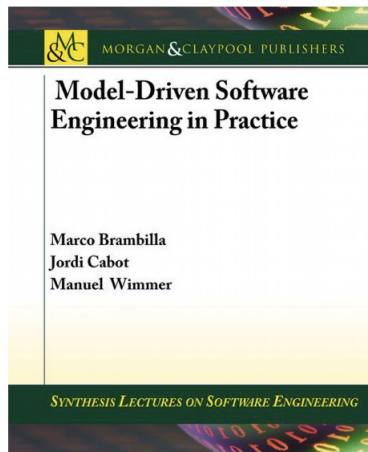


EDOM - Engenharia de Domínio
Mestrado em Engenharia Informática
Lecture 09.2
Model Driven Architecture (MDA)

Alexandre Bragança atb@isep.ipp.pt

Dep. de Engenharia Informática – ISEP

2017/2018



"Model-Driven Software Engineering in Practice", Marco Brambilla et al., Morgan & Claypool Publishers, 2012

- These slides are based on the contents of this book.

- MDA
- UML (from a metamodeling perspective)

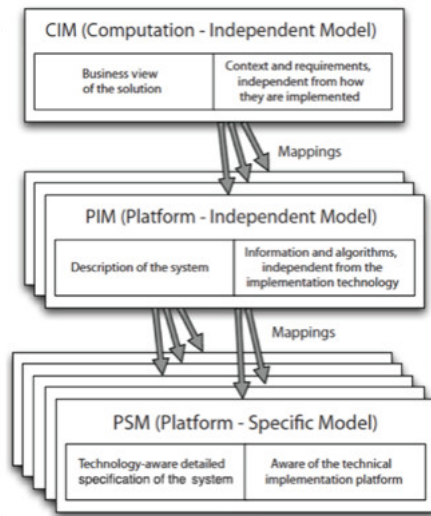
- The Object Management Group (OMG) has defined its own comprehensive proposal for applying MDE practices to system's development:

MDA (Model-Driven Architecture)

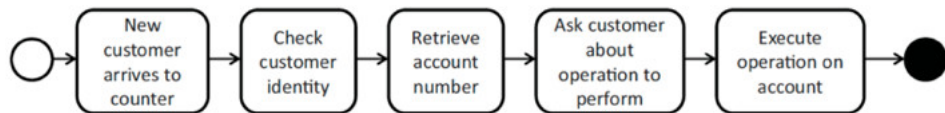
- **Models must be expressed in a well-defined notation**, so as to enable effective communication and understanding
- Systems specifications must be organized around **a set of models and associated transformations**
 - implementing mappings and relations between the models.
 - multi-layered and multi-perspective architectural framework.
- Models must be **compliant with metamodels**
- Increase acceptance, broad adoption and tool competition for MDE

- **System:** The subject of any MDA specification (program, computer system, federation of systems)
- **Problem Space (or Domain):** The context or environment of the system
- **Solution Space:** The spectrum of possible solutions that satisfy the reqs.
- **Model:** Any representation of the system and/or its environment
- **Architecture:** The specification of the parts and connectors of the system and the rules for the interactions of the parts using the connectors
- **Platform:** Set of subsystems and technologies that provide a coherent set of functionalities for a specified goal
- **Viewpoint:** A description of a system that focuses on one or more particular concerns
- **View:** A model of a system seen under a specific viewpoint
- **Transformation:** The conversion of a model into another model

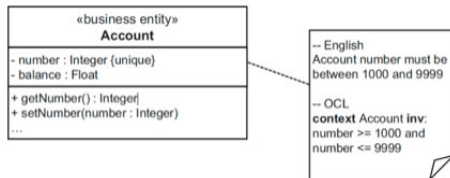
- **Computation independent (CIM):** describe requirements and needs at a very abstract level, without any reference to implementation aspects (e.g., description of user requirements or business objectives);
- **Platform independent (PIM):** define the behavior of the systems in terms of stored data and performed algorithms, without any technical or technological details;
- **Platform-specific (PSM):** define all the technological aspects in detail.



- E.g., business process

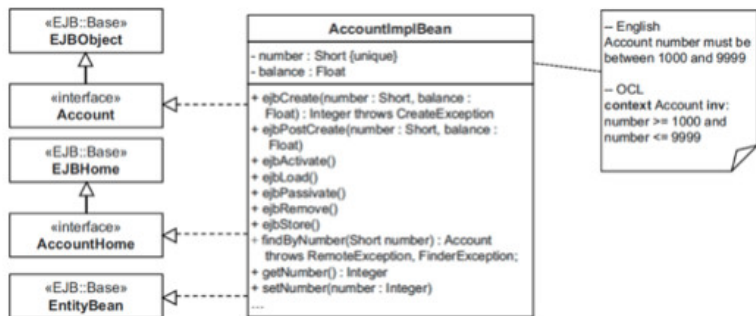


MDA Platform Independent Model (PIM)

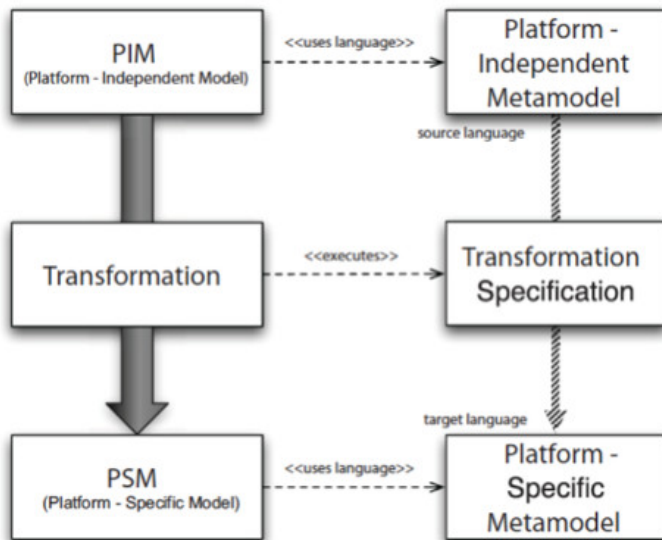


- specification of structure and behaviour of a system, abstracted from technological details
- Using the UML (optional)
- Abstraction of structure and behaviour of a system with the PIM simplifies the following:
 - Validation for correctness of the model
 - Create implementations on different platforms
 - Tool support during implementation

MDA Platform Specific Model (PSM)



- Specifies how the functionality described in the PIM is realized on a certain platform
- Using a UML-Profile for the selected platform, e.g., EJB

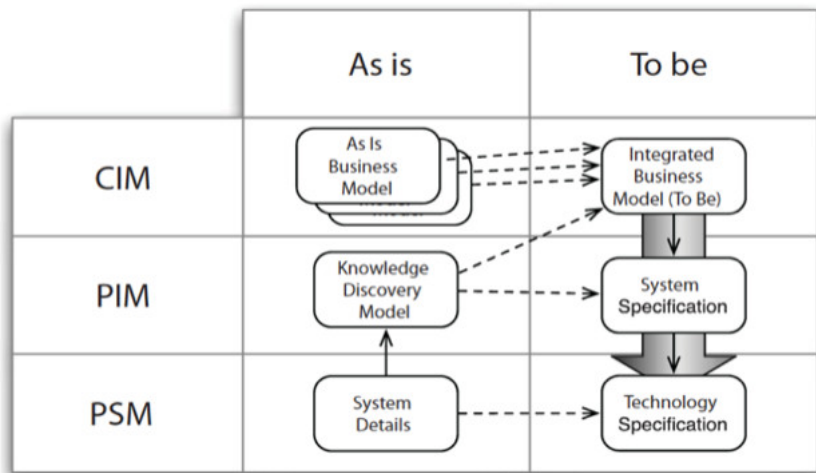


- MDA's core is UML, a standard general-purpose software modeling language
- Two options for specifying your languages:
 - (Domain-specific) UML Extensions can be defined through UML Profiles
 - Full-fledged Domain-specific languages (DSMLs) can be defined by MOF

ADM (Architecture-Driven Modernization) is addressing the problem of system reverse engineering

It includes several standards that help on this matter

- **The Knowledge Discovery Metamodel (KDM):** An intermediate representation for existing software systems that defines common metadata required for deep semantic integration of lifecycle management tools. Based on MOF and XMI
- **The Software Measurement Metamodel (SMM):** A meta-model for representing measurement information related to software, its operation, and its design.
- **The Abstract Syntax Tree Metamodel (ASTM):** A complementary modeling specification with respect to KDM, ASTM supports a direct mapping of all code-level software language statements into low-level software models.

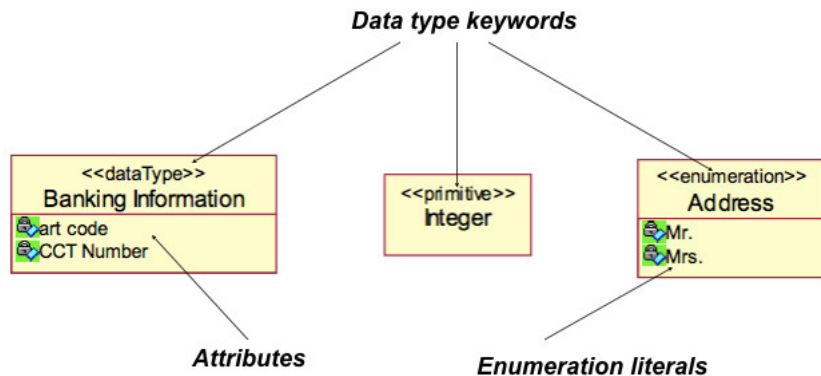


¹MDRE = Model Driven Reverse Engineering

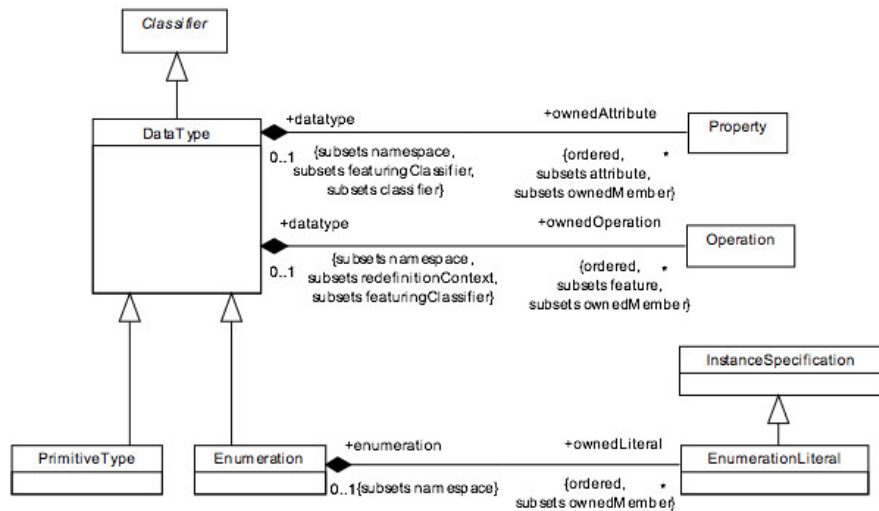


UML distinguishes between the following data types:

- **Simple data types** (DataType): a type with values that have no identity; that means two instances of a datatype with the same attributes values are indistinguishable.
- **Primitive data types** (PrimitiveType): a simple data type without structures. UML defines the following primitive data types:
 - Integer: (Infinite) set of integers: (...,-1,0,1,...)
 - Boolean: true, false.
 - UnlimitedNatural (Infinite) set of natural numbers (0, 1, 2, 3, ...) plus infinite (*).
 - String: a sequence of characters, containing any character, even from non-Roman alphabets. Values of this type are presenter in double quotes, e.g. "UMLBase".
 - Real: An instance of Real is a value in the (infinite) set of real numbers.
- **Enumeration types** – simple data types with values that originate from a limited set of enumeration literals.



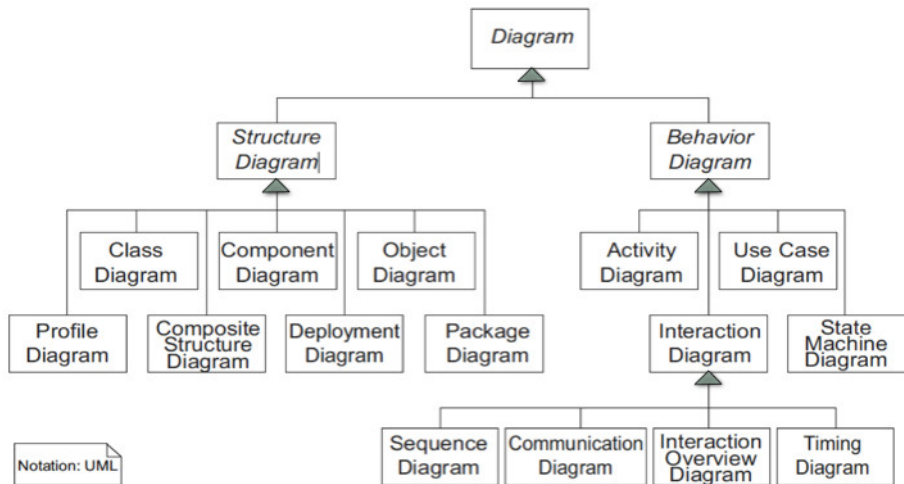
The Metamodel of Data Types

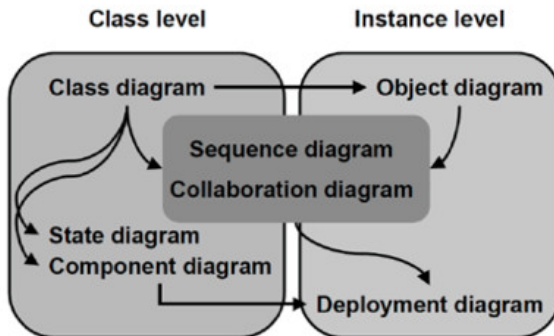


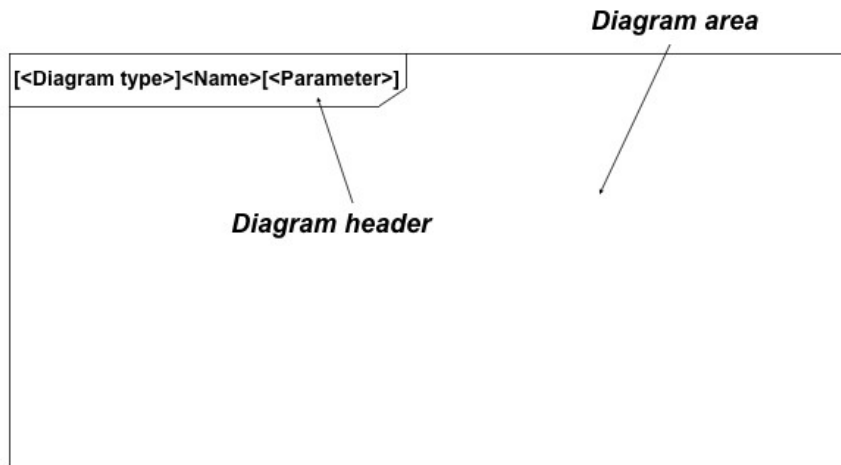
- There is no official UML diagram overview or diagram grouping.
- Although UML models and the repository underlying all diagrams are defined in UML, the definition of diagrams (i.e. special views of the repository) are relatively free.

- In UML a diagram is actually more than a collection of notational elements.
- For example, the package diagram describes the package symbol, the merge relationship, and so on.
- A class diagram describes a class, the association, and so on.
- Nevertheless, we can actually represent classes and packages together in one diagram.

Overview of the UML Diagrams



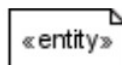
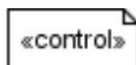
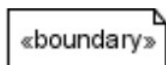




- Stereotypes are formal extensions of existing model elements within the UML metamodel, that is, **metamodel extensions**.
- The modeling element is directly influenced by the **semantics** defined by the extension.
- Rather than introducing a new model element to the metamodel, **stereotypes add semantics to an existing model element**.

- **Several stereotypes** can be used to classify one single modeling element.
- Even **the visual representation** of an element can be influenced by allocating stereotypes.
- Moreover, stereotypes **can be added to** attributes, operations and relationships.
- Further, stereotypes **can have attributes** to store additional information.

- A stereotype is placed before or above the element name and enclosed in guillemets («,»).
- **Important:** not every occurrence of this notation means that you are looking at a stereotype. **Keywords** predefined in UML are also enclosed in guillemets.



Stereotype	UML element	Description
<<call>>	Dependency(usage)	Call dependency between operation or classes
<<create>>	Dependency(usage)	The source element creates instances of the target element
<<instantiate>>	Dependency(usage)	The source element creates instances of the target element Note: This description is identical to the one of <<create>>
<<responsability>>	Dependency(usage)	The source element is responsible for the target element
<<send>>	Dependency (usage)	The source element is an operation and the target element is a signal sent by that operation
<<derive>>	Abstraction	The source element can, for instance, be derived from the target element by a calculation
<<refine>>	Abstraction	A refinement relationship (e.g. Between a desing element and a pertaining analysis element)
<<trace>>	Abstraction	Serves to trace of requirements

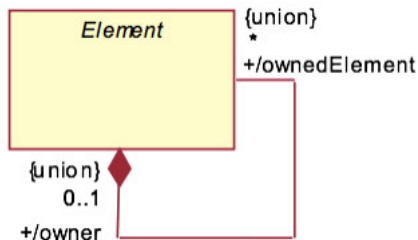
Stereotype	UML element	Description
<<script>>	Artifact	A script file (can be executed on a computer)
<<auxiliary>>	Class	Classes that support other classes (<<focus>>)
<<focus>>	Class	Classes contain the primary logic. See <<auxiliary>>
<<implementationClass>>	Class	An implementation class specially designed for a programming language, where an object may belong to one class only
<<metaclass>>	Class	A class with instances that are, in turn, classes
<<type>>	Class	Types define a set of operations and attributes, and they are generally abstract
<<utility>>	Class	Utility class are collections of global variables and functions, which are grouped into a class, where they are defined as class attributes/ operations
<<buildComponent>>	Component	An organizational motivated component

Stereotype	UML element	Description
<<implement>>	Component	A component that contains only implementation, not specification
<<framework>>	Package	A package that contains Framework elements
<<modelLibrary>>	Package	A package that contains model elements, which are reused in other packages
<<create>>	Behavioral feature	A property that creates instances of the class to which it belongs (e.g. Constructor)
<<destroy>>	Behavioral feature	A property that destroys instances of the class to which it belongs (e.g. Destructor)

Class Diagrams refer to this area of the metamodel:

- Package: `Classes::Kernel`
- Package: `Classes::Dependencies`
- Package: `Classes::Interfaces`

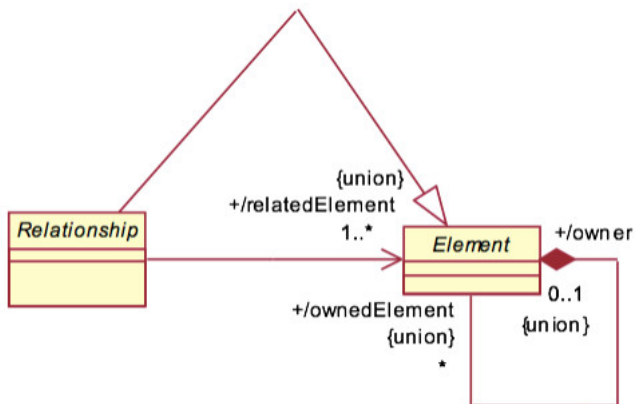
- The basis of UML is described in the **Kernel** package of the metamodel.
- Most class models have the superclass **Element** and has the ability to own other elements, shown by a composition relationship in the metamodel.
- That's the only ability an element has.



There is no notation for an element because you would never use the element construct in UML models. The class is abstract.

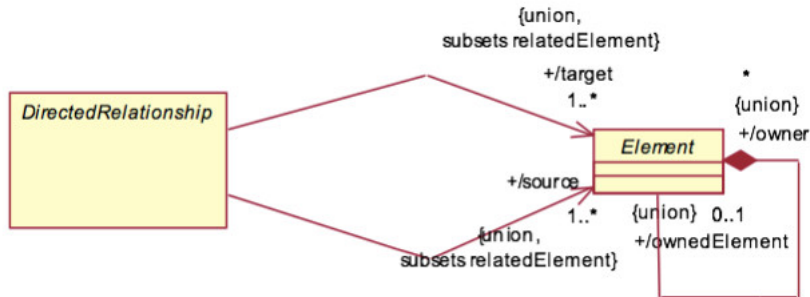
- A *Relationship* is an abstract concept to put elements in relation to one another.
- Similar to *Element*, there is no other property or semantics. The properties and the semantics are added later by abstract or concrete subclasses.
- There is no notation for *Relationship* either.

The Basic Relationship Class

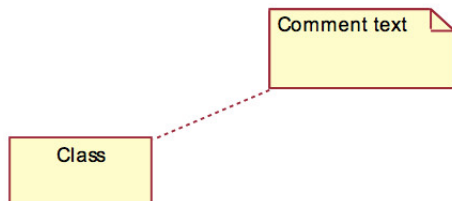


- The Relationship concept is specialized by the concept of a **direct relationship**.
- The set of related elements is divided into a set of source and a set of target elements.
- In many relationships, one element offers something and another element wants something.
- The former is called a **supplier** and the latter is a **client**. This is expressed **in one direction**.

Directed relationships



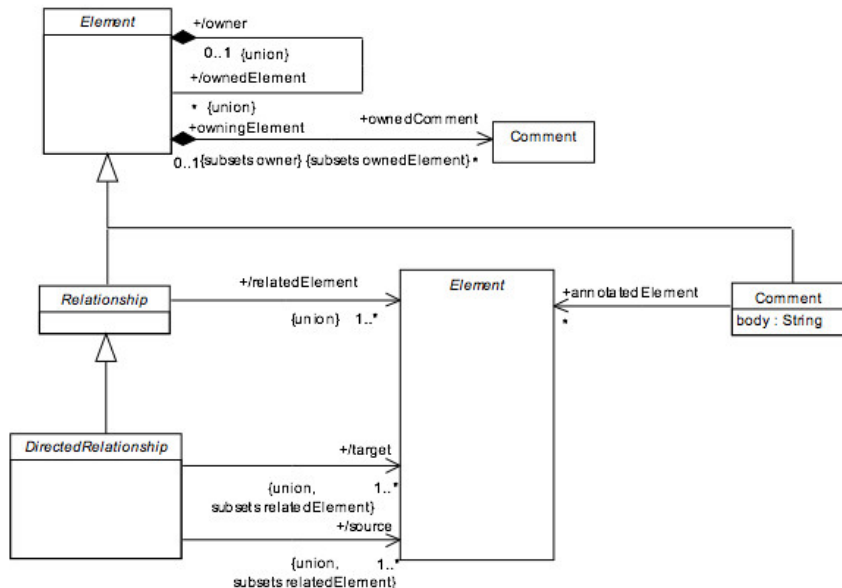
Note that we are dealing only with abstract and rather simple concepts.



The notation for comments.

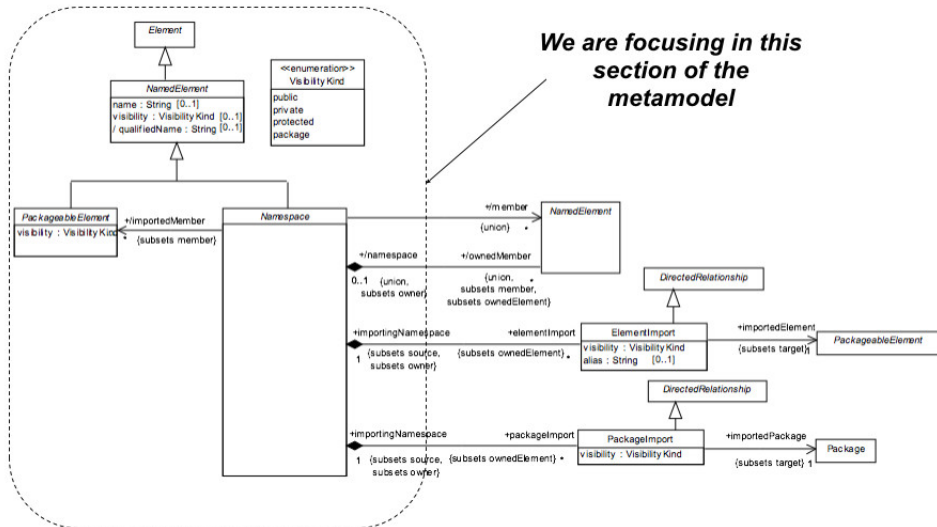
- Comments and notes are terms often used synonymously.
- A comment can be annotated to any UML model element. In the metamodel, you can see that the **Comment** class is directly associated with the *Element* base class.
- **Comment** is a concrete class.

The Basic Metamodel Concepts

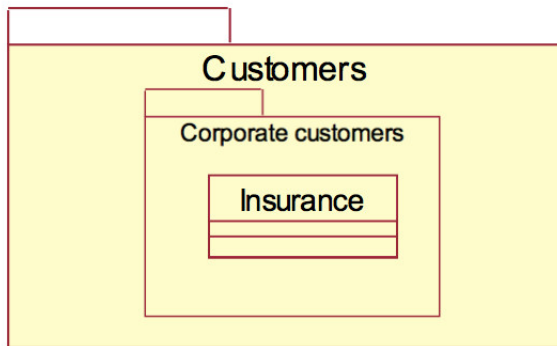


- **Definition** - A named element is an element that can have a name and a defined visibility (public, private, protected, package):
- +=public
- -=private
- #=protected
- ~=package
- The name of the element and its visibility are optional.

The Metamodel for NamedElement



- A **namespace** is a named element that can contain named elements.
- Within a namespace, named elements are uniquely identified by their names.
- In addition, they have a qualified name, resulting from nested namespaces.
- The qualified name of a named element can be derived from the nesting of the enclosing namespaces.



Qualified name

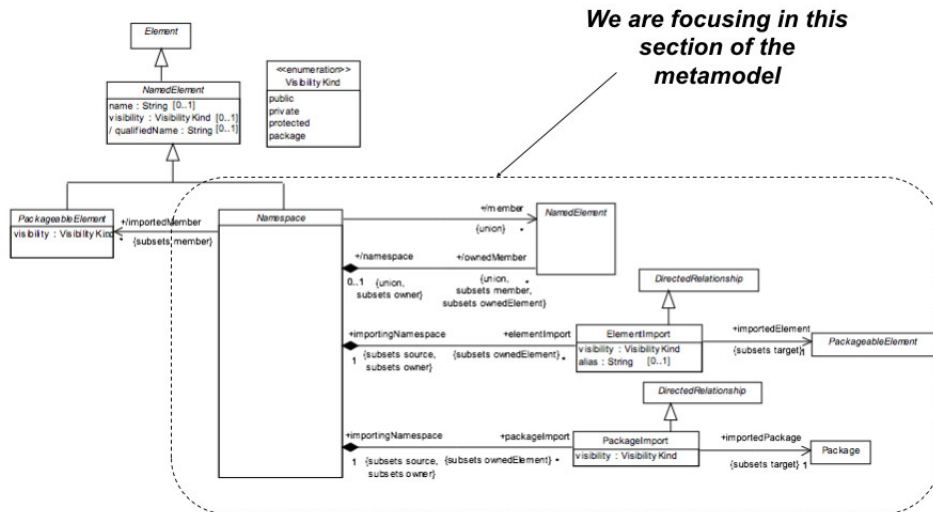
Customers::CorporateCustomers:Insurance

- A **packageable element** is a named element that can belong directly to a package.
- Example: an operation cannot belong to a package, but a class may.
- The visibility statement is mandatory for a packageable element.

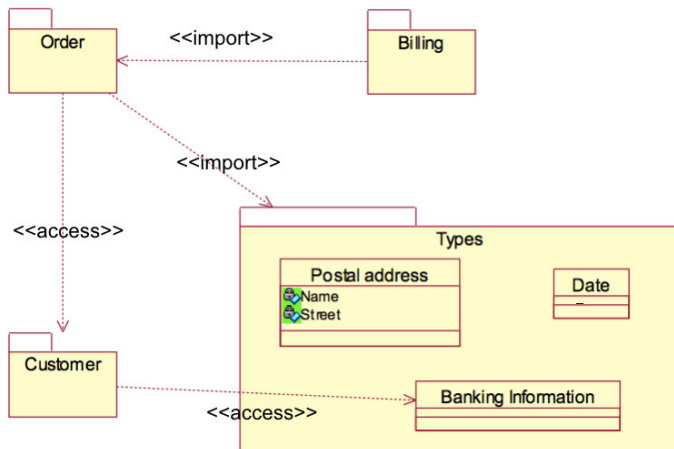
- The act of importing an element is called **ElementImport** and is a relationship between a namespace and a packageable element that resides in another namespace.
- The referenced element can then be addressed directly by its (unqualified) name. In addition, an optional alias name can be specified.

- The act of importing a package is called **PackageImport**; it is semantically equivalent to the import of a single element from that package.
- We cannot specify an alias name here.

The Metamodel for NamedElement

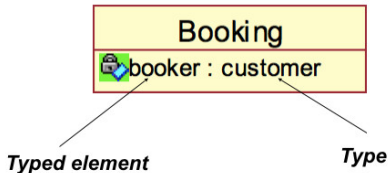


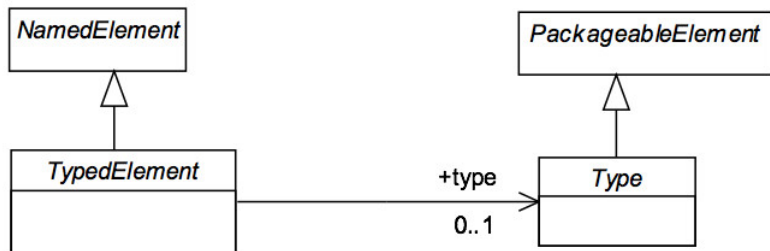
Example of Element and Package Import Relationships



- **«import»**: The visibility is **public**; for example, the postal address for Order. The public import is a transitive relationship: if A imports B and B imports C, then A is indirectly importing C too.
- **«access»**: The visibility is **private**, not public: Customer is visible in Order but not in Billing. The private import is not transitive.

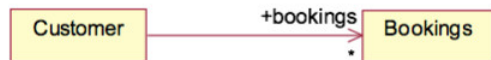
- A **typed element** is a named element that can have a type.
- Ex.- Attributes and parameteres.
- A **type** specifies a set of values for a typed element.
- Ex.- Simple data types and classes are types.





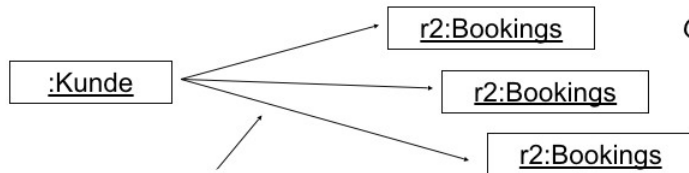
***Type and typed element are abstract classes.
They have no properties***

- A **multiplicity element** is the definition of an interval of positive integers to specify allowable cardinalities.
- A **cardinality** is a concrete number of elements in a set.
- A multiplicity element is often simply called **multiplicity**; the two terms are synonymous.



Class model

*Multiplicity=0..**



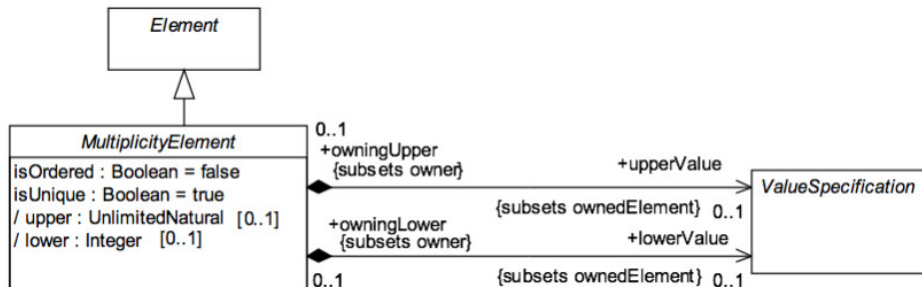
Object model

Cardinality=3

- The **notation** for multiplicity is either a single number or a value range.
- A value range is written by stating the minimum and maximum values, separated by two dots (e.g. 1..5).
- In addition, you can use the wildcard character * to specify an arbitrary number of elements.

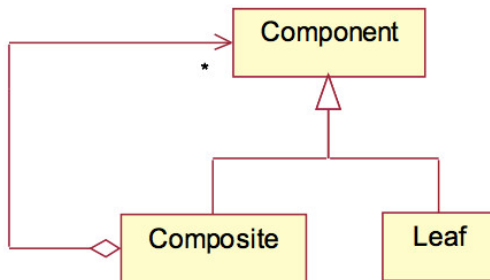
- 0..1
- 1 (shortcut for 1..1)
- * (shortcut for 0..*)
- 1..*
- 5..3 (Invalid!)
- -1..0 (Invalid! All values must be positive)
- 3+5..7+1 (Generally meaningless, but valid; the lower or upper value, respectively is defined by a value specification).

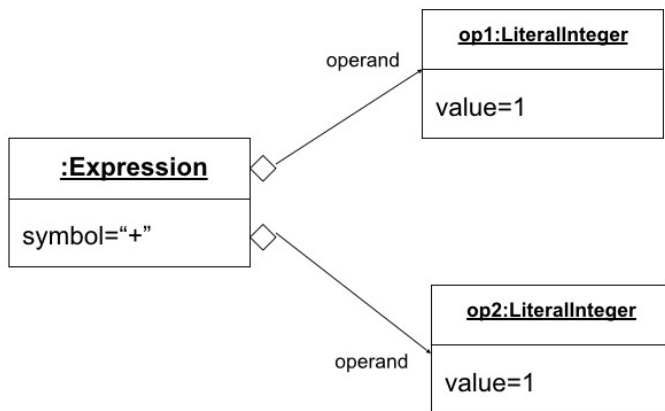
The Multiplicity Metamodel



- **Definition** - A **value specification** indicates one or several values in a model.
- **Semantics** - Examples for value specifications include simple, mathematical expressions, such as $4+2$, and expressions with values from the object model, `Integer::MAX_INT-1`
- In addition, there are language-dependent expressions defined by a language statement and the pertaining expression in that language (opaque expression), such OCL or Java expression (the language statement can be omitted if the language is implicitly defined by the expression or context).

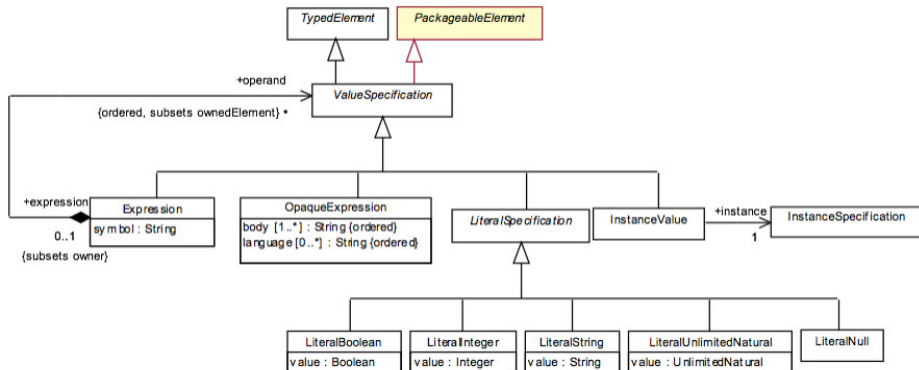
- The metamodel is based on the composite pattern:





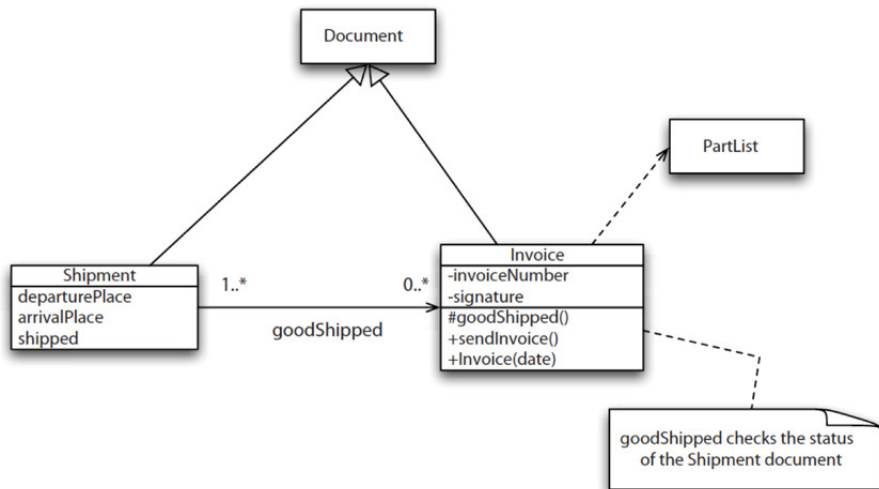
***Object Model for
1+1***

The Metamodel for Value Specifications

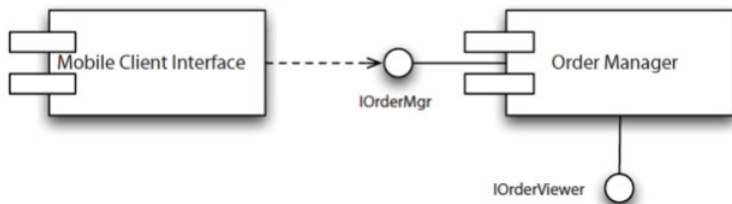


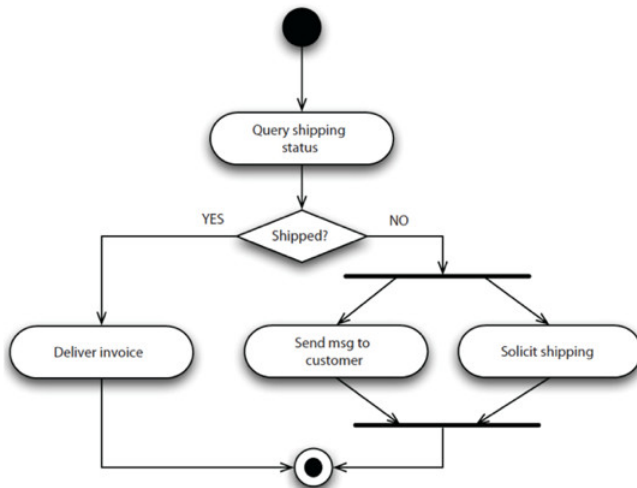


Class Diagram

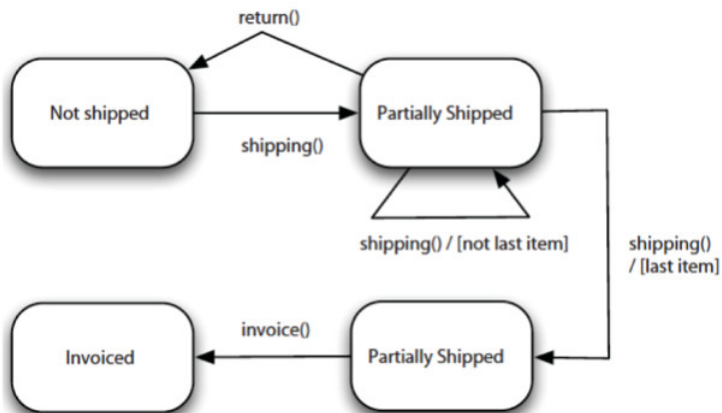


Component Diagram

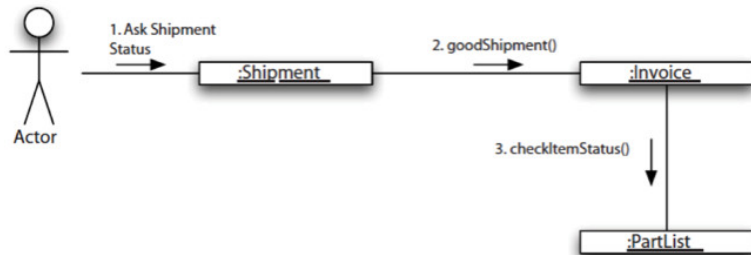
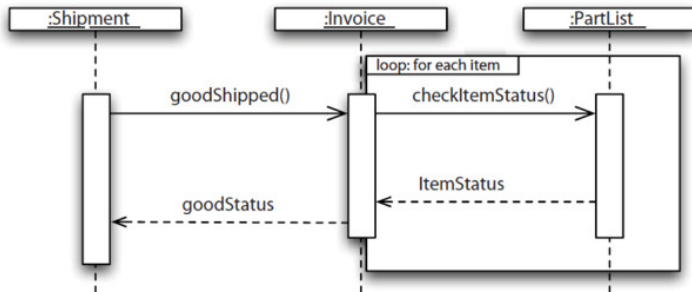


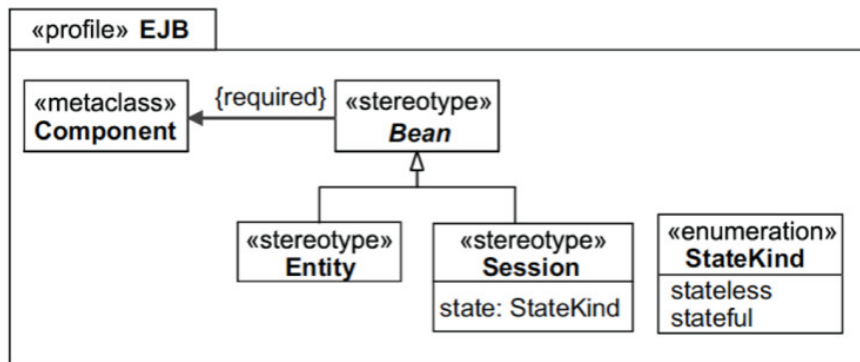


State Diagram



Sequence vs Collaboration Diagrams







OMG Unified Modeling Language™ (OMG UML)

Version 2.5

OMG Document Number formal/2015-03-01

Normative Reference: <http://www.omg.org/spec/UML/2.5>

Associated Normative Machine Consumable Files:

<http://www.omg.org/spec/UML/20131001/PrimitiveTypes.xml>

<http://www.omg.org/spec/UML/20131001/UML.xml>

<http://www.omg.org/spec/UML/20131001/StandardProfile.xml>

<http://www.omg.org/spec/UML/20131001/UMLDI.xml>

Version 2.5 is formally a minor revision to the UML 2.4.1 specification, having been substantially re-written as solicited by the UML Specification Simplification RFP ad/09-12-10. It supersedes formal/2011-08-06 (Infrastructure) and formal/2011-08-06 (Superstructure).

"OMG Unified Modeling Language (OMG UML)", Available at
<http://www.omg.org/spec/UML/2.5/>

... and also Explore the UML Metamodel in EMF

The screenshot displays the Eclipse IDE interface for exploring the UML Metamodel in EMF. The main window is divided into several panes:

- Types (0) ↑ ↓**: A list of UML metamodel elements. The 'EClass (243)' entry is selected.
 - EAnnotation (568)
 - EAttribute (115)
 - EClass (243)**
 - EEnum (13)
 - EEnumLiteral (62)
 - EGenericType (3693)
 - EOperation (806)
 - EPackage (1)
 - EParameter (1106)
 - EReference (510)
 - EStringToStringMapEntry (86)
- Instances**: A tree view showing the structure of the 'DataType' class.
 - DataType
 - [EClass] DataType -> Classifier [org.eclipse.uml2.uml]
 - /eContainer
 - name = DataType
 - instanceClassName = org.eclipse.uml2.uml.DataType
 - /instanceClass = interface org.eclipse.uml2.uml.Data
 - /defaultValue =
 - instanceTypeName = org.eclipse.uml2.uml.DataType
 - abstract = false
 - interface = false
 - eAnnotations (0)
 - ePackage (1)
 - eTypeParameters (0)
- EPackage Registry**: A table listing various UML packages and their URIs.

Name	nsUri
uml	http://www.eclipse.org/uml2/5.0.0/UML
uml	http://www.eclipse.org/uml2/5.0.0/UML
uml	http://www.eclipse.org/uml2/5.0.0/UML
uml2	http://www.eclipse.org/emf/compare/uml2/2.0
umlCollaborationUse	http://www.eclipse.org/papyrus/uml/textedit/collaborationuse/xttext/UmlC...
umlCommon	http://www.eclipse.org/papyrus/uml/textedit/common/xttext/UmlCommon