Rational. software

Designing Domain-Specific Modeling Languages Using UML Profiles

Bran Selic
IBM Distinguished Engineer, IBM Canada
bselic@ca.ibm.com

IBM Rational Software Development Conference UK 2007

What keeps me **Rational**?

AM4

IBM

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

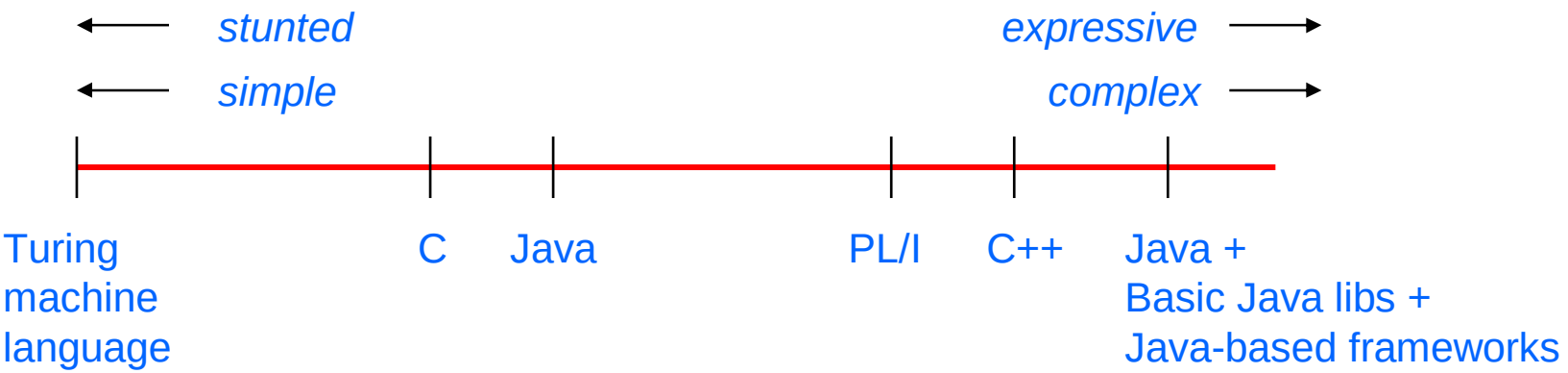
- Literally thousands of domain-specific programming languages have been defined over the past 50 years
  - Fortran: for scientific applications
  - COBOL for “data processing” applications
  - Lisp for AI applications
  - etc.
- Some trends
  - Many of the original languages are still around
  - More often than not, highly-specialized domains still tend to use general-purpose languages with specialized domain-specific program libraries and frameworks 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 *effective* tools (editors, compilers, debuggers, static and dynamic analyzers, build tools, version control tools, merge/diff tools, etc.)
  - Availability of program libraries
  - Availability of skilled practitioners
  - Availability of textbooks and training courses
  - Institutions for evolution and maintenance

On Simplicity in Language Design

- Key design question: *How complex (simple) should a language be to make it effective?*



- The art of computer language design lies in finding the right balance between expressive power and simplicity
  - Need to minimize accidental complexity while recognizing and respecting essential complexity

Some Important Conclusions

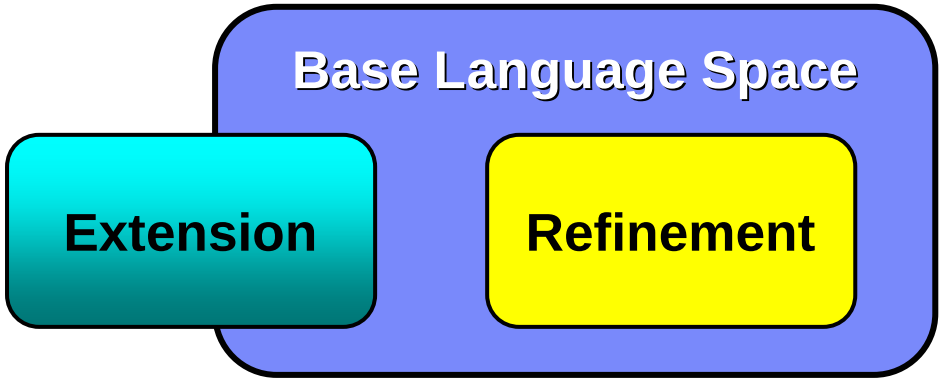
- Designing a useful computer language (modeling or programming) is hard
  - In addition to domain expertise, it requires language design expertise
  - There is no established comprehensive theory of modeling language design to guide the designer
- If the support infrastructure is inadequate, the language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

Domain-Specific Modeling Languages (DSML)

- Computer 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



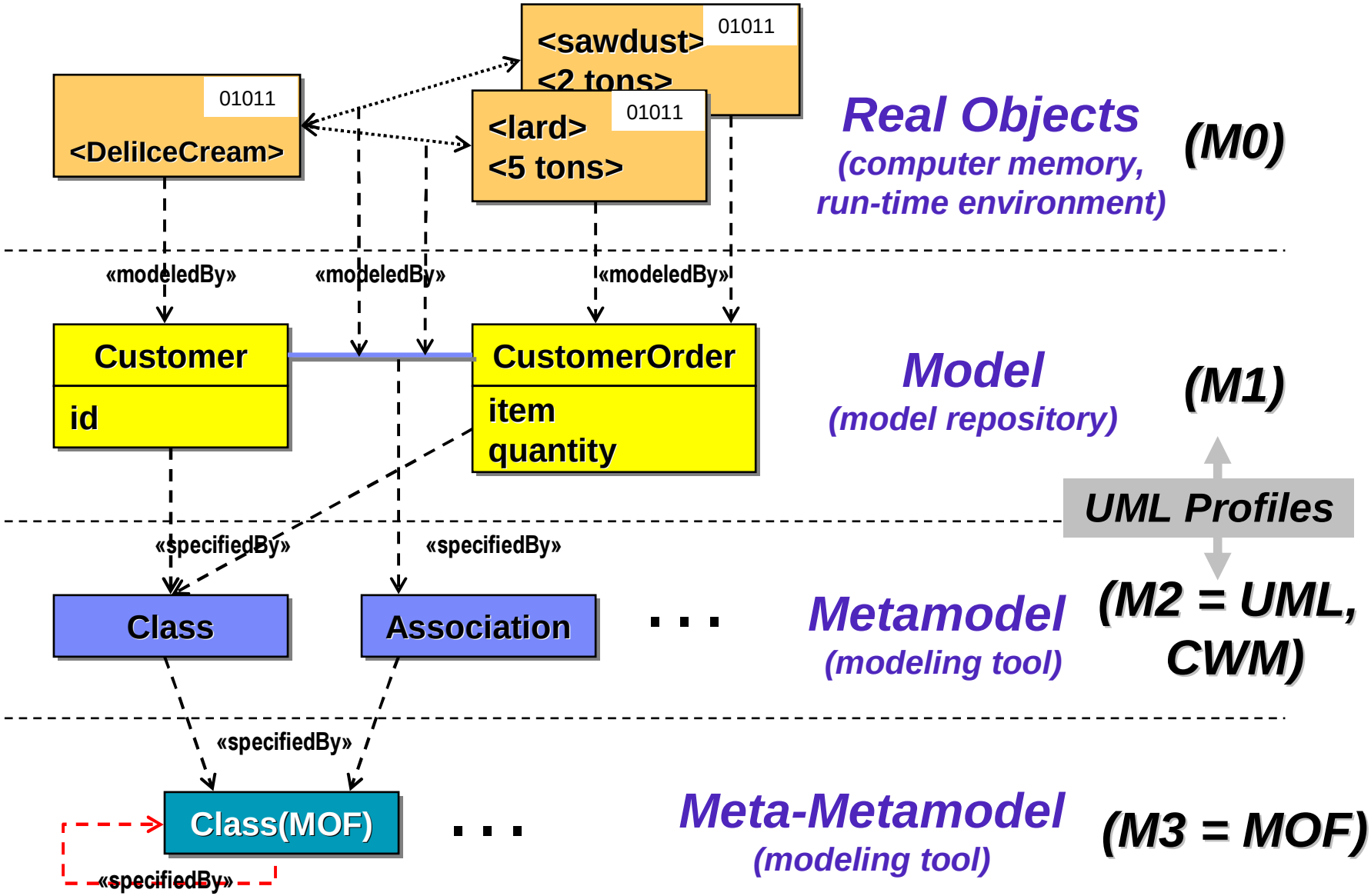
Comparing the Approaches

- New language
  - Pro: potential for maximum expressive power
  - Con: Requires language design skills
  - Con: No language support infrastructure
- 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

OMG's 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)

Defining Modeling Languages: The OMG 4-Layer Architecture



Customizing UML

- Heavyweight/extension
  - Requires adding new concepts (classes) and 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:

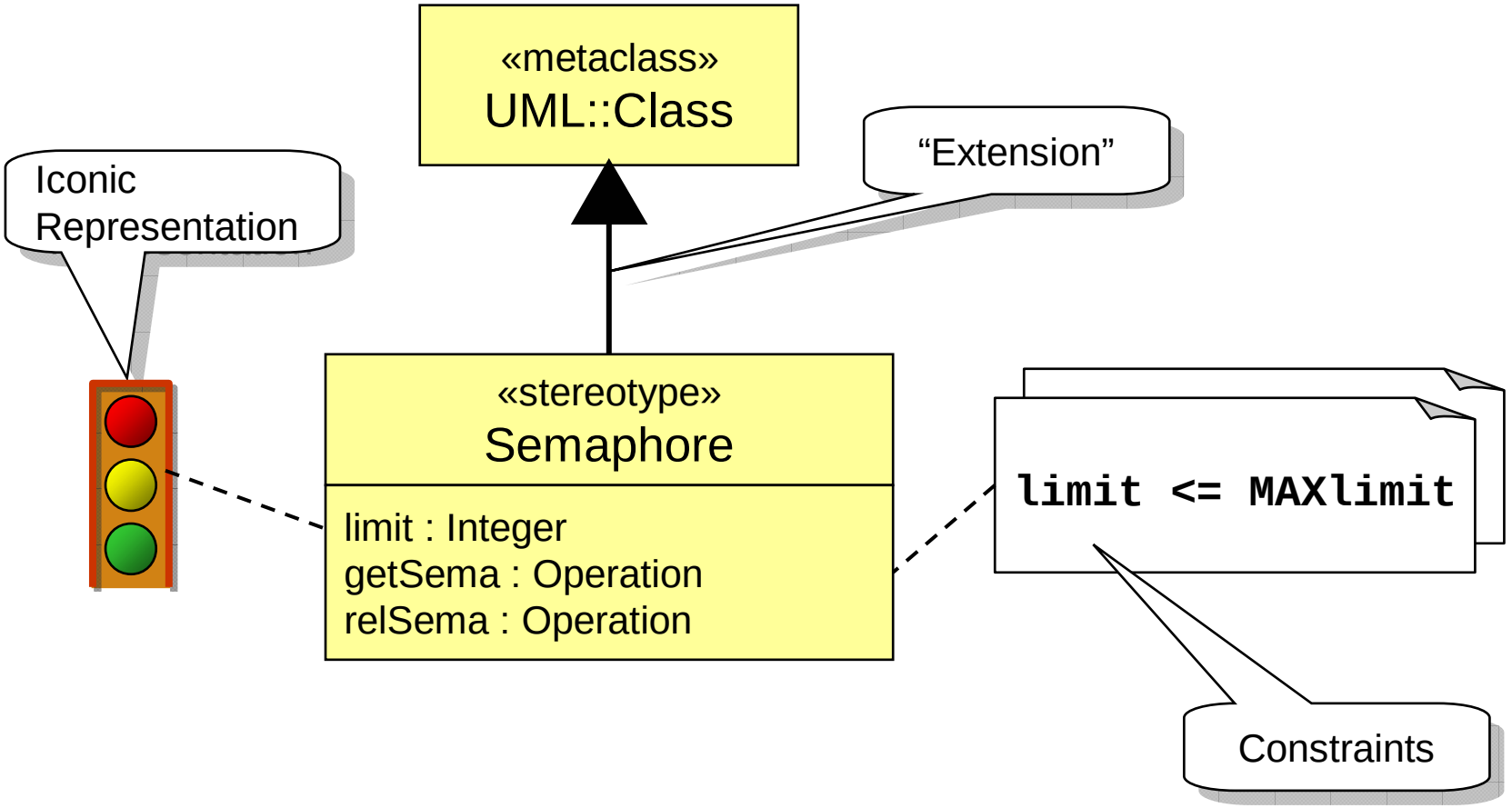
  A specialized object that limits the number of concurrent accesses in a multithreaded environment. When that limit is reached, subsequent accesses are suspended until one of the accessing threads releases the semaphore, at which point the earliest suspended access is given access.

- What is required is a special kind of object
  - Has all the general characteristics of UML objects
  - But includes additional refinements

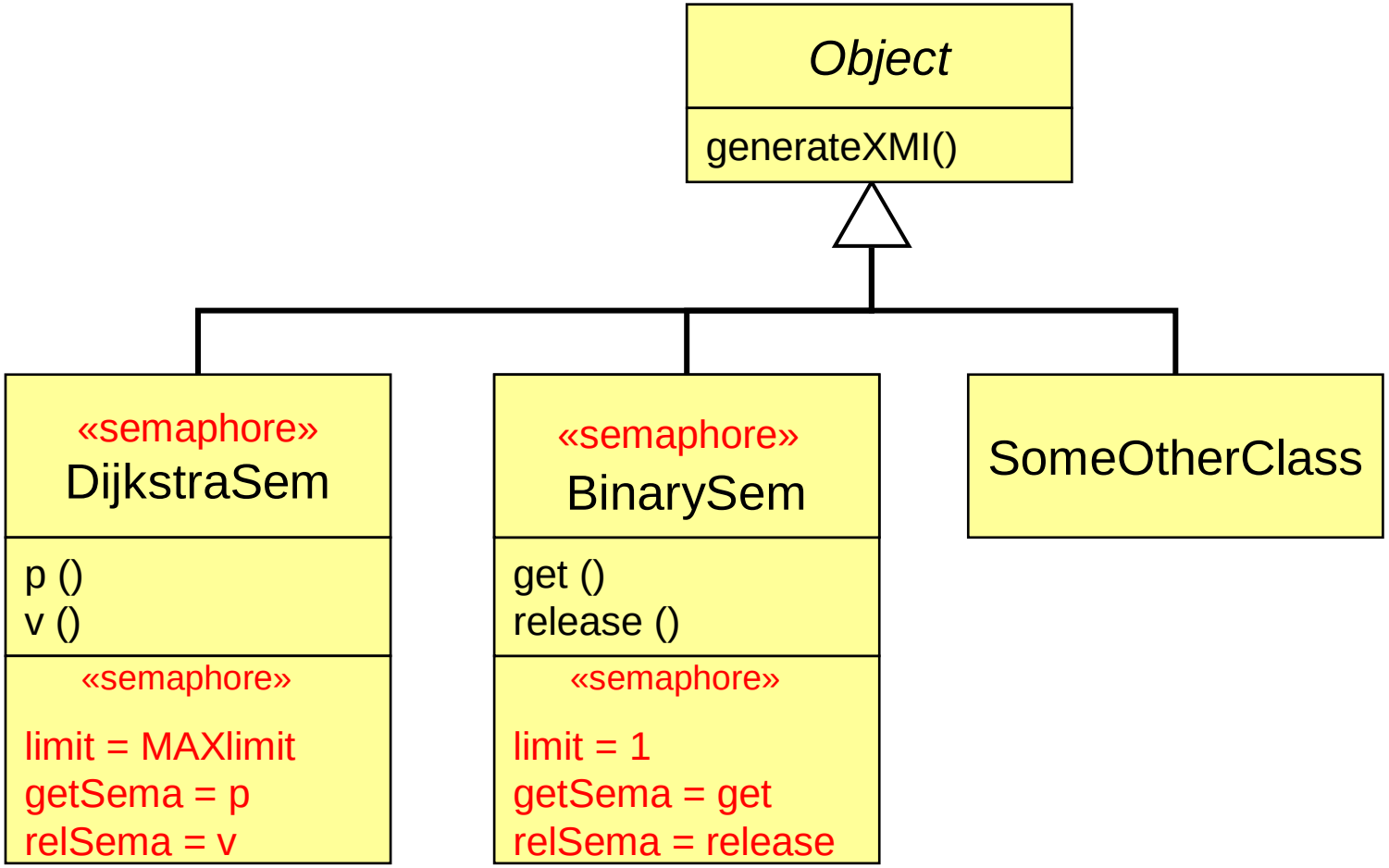
Example: The Semaphore Stereotype

- Refine the UML Class concept by
  - "Associating" 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 using tags (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

Example: Graphical Definition of the Stereotype



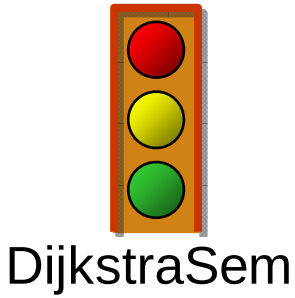
Example: Using the Stereotype



Example: Stereotype Representation Options

(a)

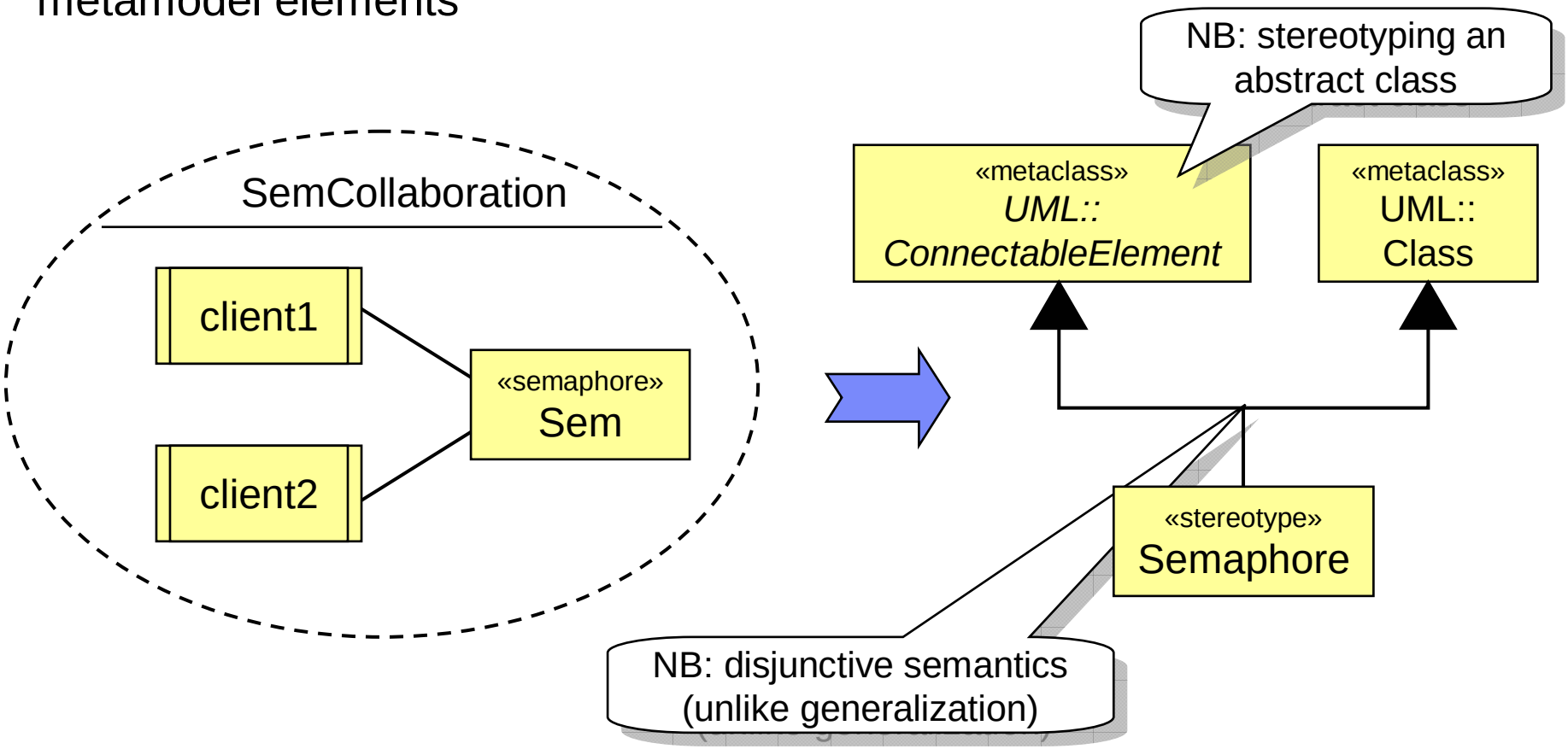
(b)



(c)

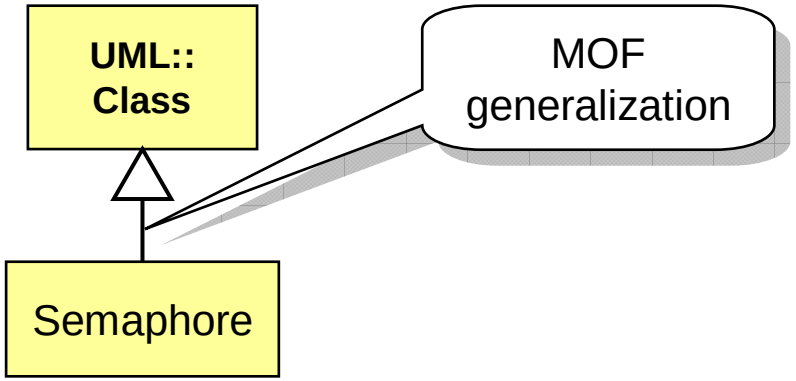
Example: Extending the Scope of a Stereotype

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



Why are Stereotypes Needed?

- Why not simply create a new metaclass?

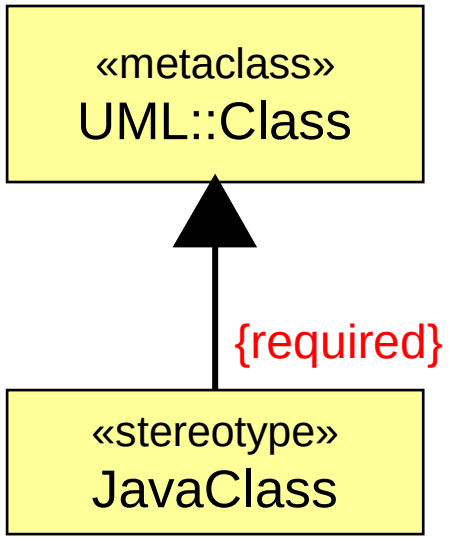


Rationale:

1. Not all modeling tools support meta-modeling ⇒ 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)

“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

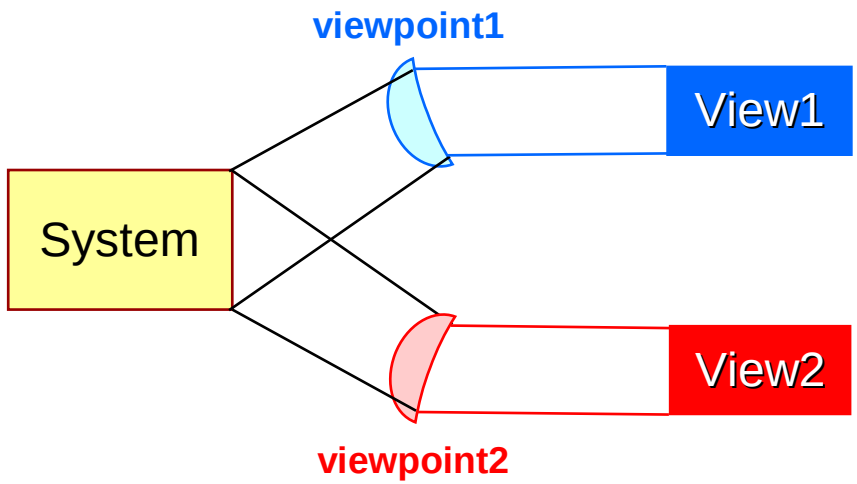


UML Profiles

- 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:
  - To define a domain-specific modeling language
  - To define a domain-specific viewpoint

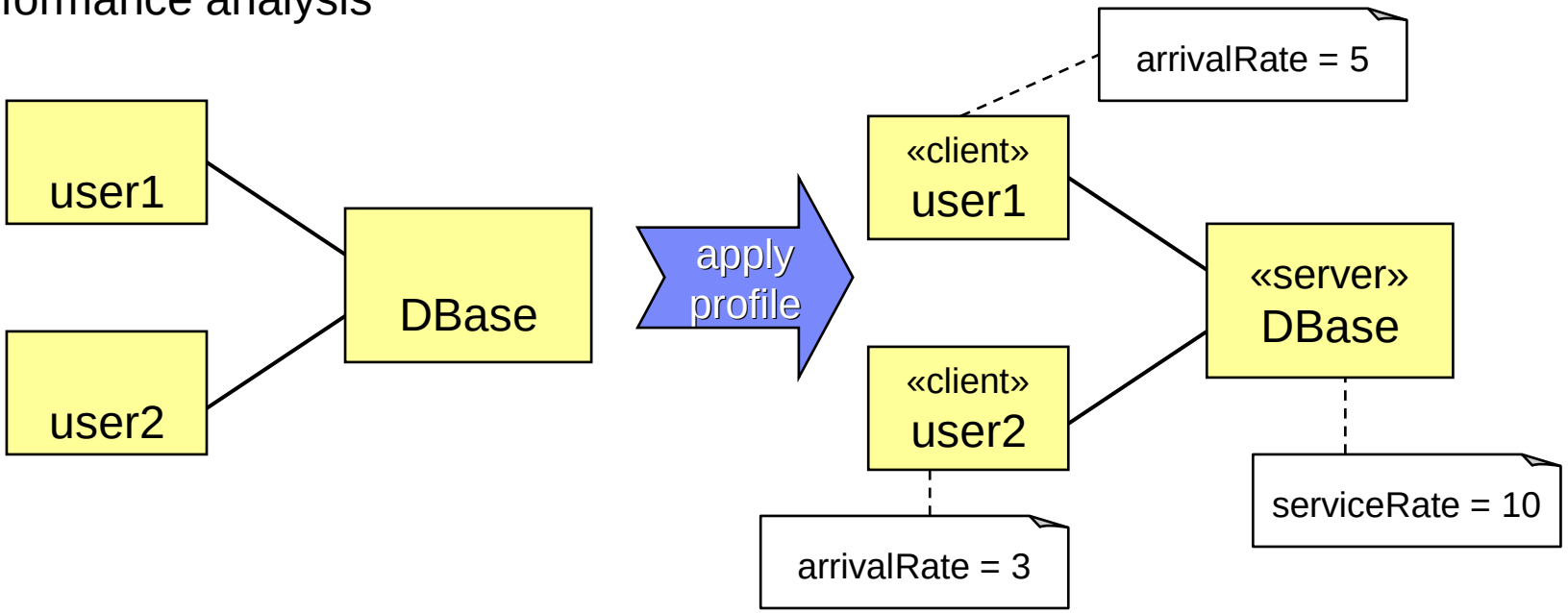
Views and Viewpoints (IEEE 1471)

- *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 a Viewpoint Mechanism

- A profile can be used as an overlay mechanism that can be dynamically applied or "unapplied" to provide a desired view of a 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 UML model
- Example: viewing a UML model fragment as a queueing network to do performance analysis



Strict Profile Application

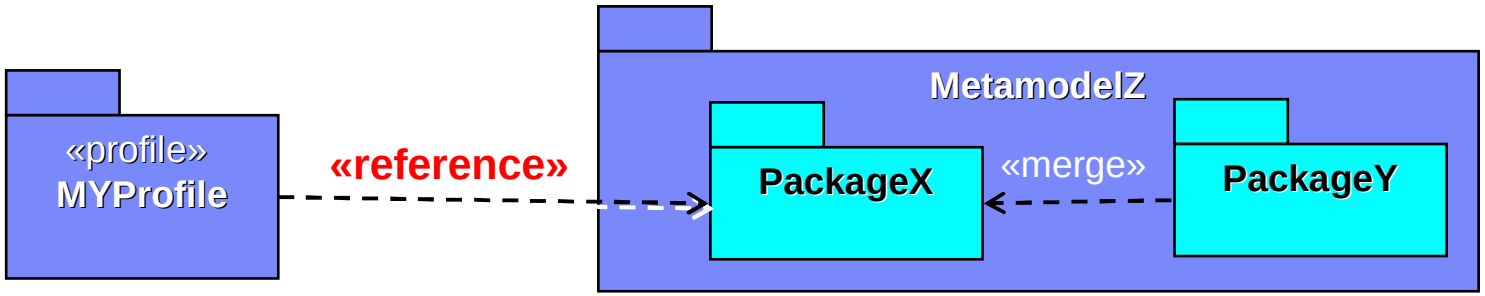
- A *strict* application of a profile will hide from view all model elements that do not have a corresponding stereotype in that profile
  - Convenient for generating views
  - NB: *This does not change the underlying model in any way*
- Strictness is a characteristic of the profile application and not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

Metamodel Subsetting with Profiles (1)

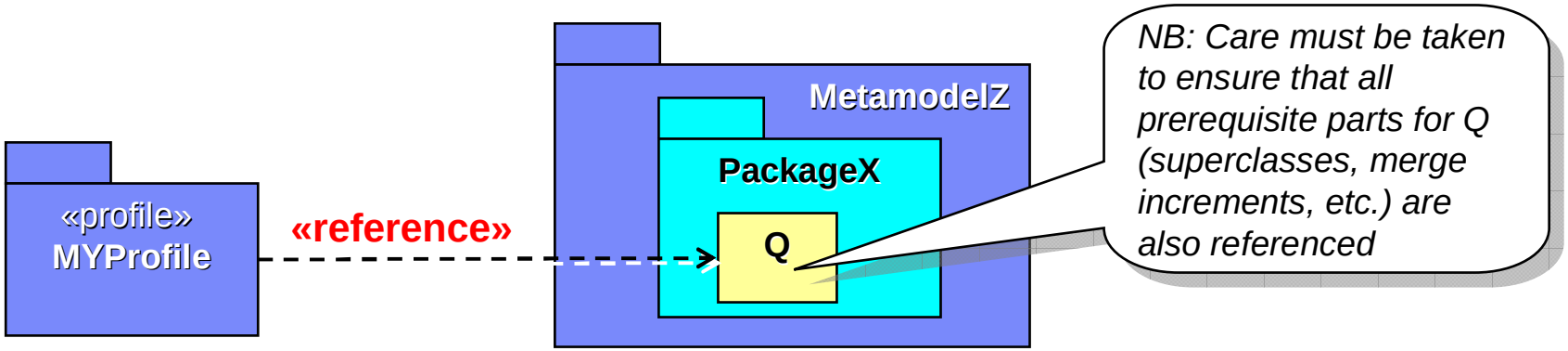
- It is often useful to remove segments of the full UML metamodel resulting in a minimal DSML
  - NB: This is a different mechanism from “strict” profile application – the hiding is part of the profile definition and cannot be applied selectively
- The UML 2.1 profile mechanism adds controls that define 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

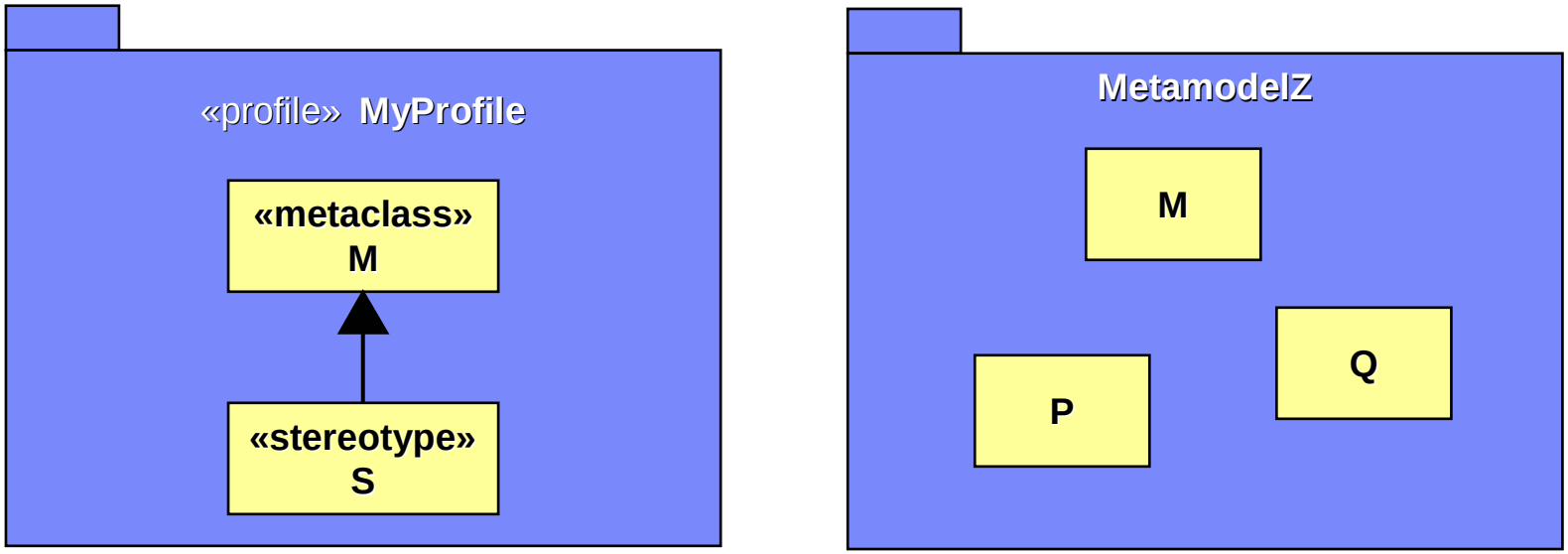


- Case 2: Explicit Metaclass Reference
  - Metaclass Q is visible and can serve as a base metaclass for stereotypes in MyProfile



Metamodel Subsetting with Profiles (3)

- Implicit metaclass reference
  - Metaclass M is visible



Recommended Approach to 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 chosen 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 confliciting meta-associations
    - These too can be eliminated sometimes by constraints

Beware of Syntactic Matches!

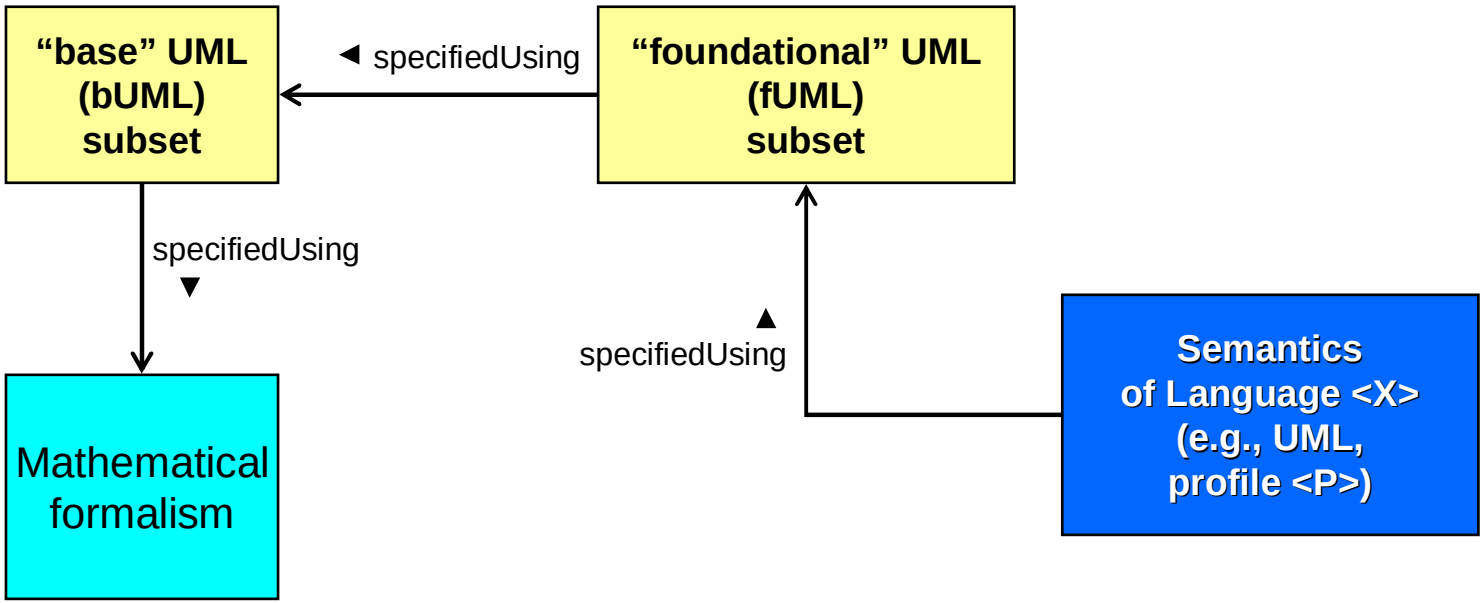
- Avoid seductive appeal of a syntactic match
  - Example: using packages to represent groupings of run-time entities
  - Example: using connector and part structures to capture design time dependencies (e.g., requirements dependencies)
- This may confuse both tools and users

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

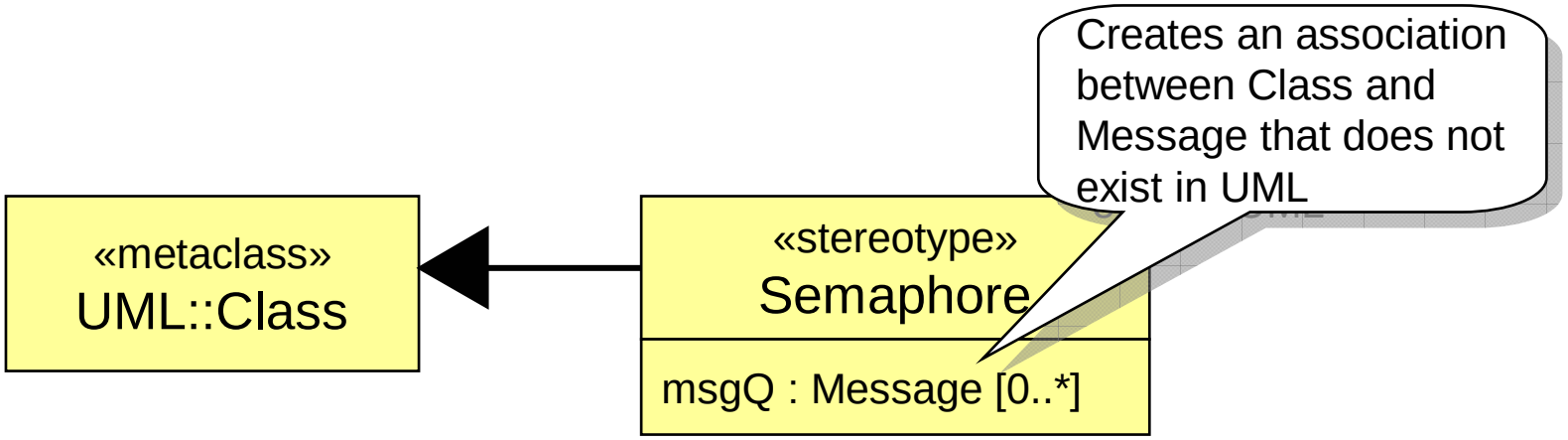
Executable UML Foundation Standard (under development)

- 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):



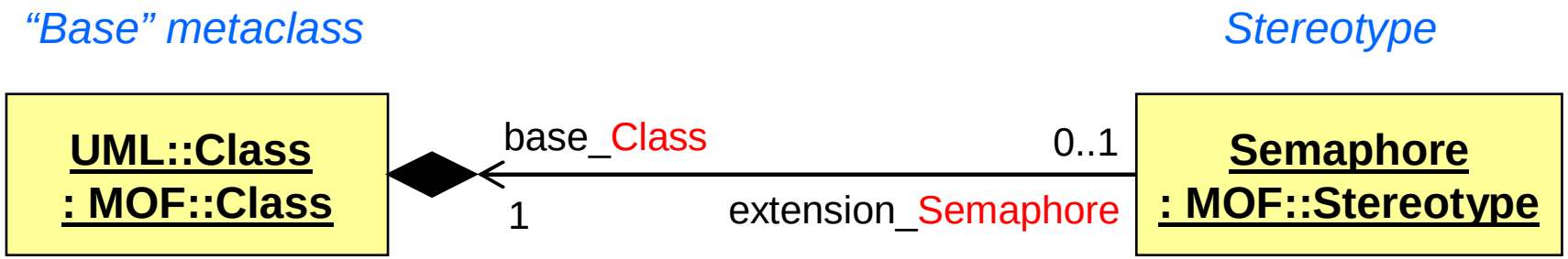
Adding New Meta-Associations

- 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!



The MOF Semantics of UML Extension

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

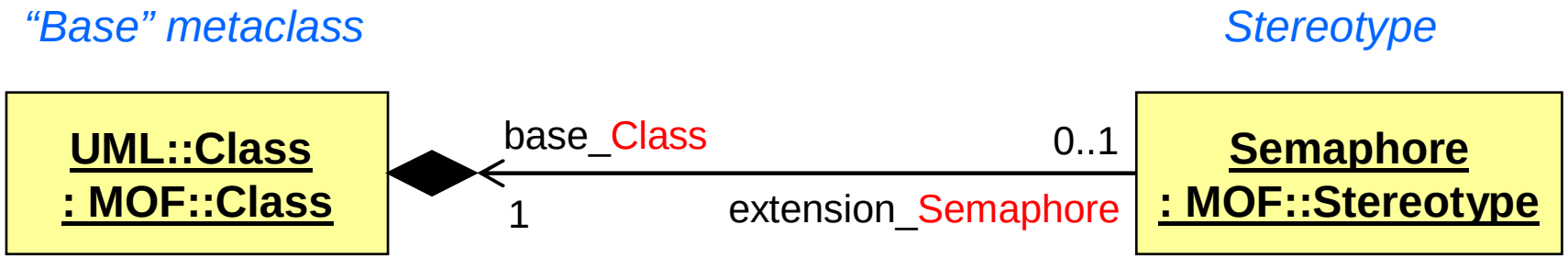


- Association ends naming convention:

  `base_<base-class-name>`
  `extension_<stereotype-name>`

- Required for writing correct OCL constraints for stereotypes

Example: OCL Constraint for a Stereotype



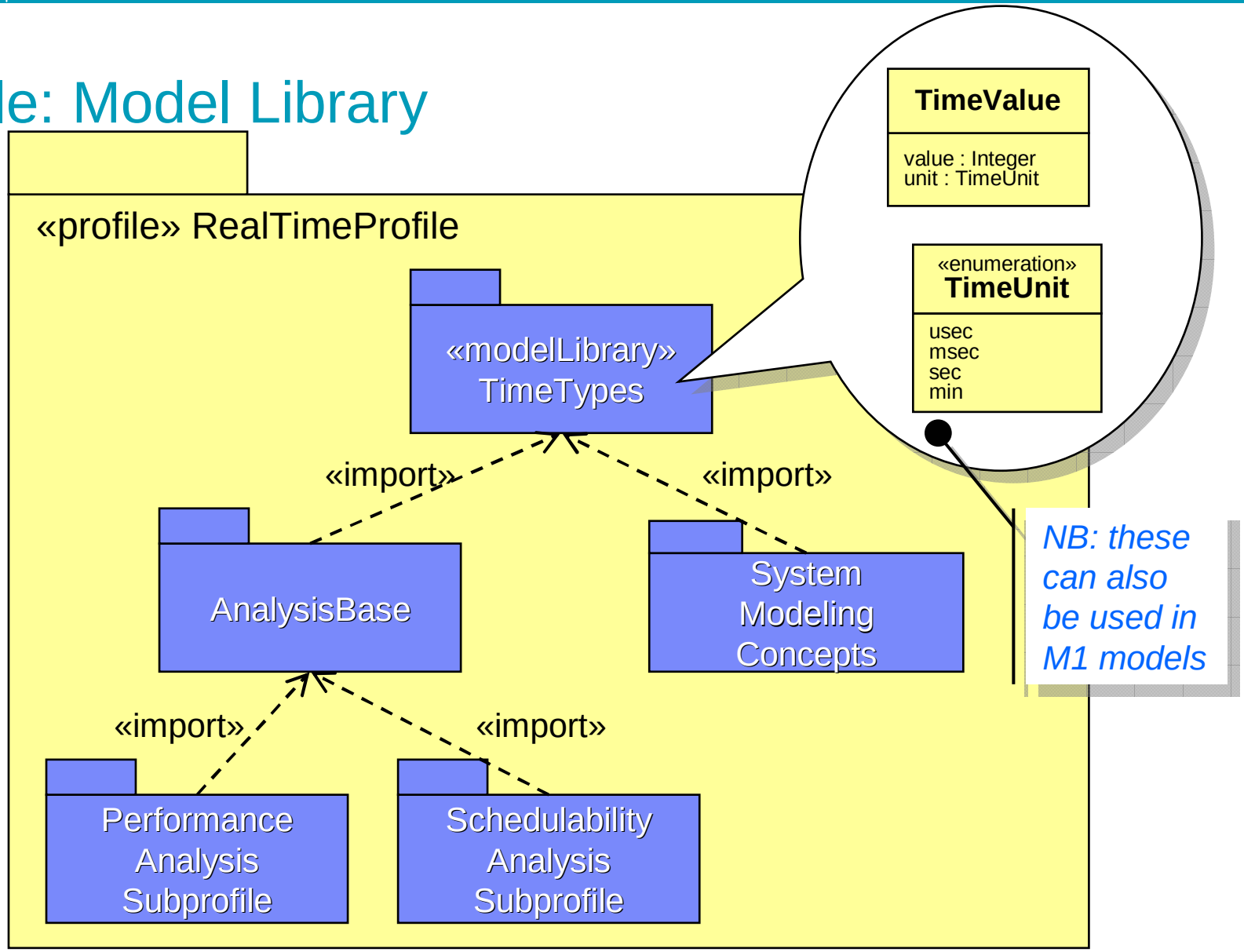
- Semaphore constraint:
  the base Class must have an owned ordered attribute called "msgQ" of type Message

```
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))
```

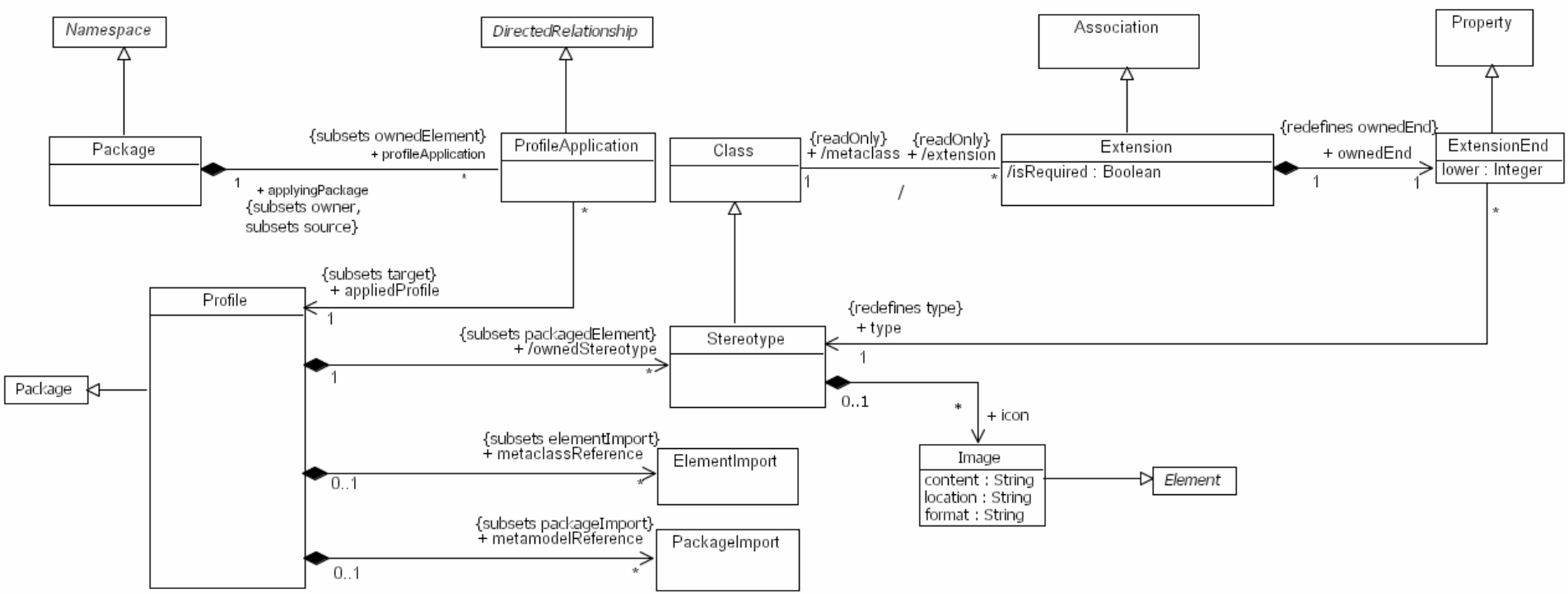
Model Libraries

- 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



MOF or Profile for My DSML?

- 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
- UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE)
- UML Profile for DoDAF/MoDAF (UPDM)

Summary (1)

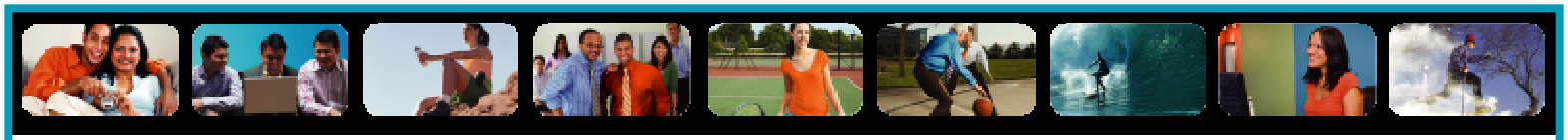
- The trend towards DSMLs is likely to accelerate
  - Some concerns with too many DSMLs and resulting fragmentation
- Three basic approaches
  - “From scratch”
    - Opportunity for optimal constructs
    - …but, may lead to infrastructure problems
  - Extend an existing modeling language
    - Reuse 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)

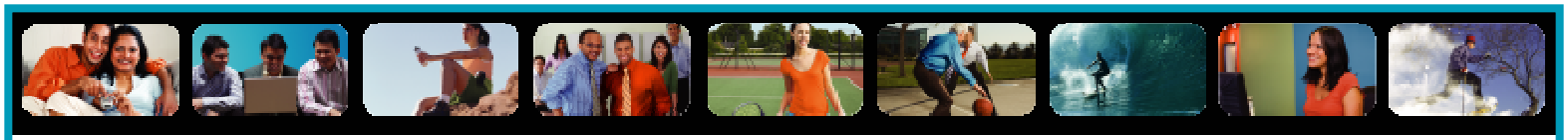
- The MOF/UML profile mechanism is based on the refinement approach
- Profiles can be used for two different purposes:
  - To define DSMLs based on the UML metamodel
  - To define viewpoints for selective viewing of UML models
- The capabilities of the profile mechanism has been refined significantly in UML 2.1
  - Ability to subset the metamodel
  - Ability to add meta-associations
  - Strict vs non-strict application of profiles
- Tools such as Rational's RSx series of modeling tools support the profile mechanism and have been used to define numerous standard and custom profiles

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



Questions



Thank You

Bran Selic
bselic@ca.ibm.com