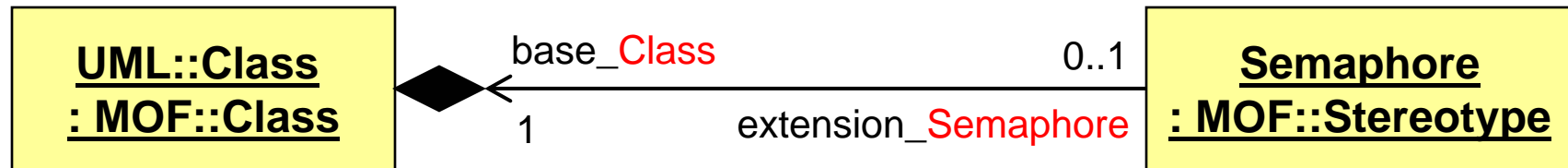


- Association ends naming convention:**
`base_<base-class-name>`
`extension_<stereotype-name>`
- Required for writing correct OCL constraints for stereotypes

Example: OCL Constraint for a Stereotype

“Base” metaclass

Stereotype



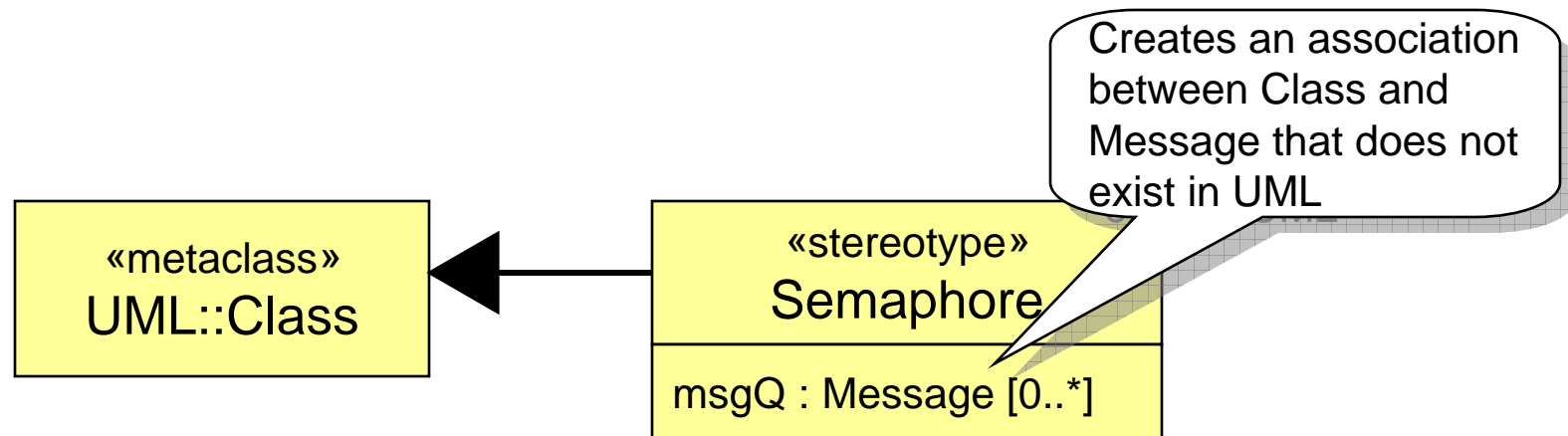
- Semaphore constraint:
the base Class must have an owned ordered attribute called “msgQ” of type Message

```

context Semaphore inv:
  self.base_Class.ownedAttribute->
    exists (a | (a.name = 'msgQ')
      and (a.type->notEmpty())
      and (a.type = Message)
      and (a.isOrdered)
      and (a.upperValue = self.limit))
  
```

Adding New Meta-Associations

- **This was not possible in UML 1.x profiles**
 - Meta-associations represent semantic relationships between modeling concepts
 - New meta-associations create new semantic relationships
 - Possibility that some tools will not be able to handle such additions
- **UML 2.0 capability added via stereotype attribute types:**
 - *To be used with care!*

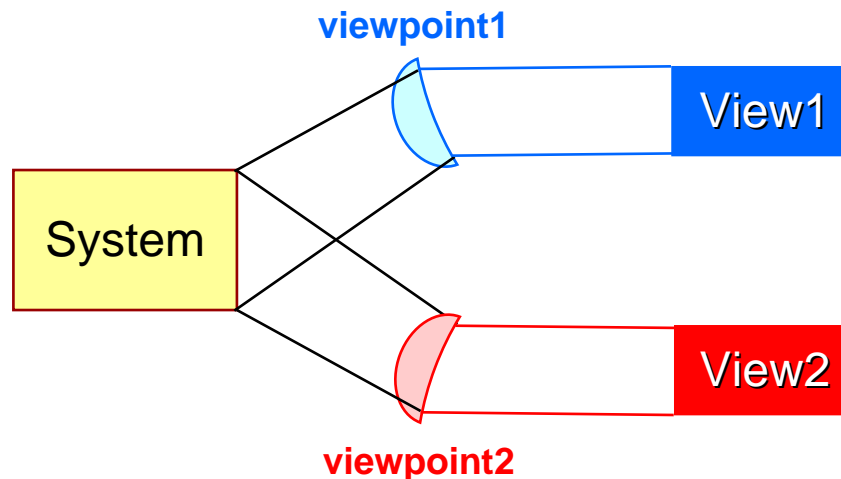


UML Profiles

- **Profile:**
 - *A special kind of package containing stereotypes and model libraries that, in conjunction with the UML metamodel, define a group of domain-specific concepts and relationships*
 - The profile mechanism is also available in MOF where it can be used for other MOF-based languages
- **Profiles can be used for two different purposes:**
 1. To define a domain-specific modeling language
 2. To define a domain-specific viewpoint

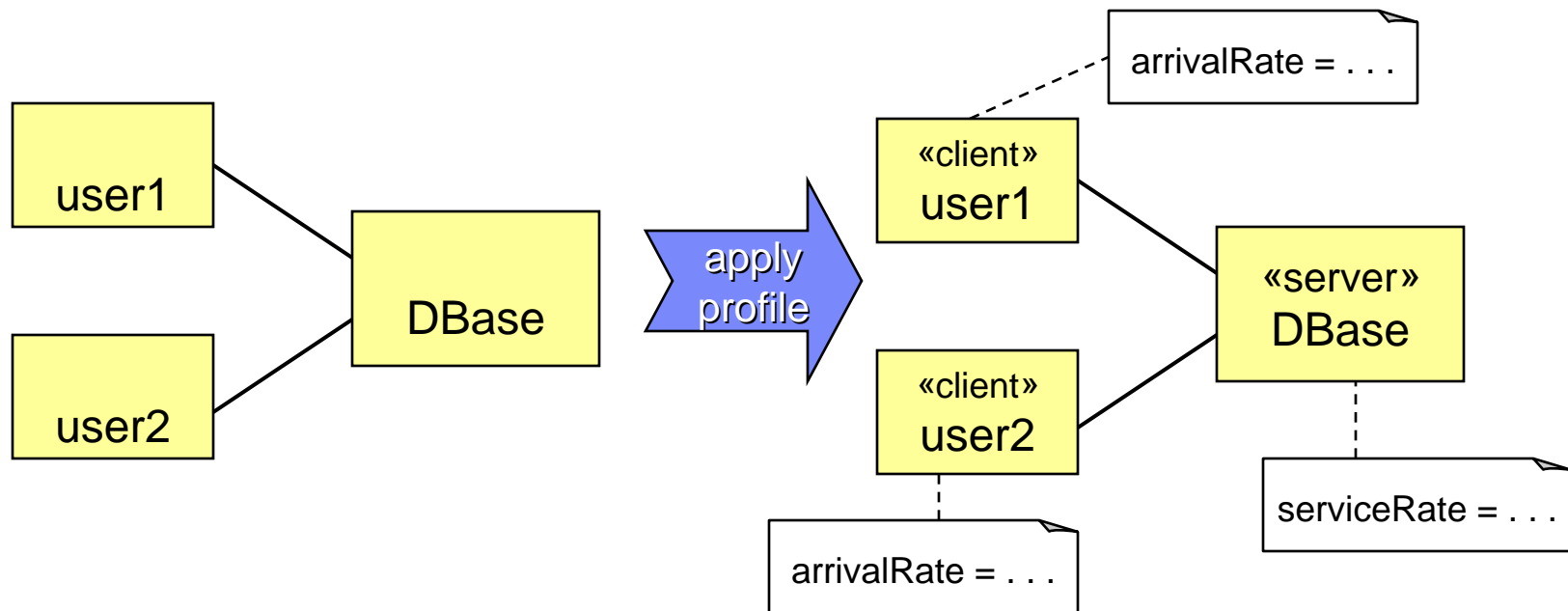
Views and Viewpoints (IEEE 1471)

- **View**: A representation of a whole system from the perspective of a related set of concerns
 - Relevance depends on the interests of the viewer
 - *A view does not affect the system being viewed*
- **Viewpoint**: A pattern or template from which to develop individual views by establishing the purposes and audience for a view and the techniques for its creation and analysis



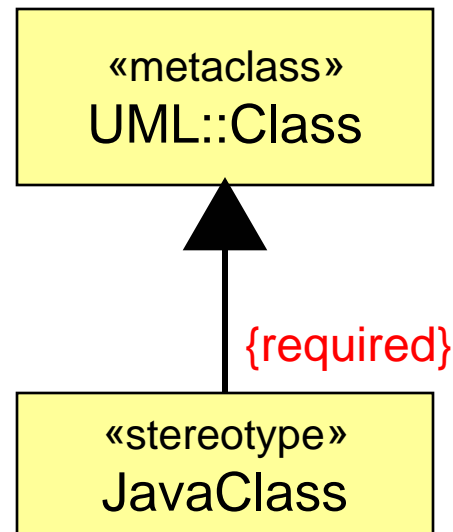
Profiles as Viewpoints

- A profile can be used as an overlay mechanism that can be dynamically applied or “unapplied” to provide a desired view of an UML model
 - Allows a UML model to be interpreted from the perspective of the viewpoint definer
- **NB: Applying or unapplying profiles has no effect on the underlying model**
- **Example: viewing a UML model fragment as a queueing network to do performance analysis**



“Required” Extensions

- **An extension can be marked as “required”**
 - Implies that every instance of the base class will be stereotyped by that stereotype
 - Used by modeling tools to autogenerate the stereotype instances
 - Facilitates working in a DSML context by avoiding manual stereotyping for every case
 - E.g., modeling Java



Strict Profile Application

- **A *strict* application of a profile will hide from view all model elements that do not have a corresponding stereotype in that profile**
 - Convenient for generating views
 - NB: *This does not change the underlying model in any way*
 - (In contrast, marking extensions “required” does)
- **Strictness is a characteristic of the profile application and not of the profile itself**
 - Any given profile can be applied either way

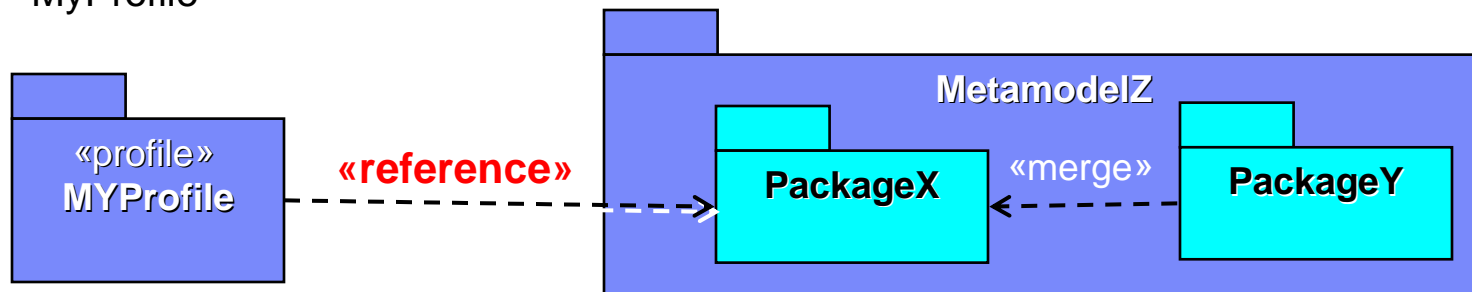
Metamodel Subsetting with Profiles (1)

- **It is often useful to remove segments of the full UML metamodel resulting in a minimal DSML definition**
 - NB: Different mechanism from strict profile application – the hiding is part of the profile definition and cannot be applied selectively
- **The UML 2.1 profile mechanism adds controls that define which parts of the metamodel are used**
 - Based on refinement of the package import and element import capabilities of UML

Metamodel Subsetting with Profiles (2)

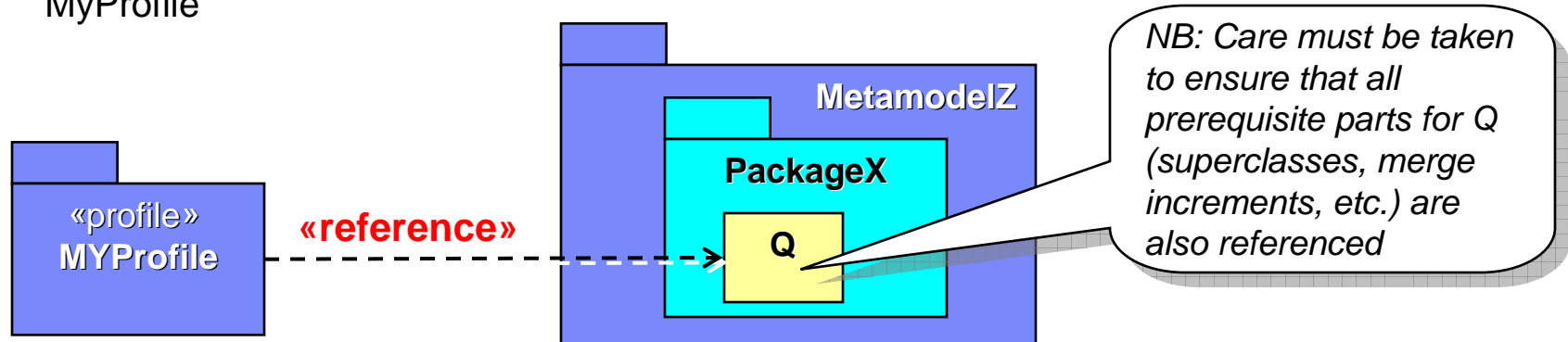
Case 1: Metamodel Reference

- All elements of the referenced MOF package (PackageX) are visible (but not the elements of PackageY)
- These elements can also serve as the base metaclasses for stereotypes in MyProfile



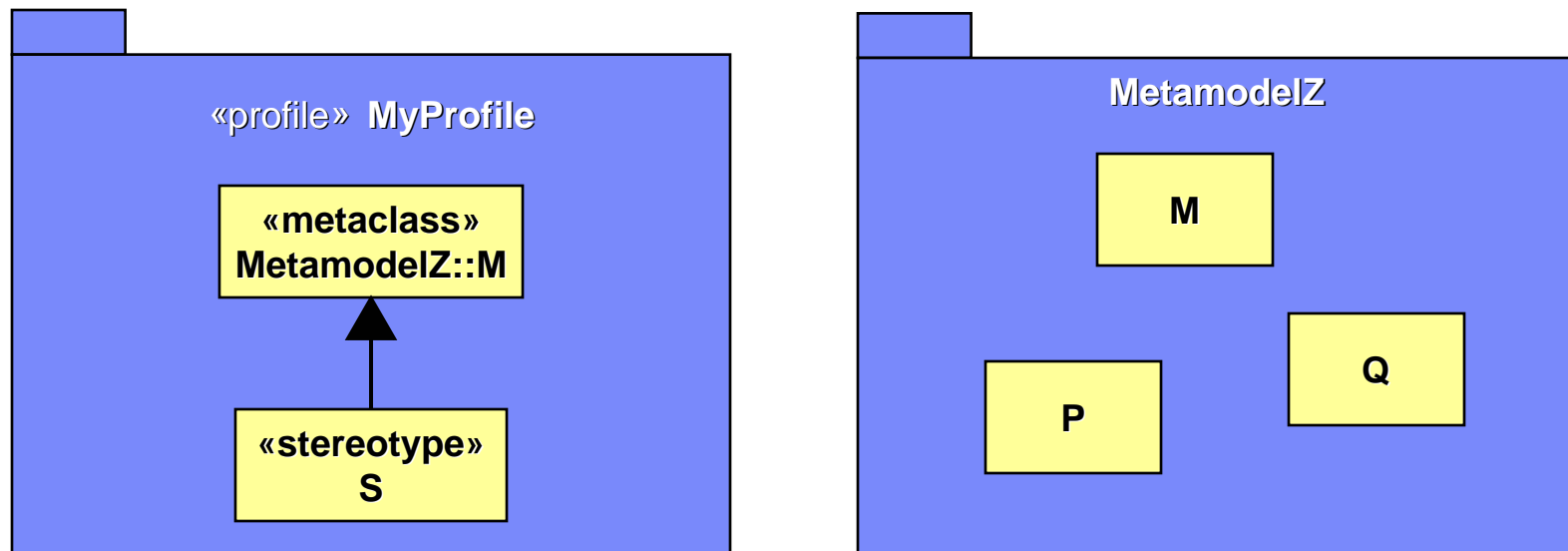
Case 2: Explicit Metaclass Reference

- Metaclass Q is visible and can serve as a base metaclass for stereotypes in MyProfile



Metamodel Subsetting with Profiles (3)

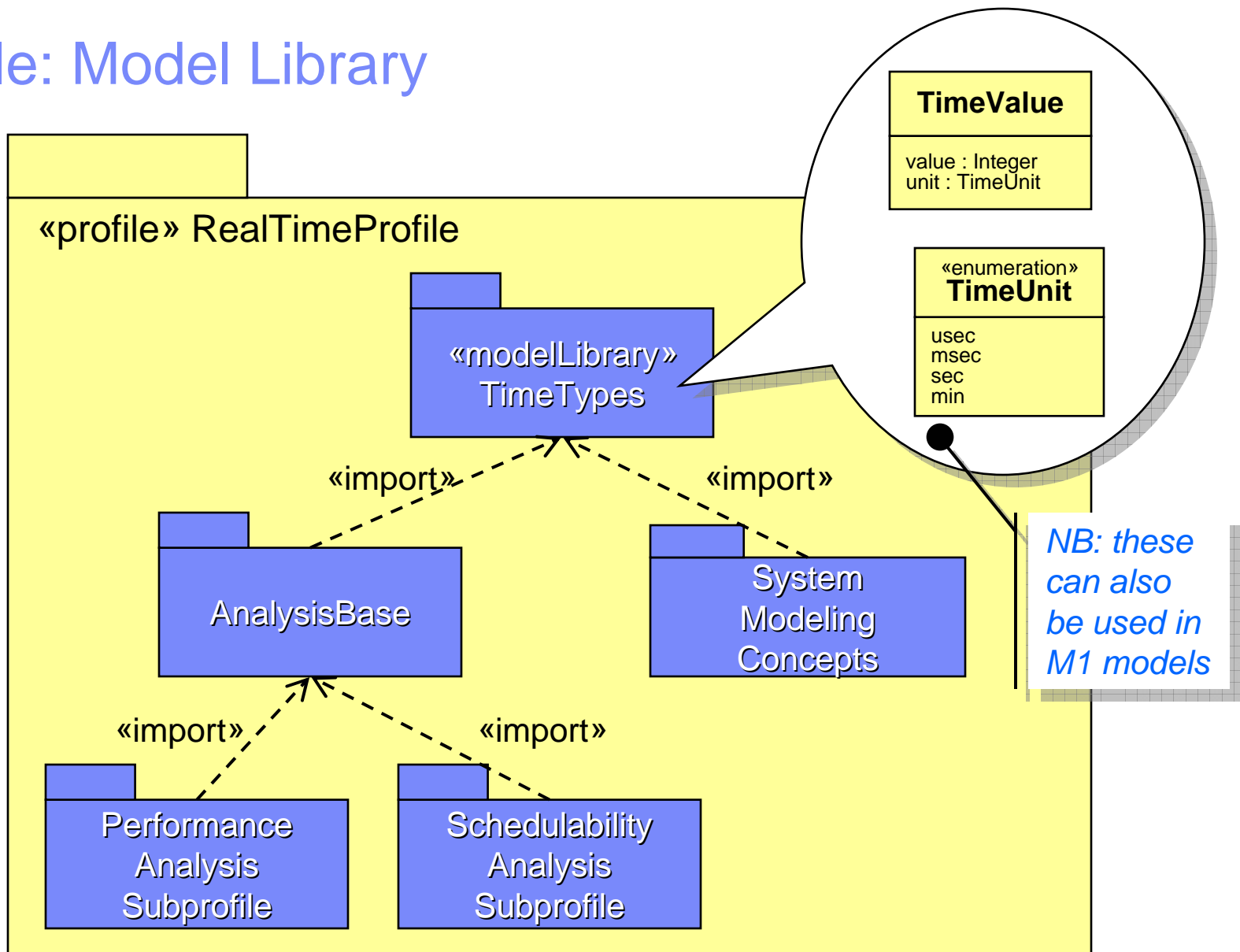
- **Case 3: Implicit metaclass reference**
 - Metaclass M is visible



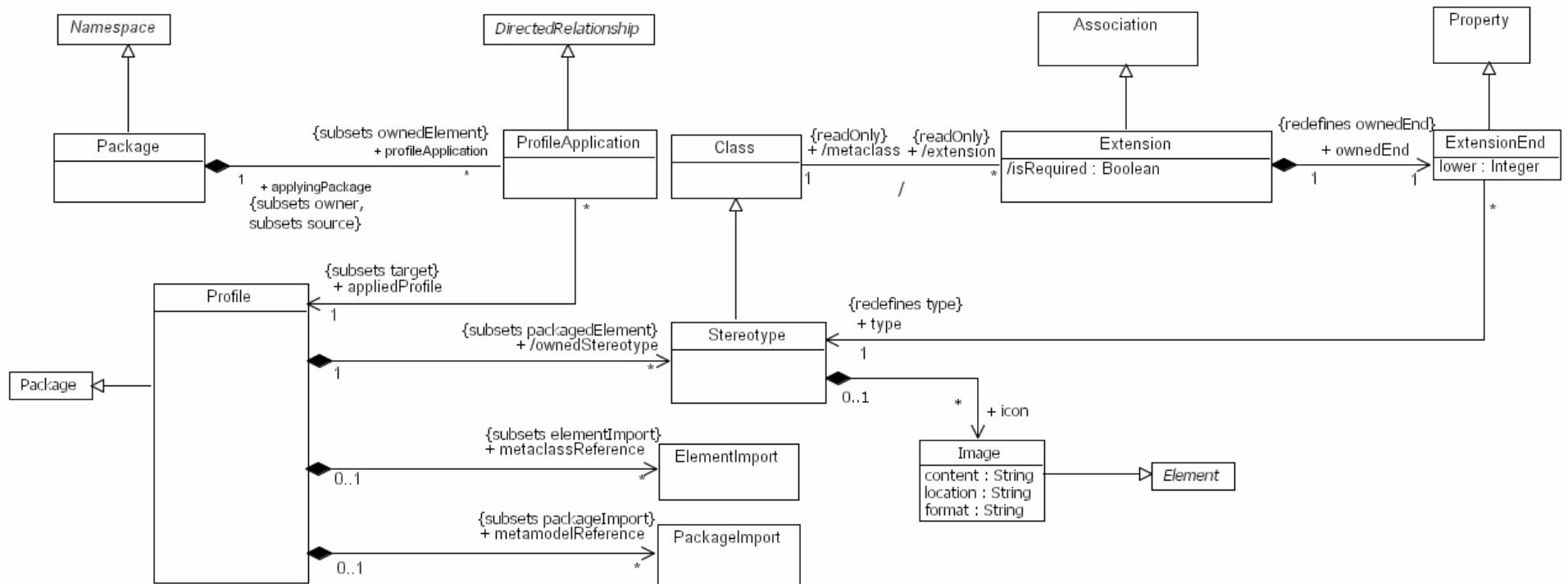
Model Libraries

- **M1 level model fragments packaged for reuse**
 - Identified by the «modelLibrary» standard stereotype
- **Can be incorporated into a profile**
 - Makes them formally part of the profile definition
 - E.g., define an M1 “Semaphore” class in a library package and include the package in the profile
 - The same implicit mechanism of attaching semantics used for stereotypes can be applied to elements of the library
 - Overcomes some of the limitations of the stereotype mechanism
 - Can also be used to type stereotype attributes
- **However, it also precludes some of the advantages of the profiling mechanism**
 - E.g., the ability to view a model element from different viewpoints
- **Model libraries should be used to define useful types shared by two or more profiles or profile fragments as well as by models at the M1 level**

Example: Model Library



The UML Profile Metamodel



Overview

- **A little bit of MOF**
- **Domain-specific modeling languages (DSML)**
- **The UML profile mechanism**
- **UML profile designer's guide**

Guidelines for Defining Profiles

- **Always define a pure domain model (using MOF) first and the profile elements second**
 - Allows separation of two different concerns:
 - What are the right concepts and how are they related?
 - How do the domain-specific concepts map to corresponding UML concepts?
 - Mixing these two concerns often leads to inadequate profiles
- **For each domain concept, find the UML concept(s) that most closely match and define the appropriate stereotype**
 - If no matching UML concept can be found, a UML profile is probably unsuitable for that DSML
 - Fortunately, many of the UML concepts are quite general (object, association) and can easily be mapped to domain-specific concepts

Matching Stereotypes to Metaclasses

- **A suitable base metaclass implies the following:**
 - Semantic proximity
 - The domain concept should be a special case of the UML concept
 - No conflicting well-formedness rules (OCL constraints)
 - Presence of required characteristics and (meta)attributes
 - e.g., multiplicity for domain concepts that represent collections
 - New attributes can always be added but should not conflict with existing ones
 - No inappropriate or conflicting characteristics or (meta)attributes
 - Attributes that are semantically unrelated to the domain concept
 - These can sometimes be eliminated by suitable constraints (e.g., forcing multiplicity to always have a value of 1 or 0)
 - Presence of appropriate meta-associations
 - It is possible to define new meta-associations
 - No inappropriate or conflicting meta-associations
 - These too can be eliminated sometimes by constraints

Beware of Syntactic Matches!

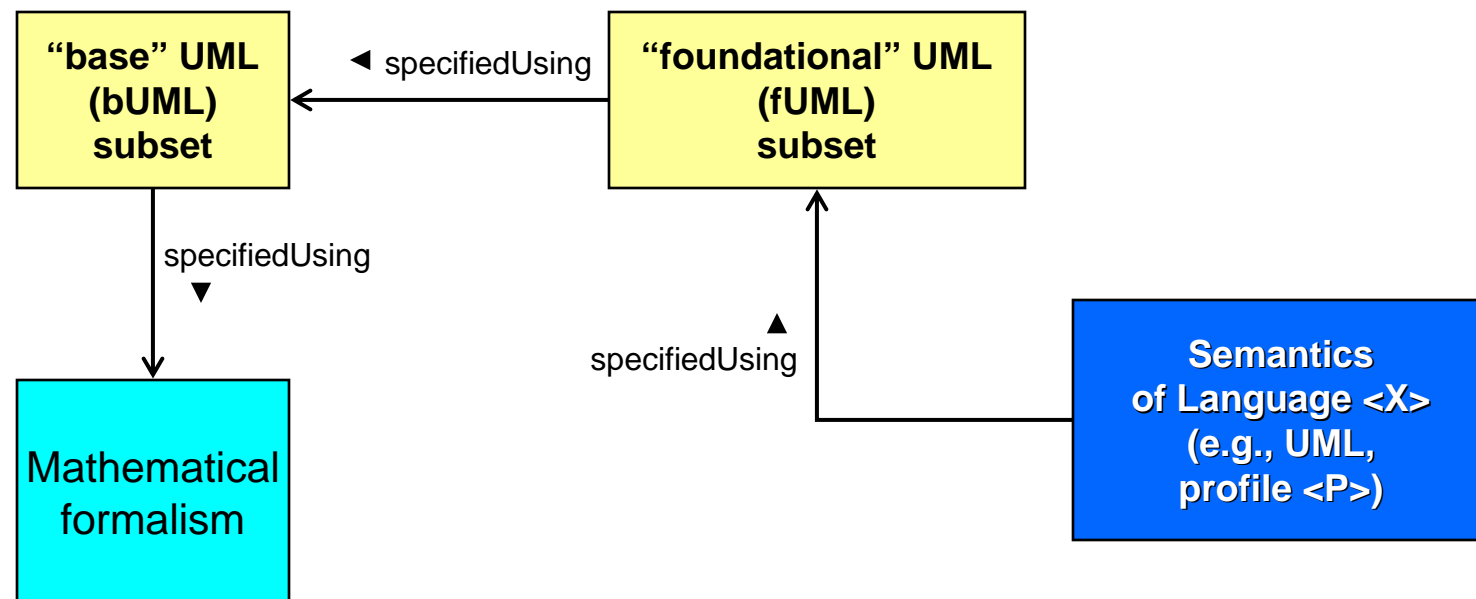
- **Avoid seductive appeal of a syntactic match**
 - In particular, do **not** use things that model M1 entities to capture M0 elements and vice versa
 - Example: using packages to represent groupings of run-time entities
 - Example: using connector and part structures to capture design time dependencies (e.g., requirements dependencies)
- **This may confuse both tools and users**

On Specifying Semantics

- **Semantic compatibility is at the core of the refinement approach to DSMLs**
- **However, currently no standard way to define run-time semantics of a modeling concept**
 - Typically informal natural language description
 - Difficult to validate true semantic compatibility with UML
- **The “Executable UML Foundation” specification is intended to address that by providing a standard way of defining semantics**
 - Will eventually require an addendum to the UML standard which defines the run-time semantics of all standard UML concepts that have a run-time manifestation

Executable UML Foundation Standard

- **Use a small subset of UML as a means to provide a formal definition of run-time semantics**
 - Currently under development (ETA 2007)
- **Two-level approach (operational style):**



MOF or Profile?

- **Depends on the problem at hand**
 - Is there significant semantic similarity between the UML metamodel and the DSML metamodel?
 - Does every domain concept represent a semantic specialization of some UML concept?
 - No semantic or syntactic conflicts?
 - Is language design expertise available?
 - Is domain expertise available?
 - Is support for the DSML available?
 - Tools, training, expertise, literature, etc.
 - Will it be necessary to integrate models with models based on other DSMLs?
- **Example: Specification and Description Language (SDL) (ITU-T standard Z.100)**
 - DSML for defining telecommunications systems and standards
 - First defined in 1970
 - Currently being redefined as a UML profile

Catalog of Adopted OMG Profiles

- **UML Profile for CORBA**
- **UML Profile for CORBA Component Model (CCM)**
- **UML Profile for Enterprise Application Integration (EAI)**
- **UML Profile for Enterprise Distributed Object Computing (EDOC)**
- **UML Profile for Modeling QoS and Fault Tolerance Characteristics and Mechanisms**
- **UML Profile for Schedulability, Performance, and Time**
- **UML Profile for System on a Chip (SoC)**
- **UML Profile for Systems Engineering (SysML)**
- **UML Testing Profile**

Summary (1)

- **The trend towards DSMLs is likely to accelerate**
 - Some concerns with too many DSML and fragmentation
- **Three basic approaches**
 - “From scratch”
 - Opportunity for optimal constructs
 - ...but, may lead to infrastructure problems
 - Extend an existing modeling language
 - Reuse of proven concepts
 - ...but, may lead to infrastructure problems
 - Refine an existing modeling language
 - Reuse of proven concepts and infrastructure
 - ...but may lead to suboptimal language

Summary (2)

- **The MOF/UML profile mechanism is based on the refinement approach**
- **Profiles can be used for two different purposes:**
 - To define DSMLs based on the UML metamodel
 - To define viewpoints for selective viewing of UML models
- **The capabilities of the profile mechanism has been refined significantly in UML 2.1**
 - Ability to subset the metamodel
 - Ability to add meta-associations
 - Strict vs non-strict application of profiles
- **However, some key issues remain**
 - Definition of semantic compatibility and methods for specifying profile semantics

Bibliography

- **IEEE Standard 1471-2000: *Architectural Description of Software-Intensive Systems***
 - http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html
- **OMG's Executable UML Foundation RFP**
 - <http://www.omg.org/docs/ad/05-04-02.pdf>
- **UML 2 Semantics project**
 - <http://www.cs.queensu.ca/~stl/internal/uml2/index.html>
- **ITU-T SDL language standard (Z.100)**
 - http://www.itu.int/ITU-T/studygroups/com10/languages/Z.100_1199.pdf
- **ITU-T UML Profile for SDL (Z.109)**
 - <http://www.itu.int/md/T05-SG17-060419-TD-WP3-3171/en>
- **OMG UML Profiles specifications**
 - http://www.omg.org/technology/documents/profile_catalog.htm

Exercise: Define a profile for the CSP-like language

