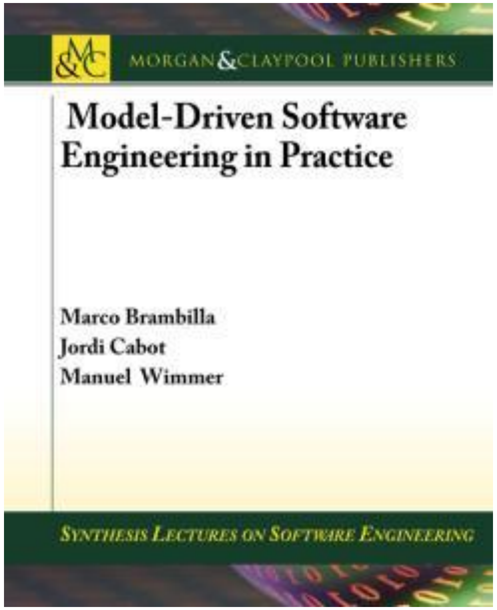
MORGAN & CLAYPOOL PUBLISHERS

Chapter #4

MODEL DRIVEN ARCHITECTURE (MDA)

Teaching material for the book
Model-Driven Software Engineering in Practice
by Marco Brambilla, Jordi Cabot, Manuel Wimmer.
Morgan & Claypool, USA, 2012.



Contents

- MDA
- UML (from a metamodeling perspective)

Model Driven Architecture

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

MDA (Model-Driven Architecture)

Four principles of MDA

- **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

Definitions according to MDA

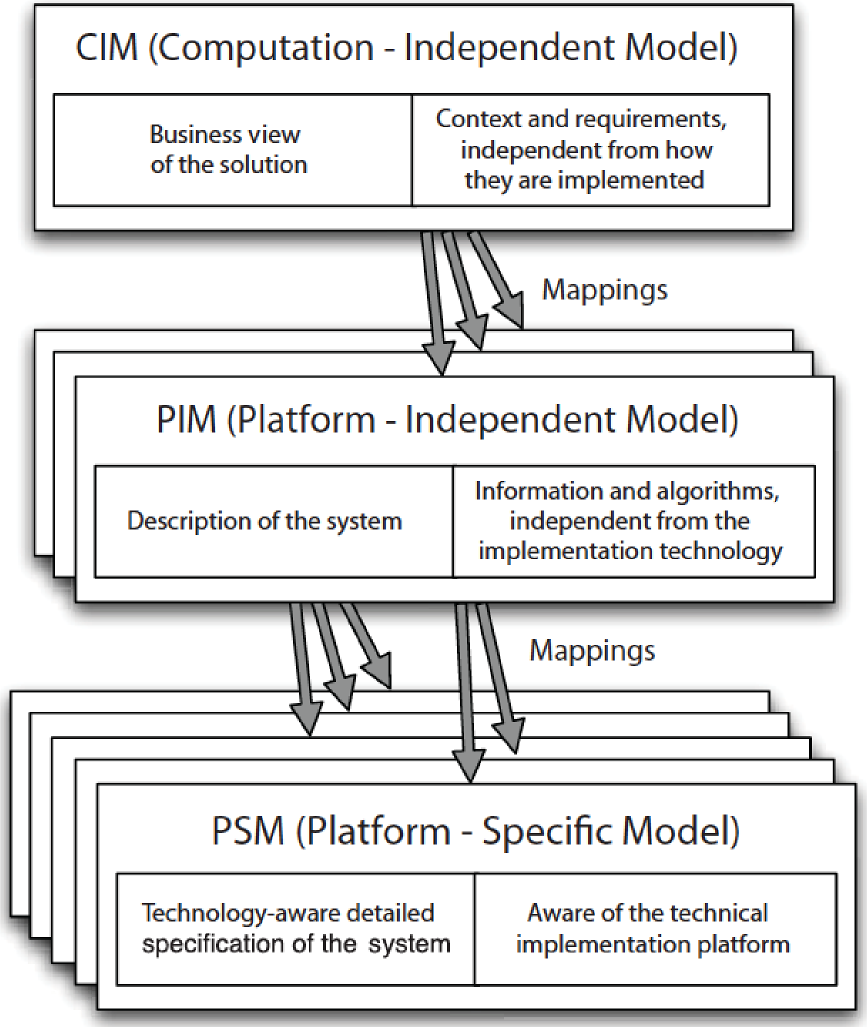
- **System**: The subject of any MDA specification (program, computer system, federation of systems)
- **Problem Space** (or **Domain**): The context or environment of the system
- **Solution Space**: The spectrum of possible solutions that satisfy the reqs.
- **Model**: Any representation of the system 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

Modeling Levels

CIM, PIM, PSM

- **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.

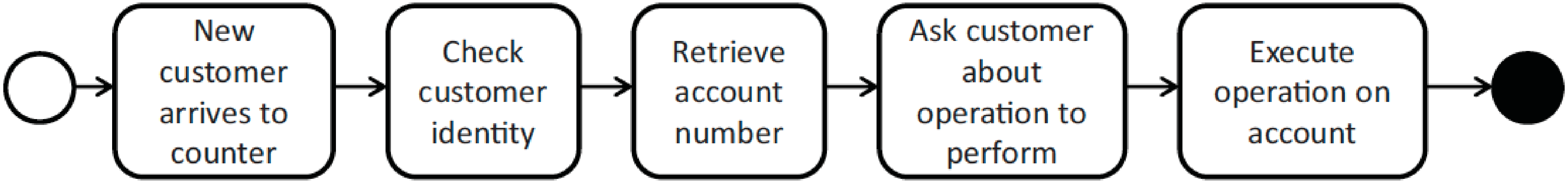
CIM, PIM and PSM



CIM

MDA Computation Independent Model (CIM)

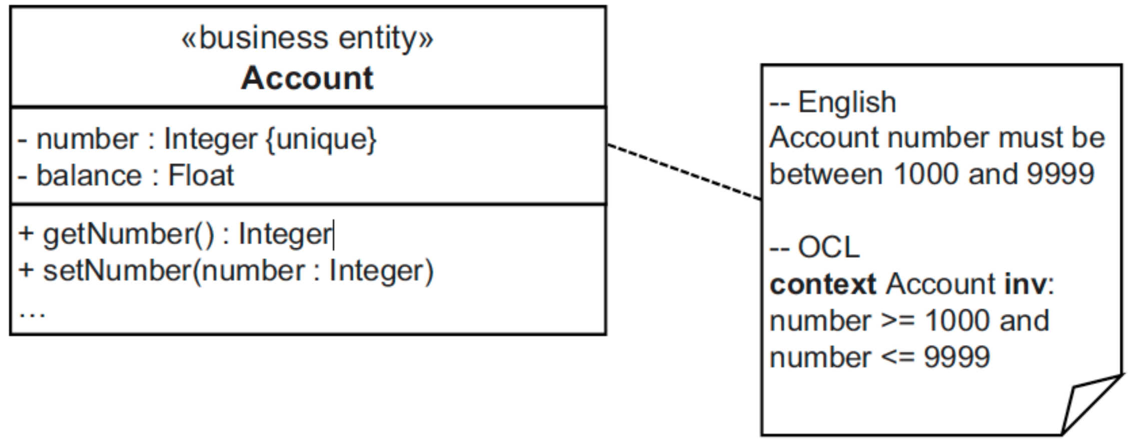
- E.g., business process



PIM

MDA Platform Independent Model (PIM)

- Specification of structure and behaviour of a system, abstracted from technologicical details

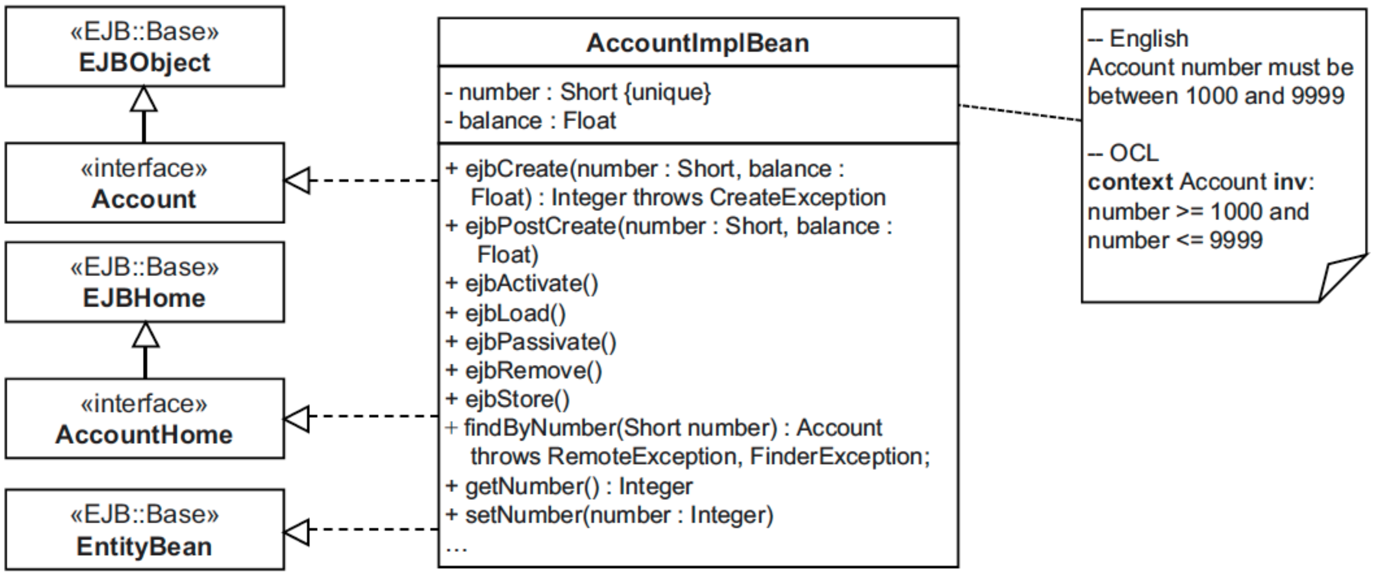


- Using the UML(optional)

- 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

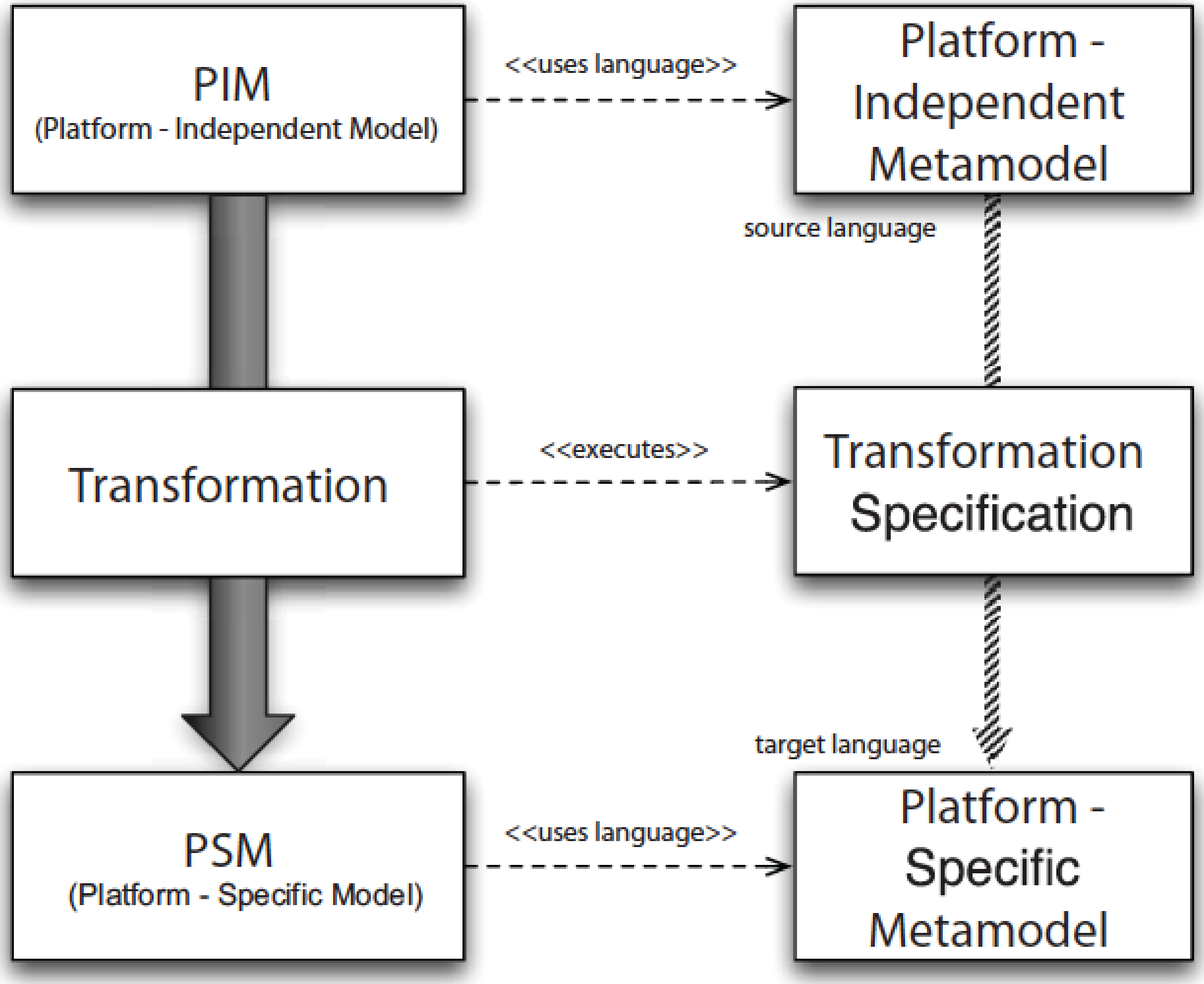
PSM

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

CIM – PIM – PSM mappings



Modeling language specification

- 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

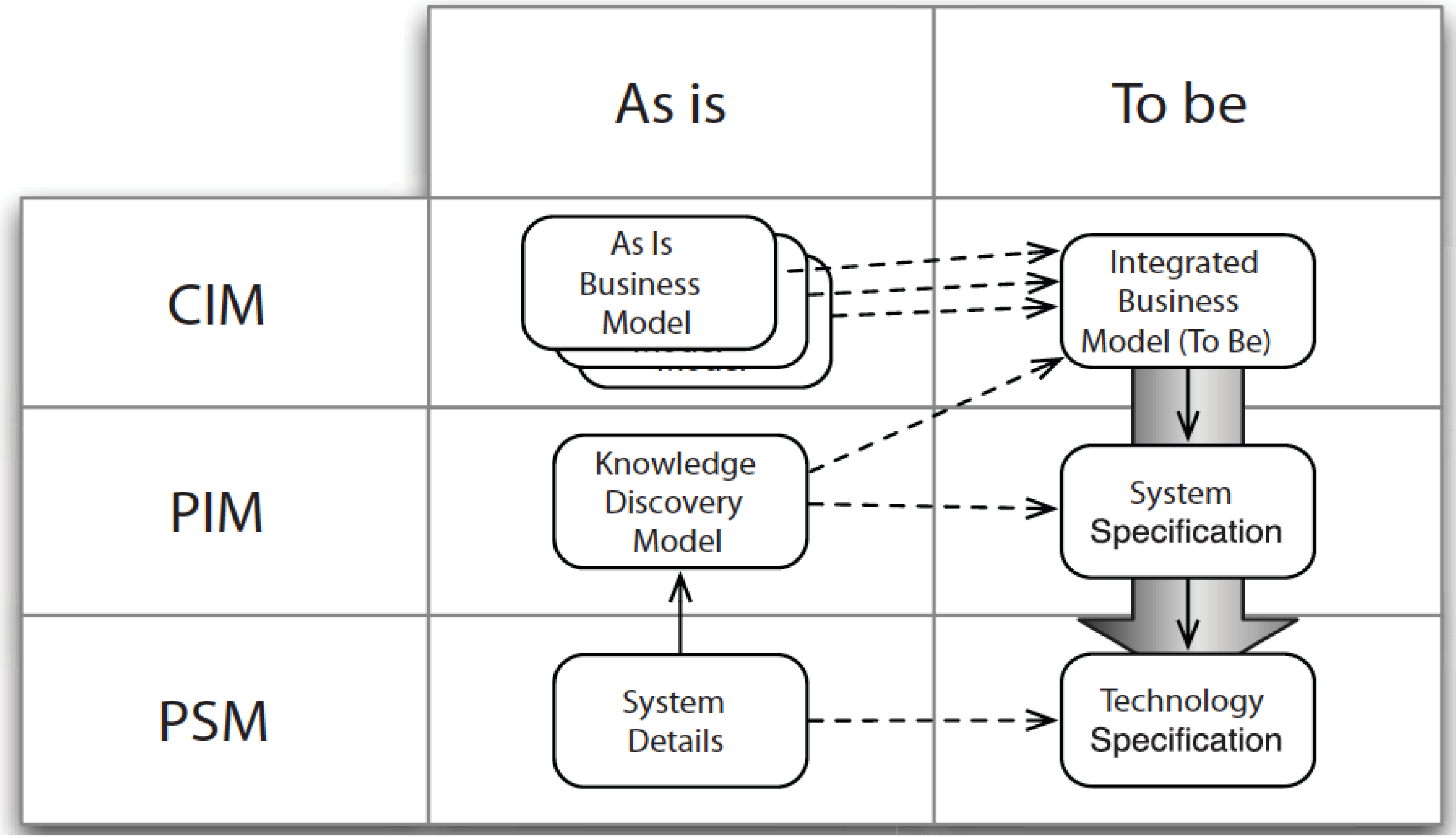
ADM

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

It includes several standards that help on this matter

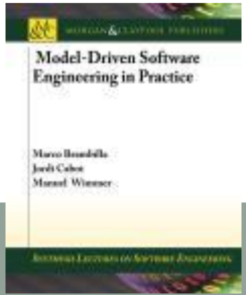
- **The Knowledge Discovery Metamodel (KDM)**: An intermediate representation for existing software systems that defines common metadata required for deep semantic integration of lifecycle management tools. Based on MOF and XMI
- **The Software Measurement Metamodel (SMM)**: A meta-model for representing 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.

MDA vs. ADM – the MDRE process



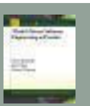
MOF – META OBJECT FACILITY

UML – UNIFIED MODELING LANGUAGE

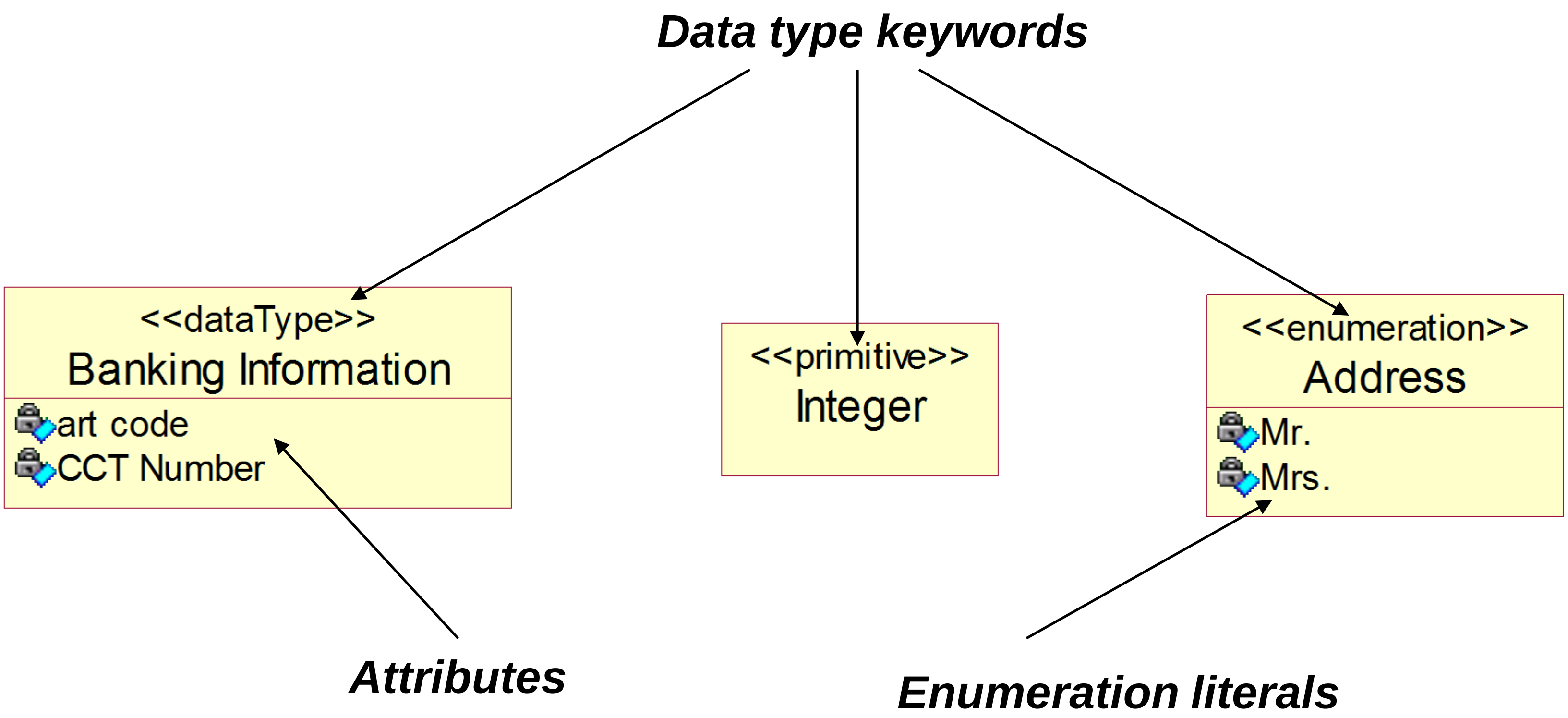


Datatypes

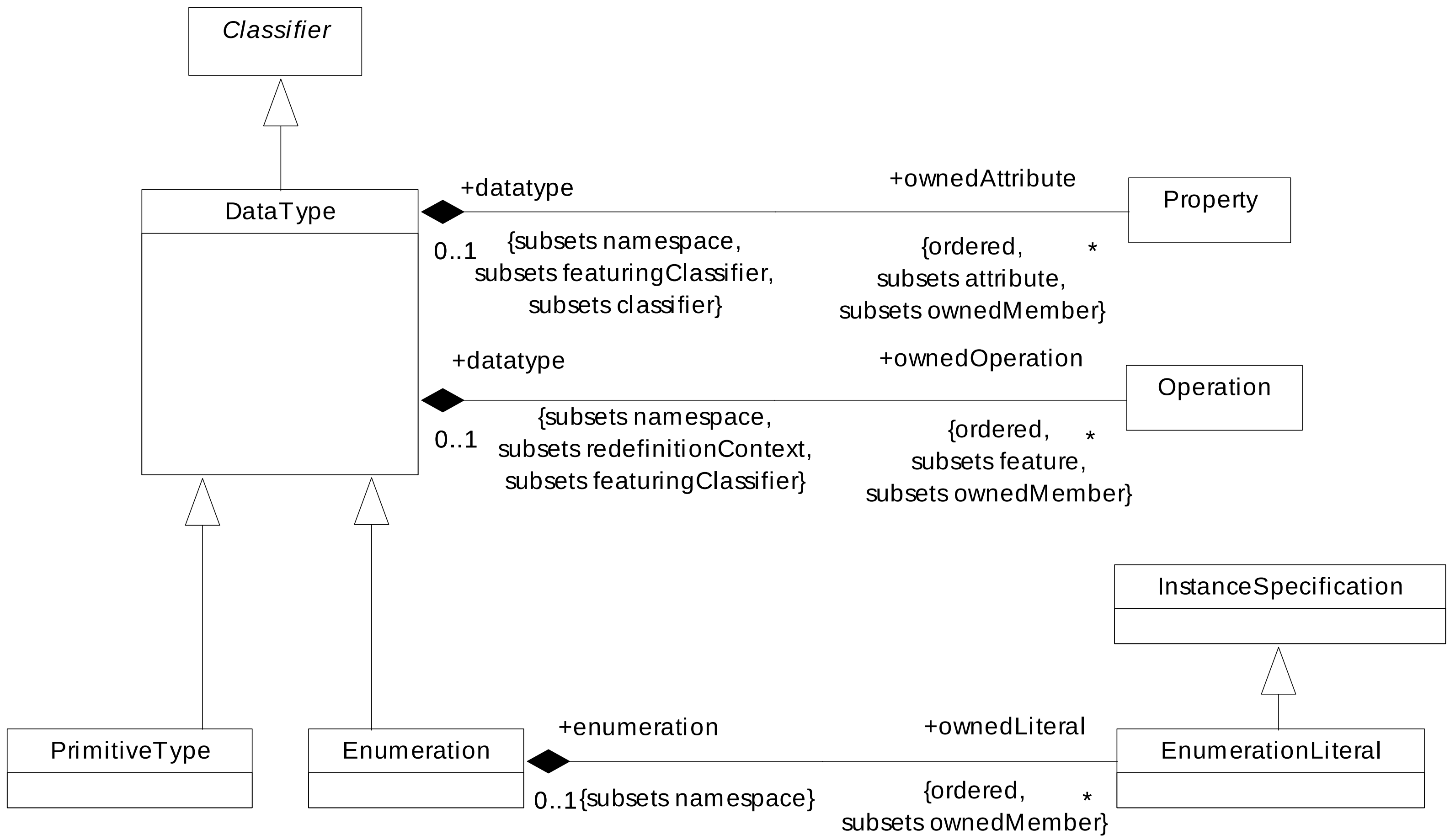
- UML distinguishes between the following data types:
  - **Simple data types** (*DataType*): a type with values that have no identity; that means two instances of a datatype with the same attributes values are indistinguishable.
  - **Primitive data types** (*PrimitiveType*): a simple data type without 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 plus infinite (*).
  - **Enumeration types** – simple data types with values that originate from a limited set of enumeration literals.



Examples of data types



The metamodel of data types



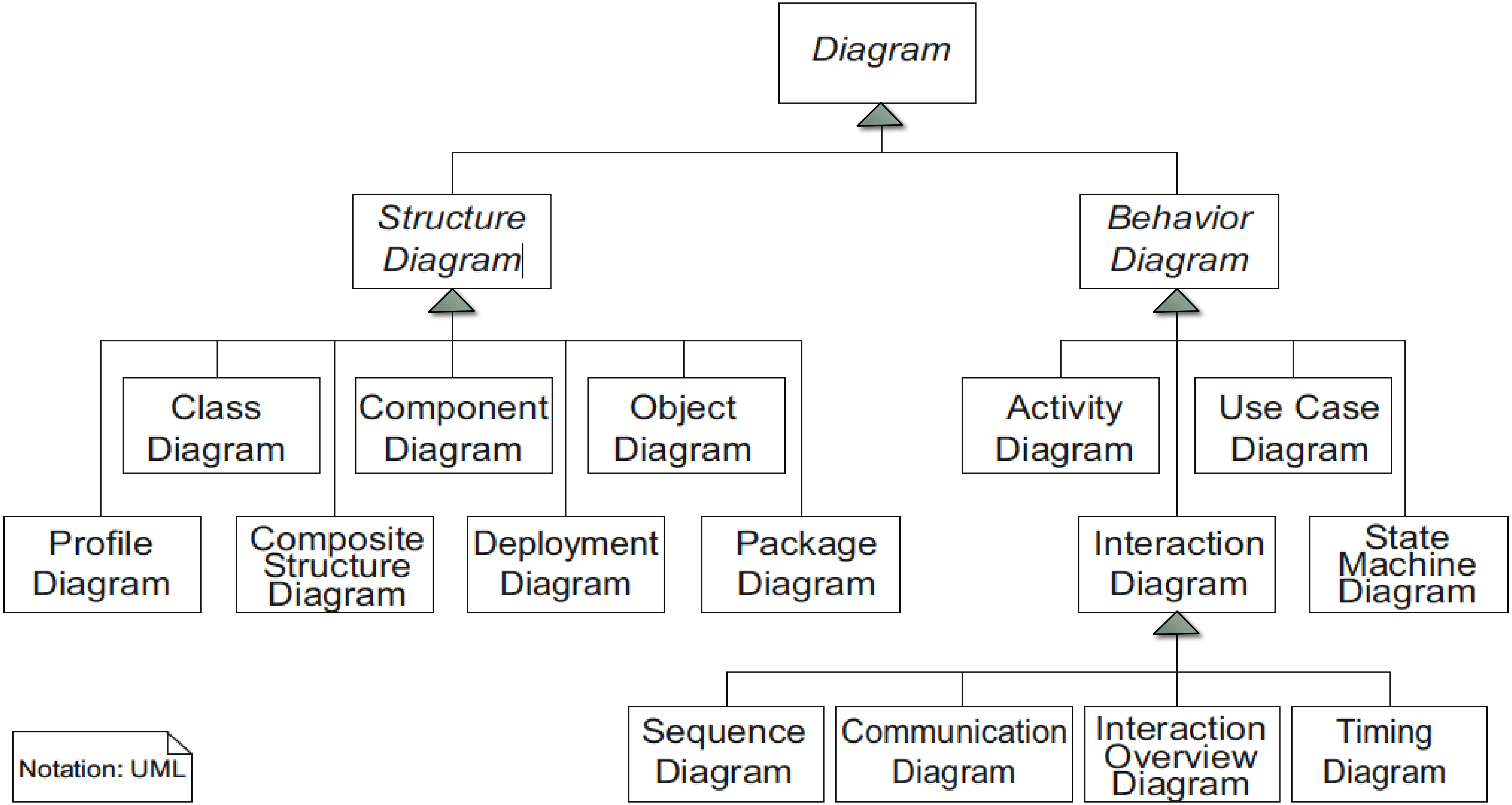
Overview of Diagrams

- 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.

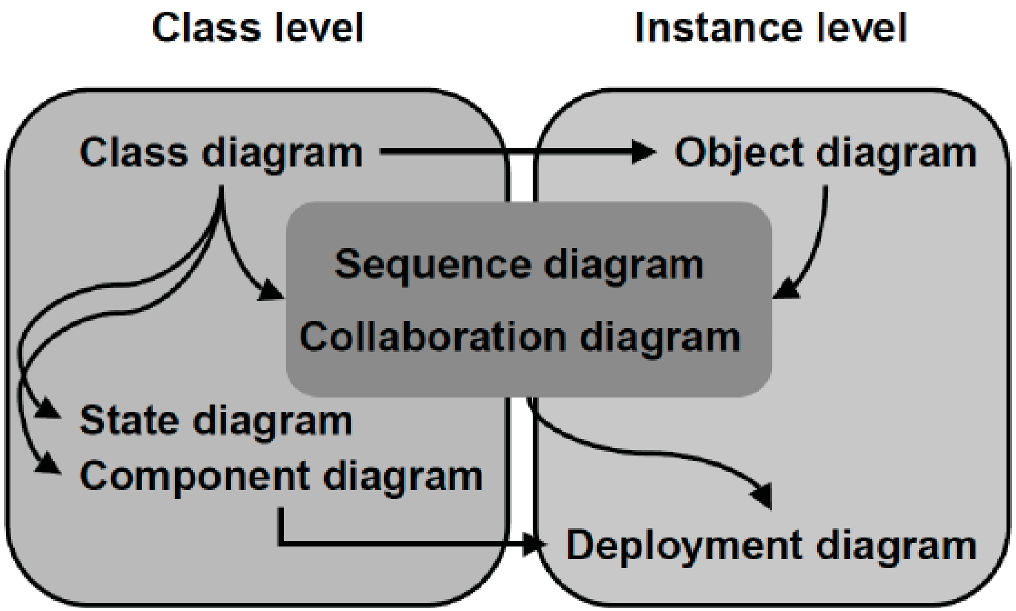
Overview of Diagrams

- 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.

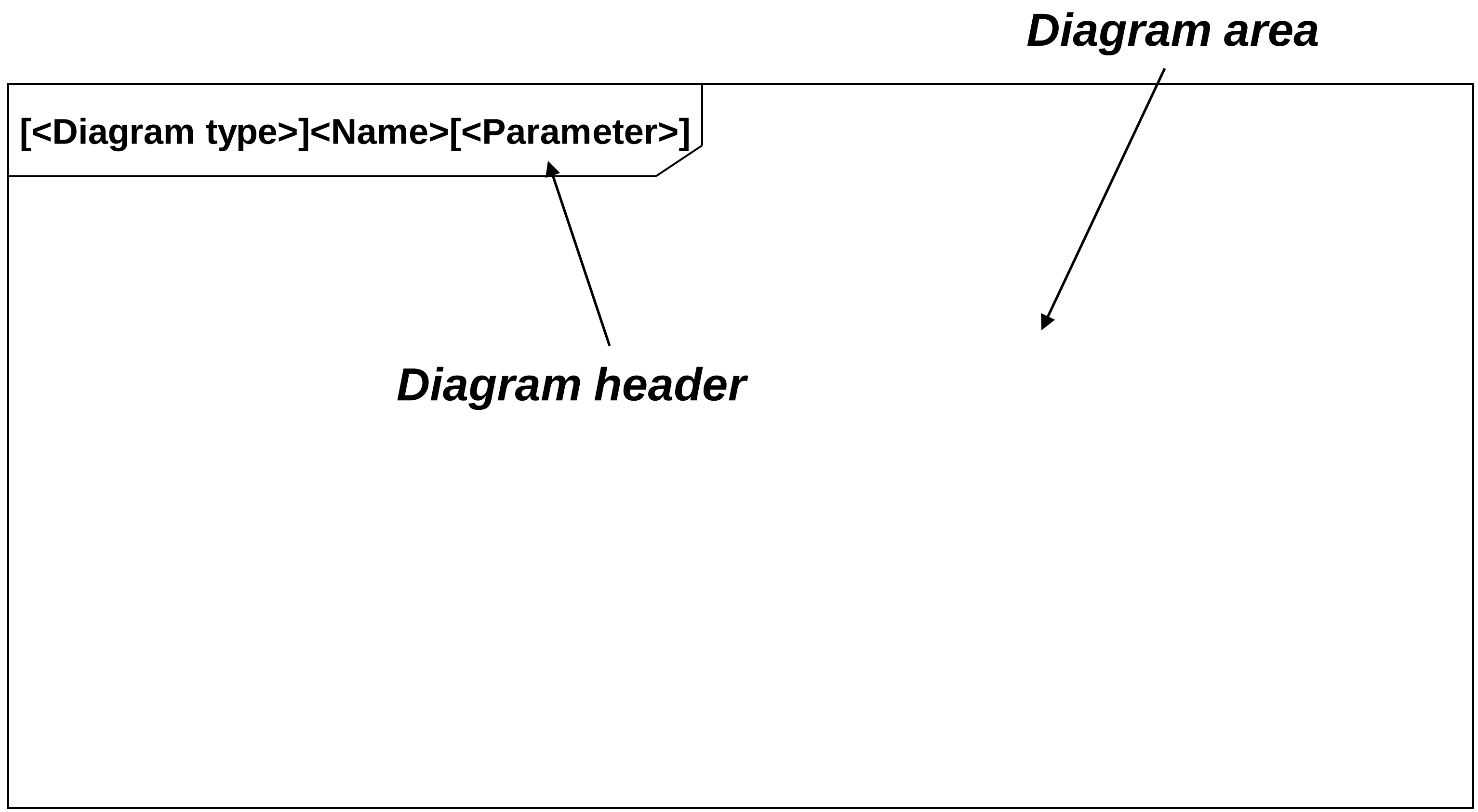
Overview of the UML diagrams



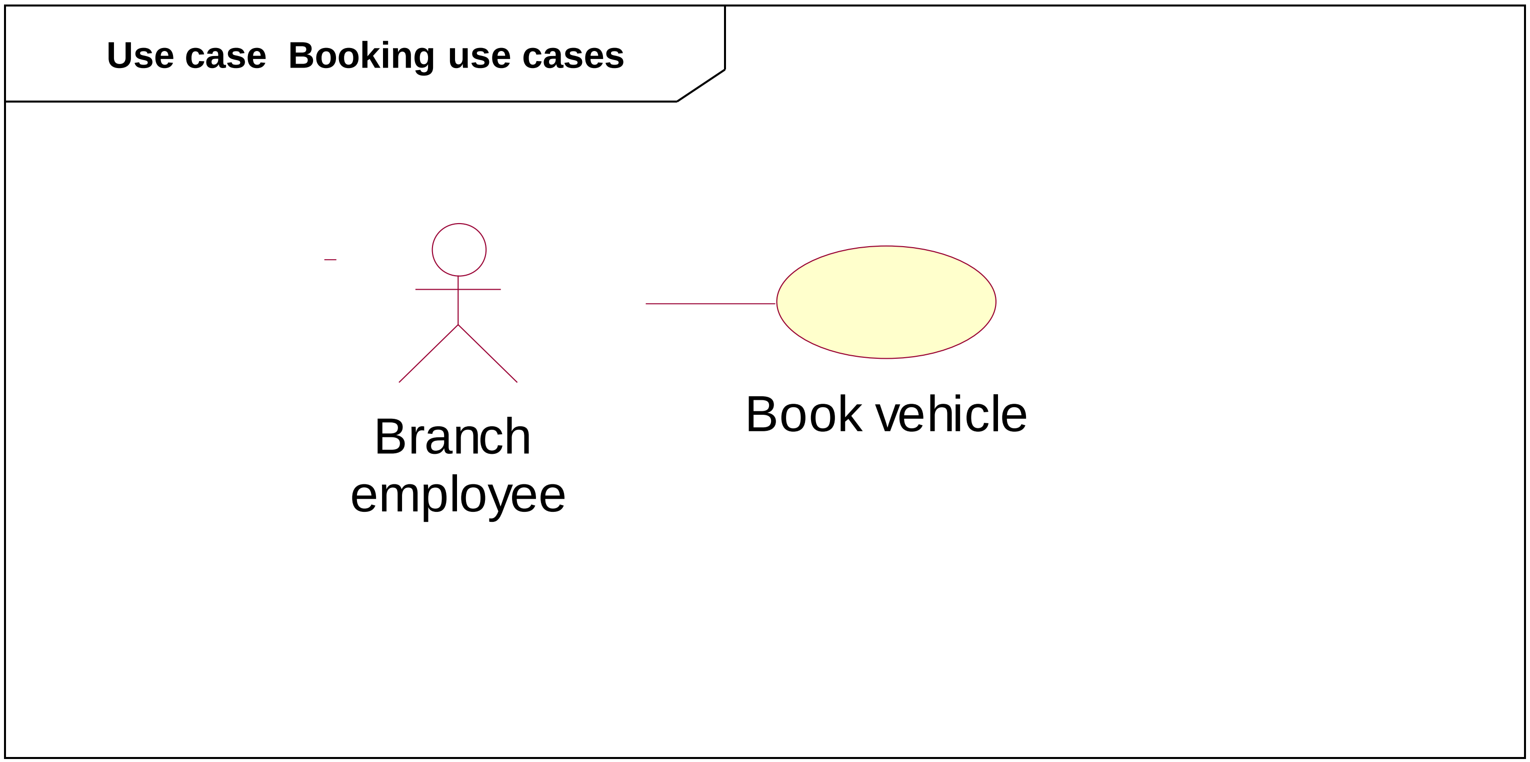
Class vs. instance



Basic notation for diagrams



Example of a use case diagram



Stereotypes-definition

- 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**.

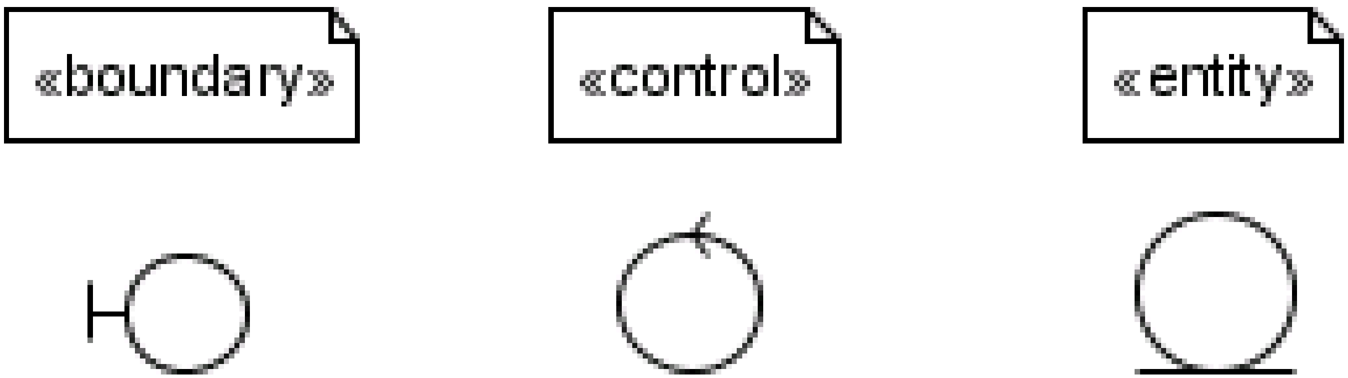
Multiple stereotyping

- **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.

Stereotypes Notation

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

Graphical symbols



UML standard stereotypes

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

UML standard stereotypes

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

UML standard stereotypes

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)

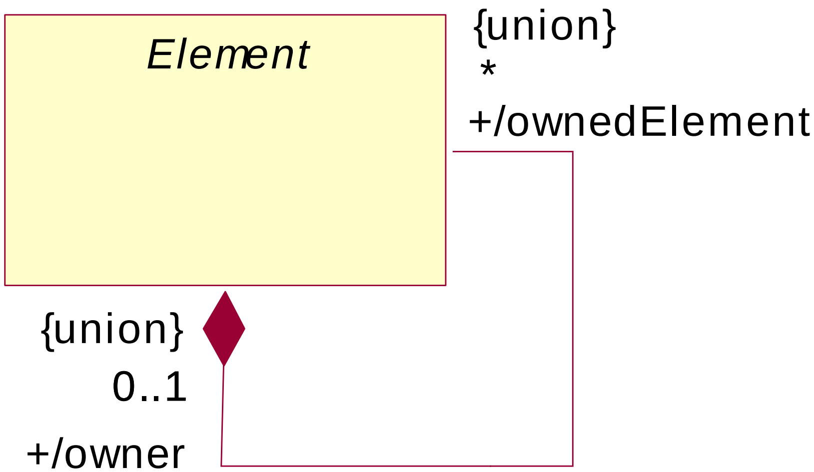
Class Diagrams

- Class Diagrams refer to this area of the metamodel:
  - Package: ***Classes::Kernel***
  - Package: ***Classes::Dependencies***
  - Package: ***Classes::Interfaces***

Class Diagrams: basic concepts

- 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.

The basic UML class

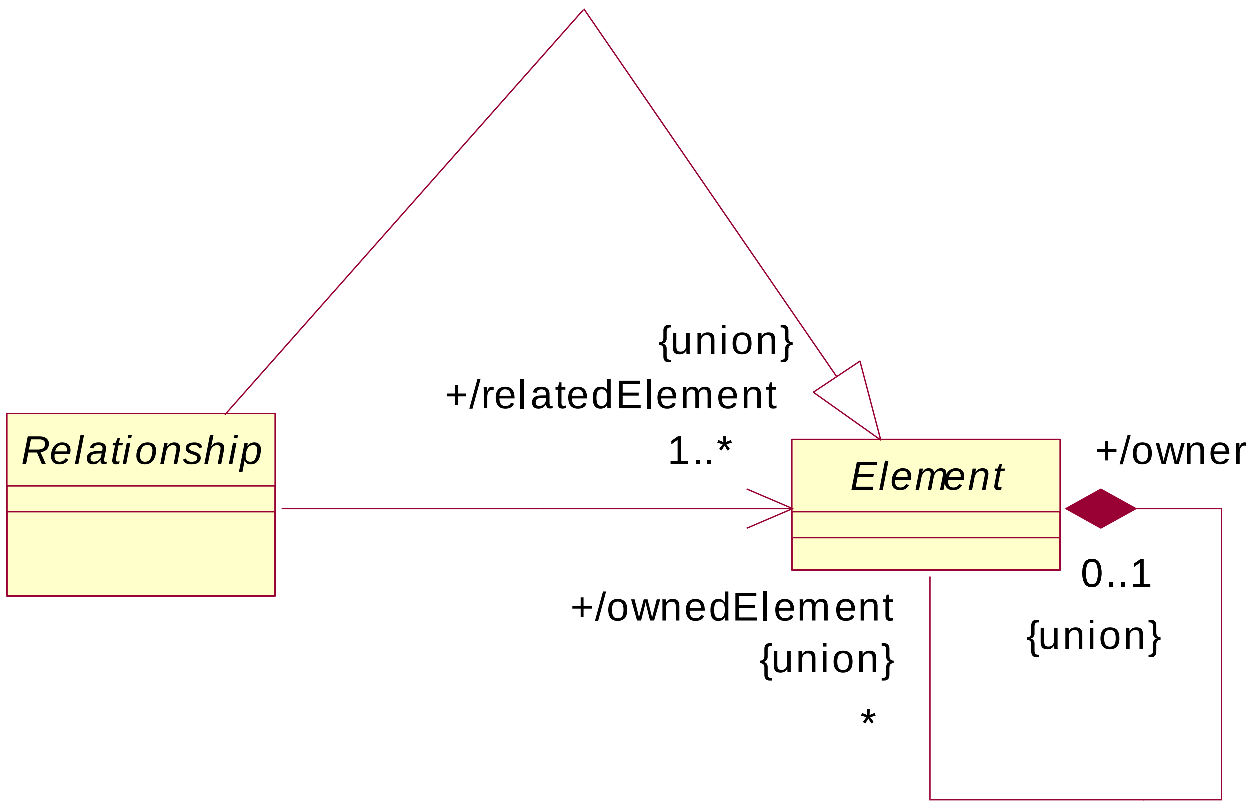


There is no notation for an element because you would never use the element construct in UML models. The class is abstract.

Relationship

- 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.

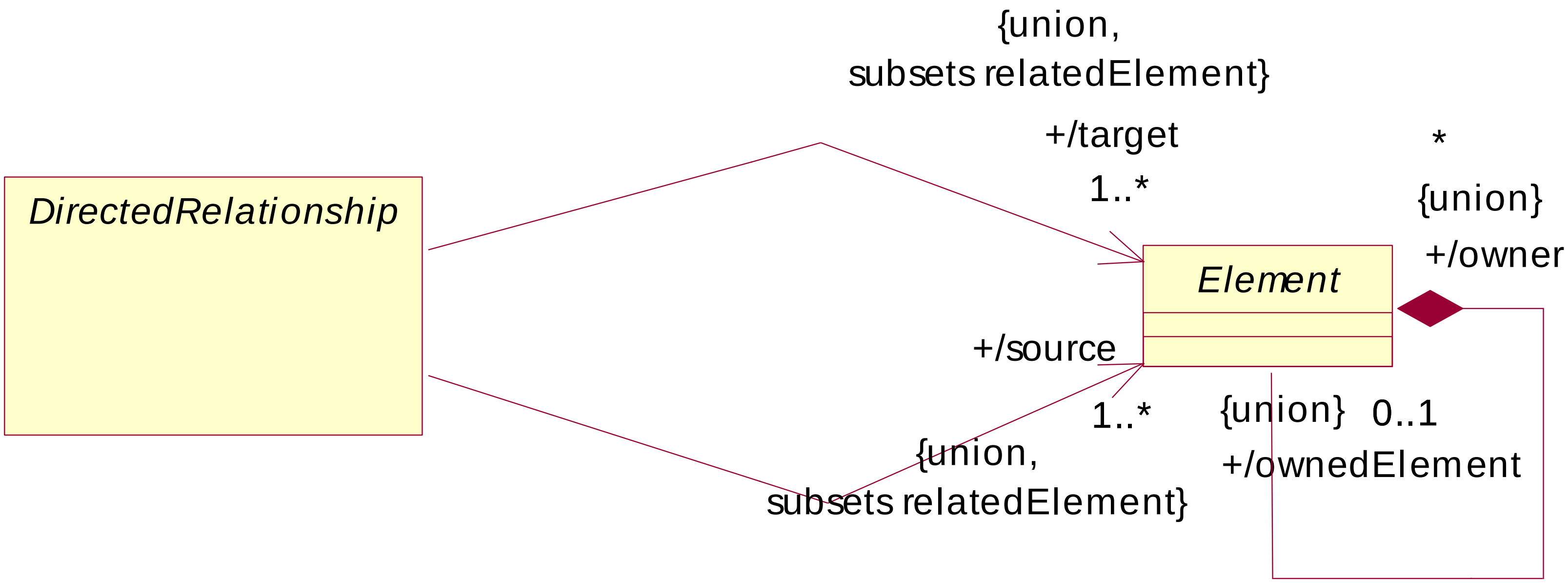
The basic Relationship class



Supplier and client

- 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 later is a ***client***. This is expressed **in one direction**.

Directed relationships

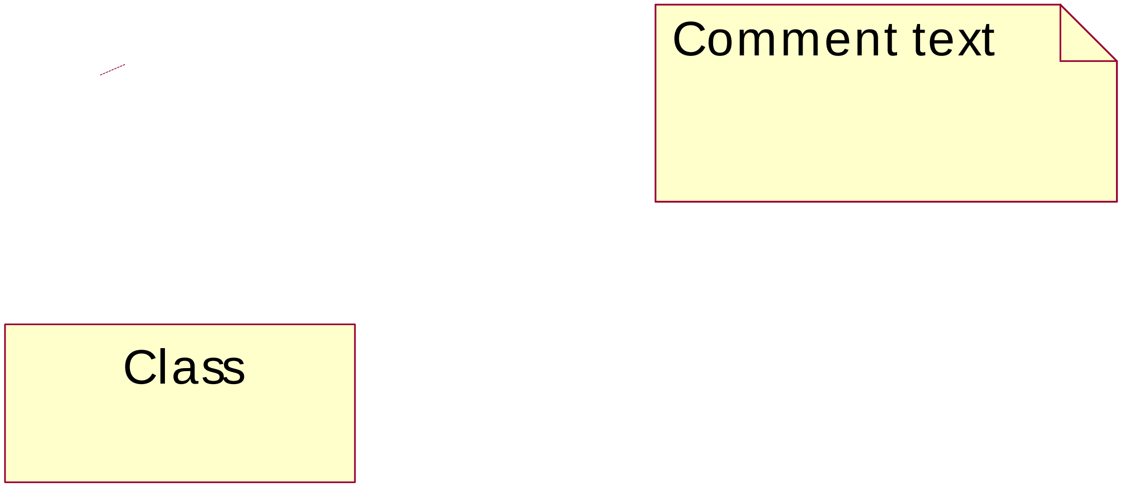


Note that we are dealing only with abstract and rather simple concepts.

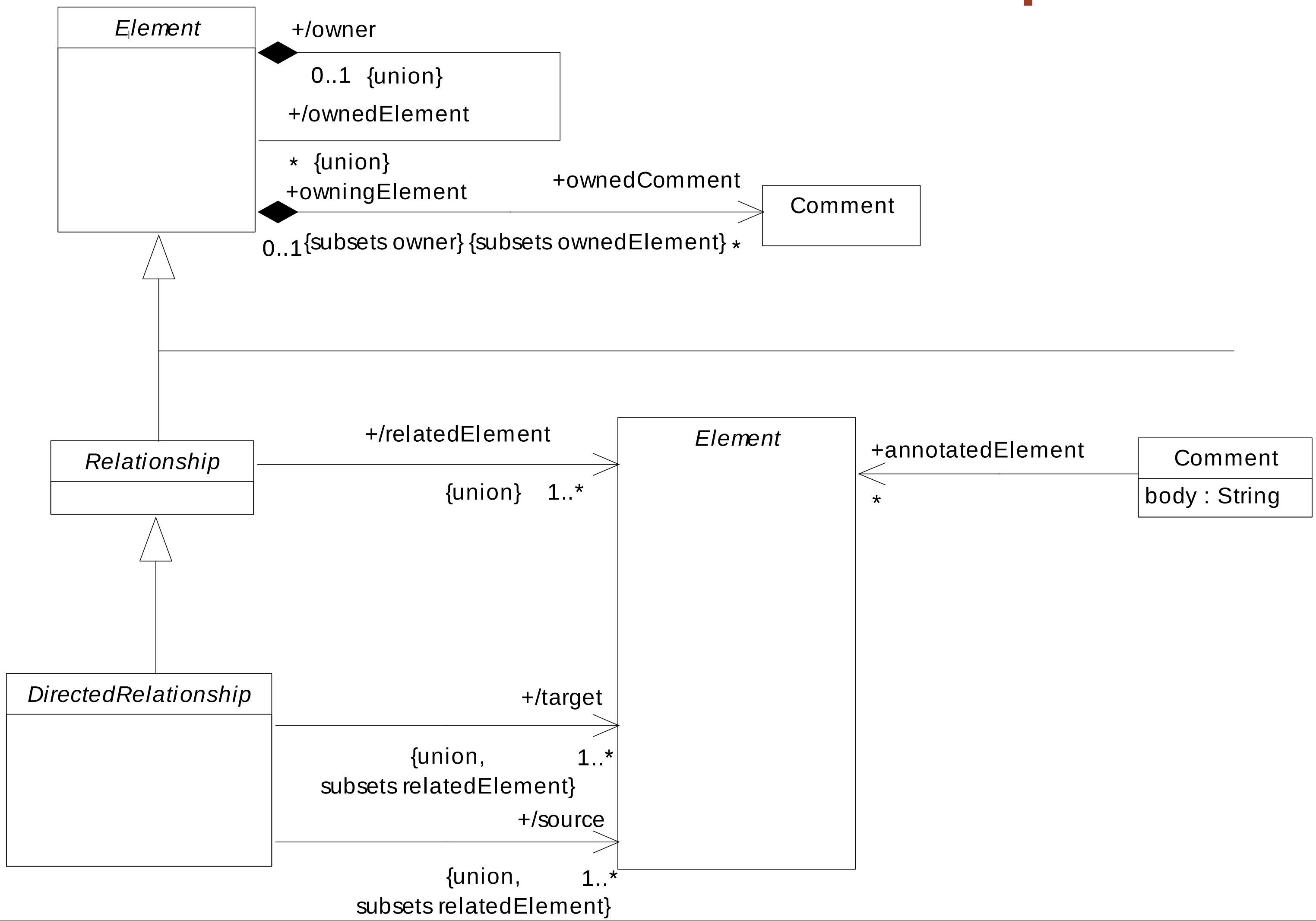
Coments and notes

- 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 notation for comments



The basic metamodel concepts

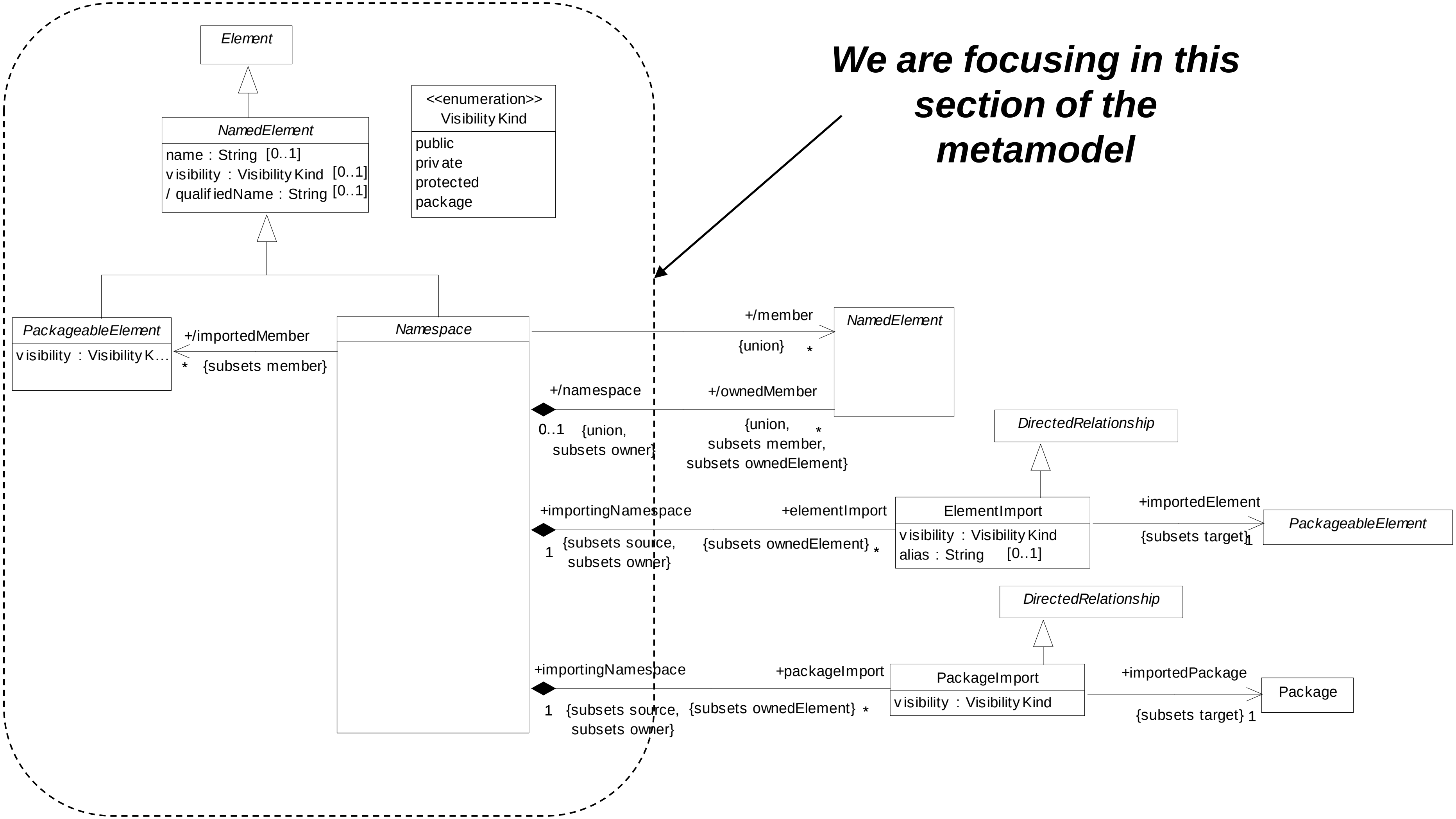


Namespaces

- **Def**.-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

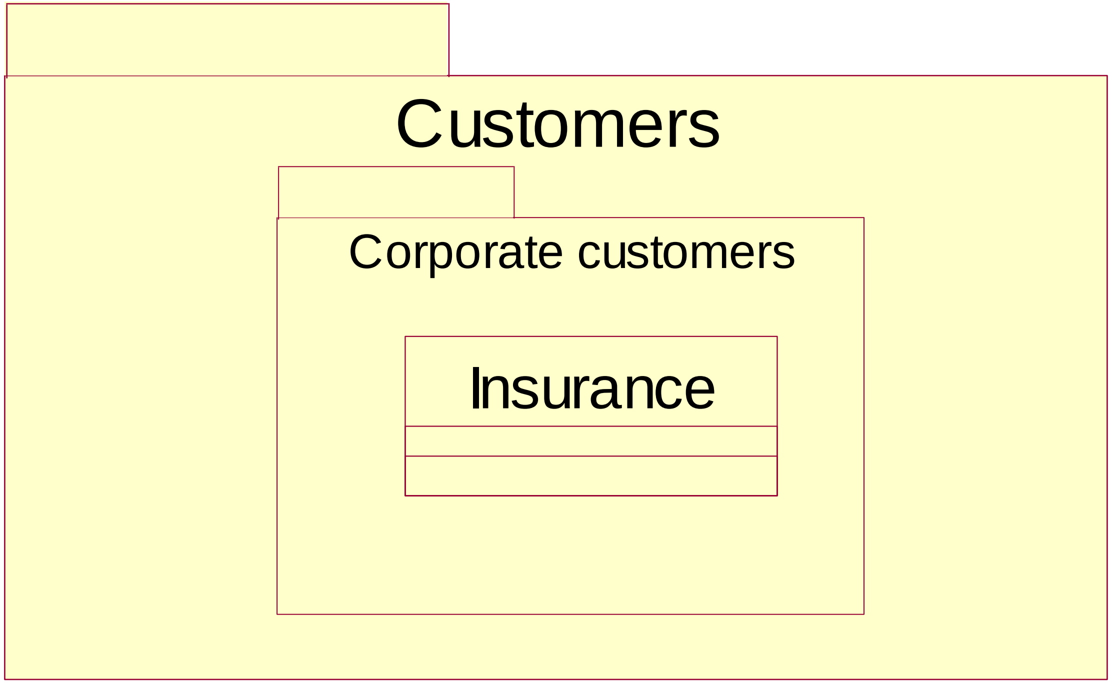
We are focusing in this section of the metamodel



Namespace

- 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.

Nested namespaces



Qualified name

Customers::CorporateCustomers:Insurance

Packageable element

- 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 can.
- The visibility statement is mandatory for a packageable element.

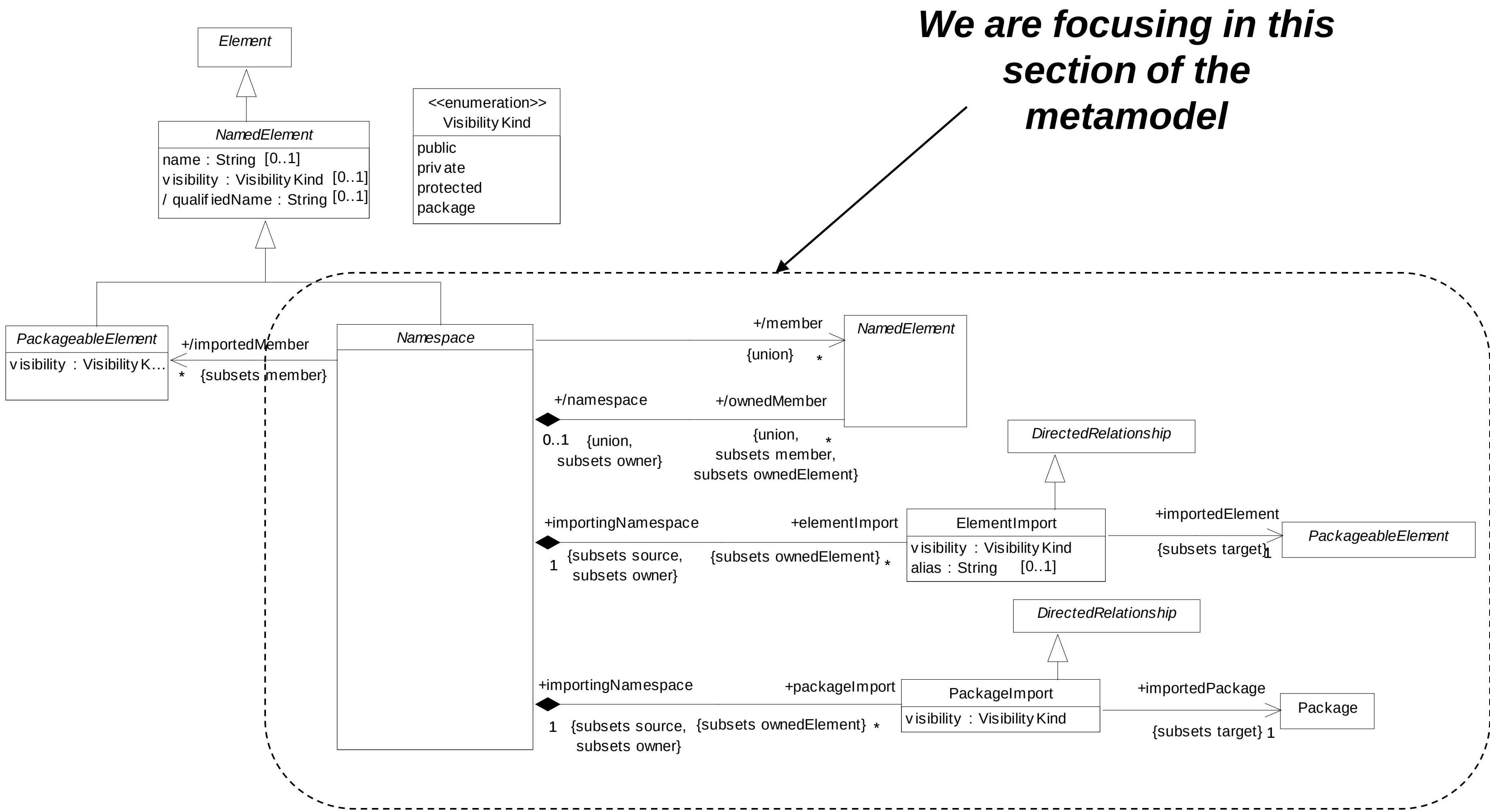
ElementImport

- 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.

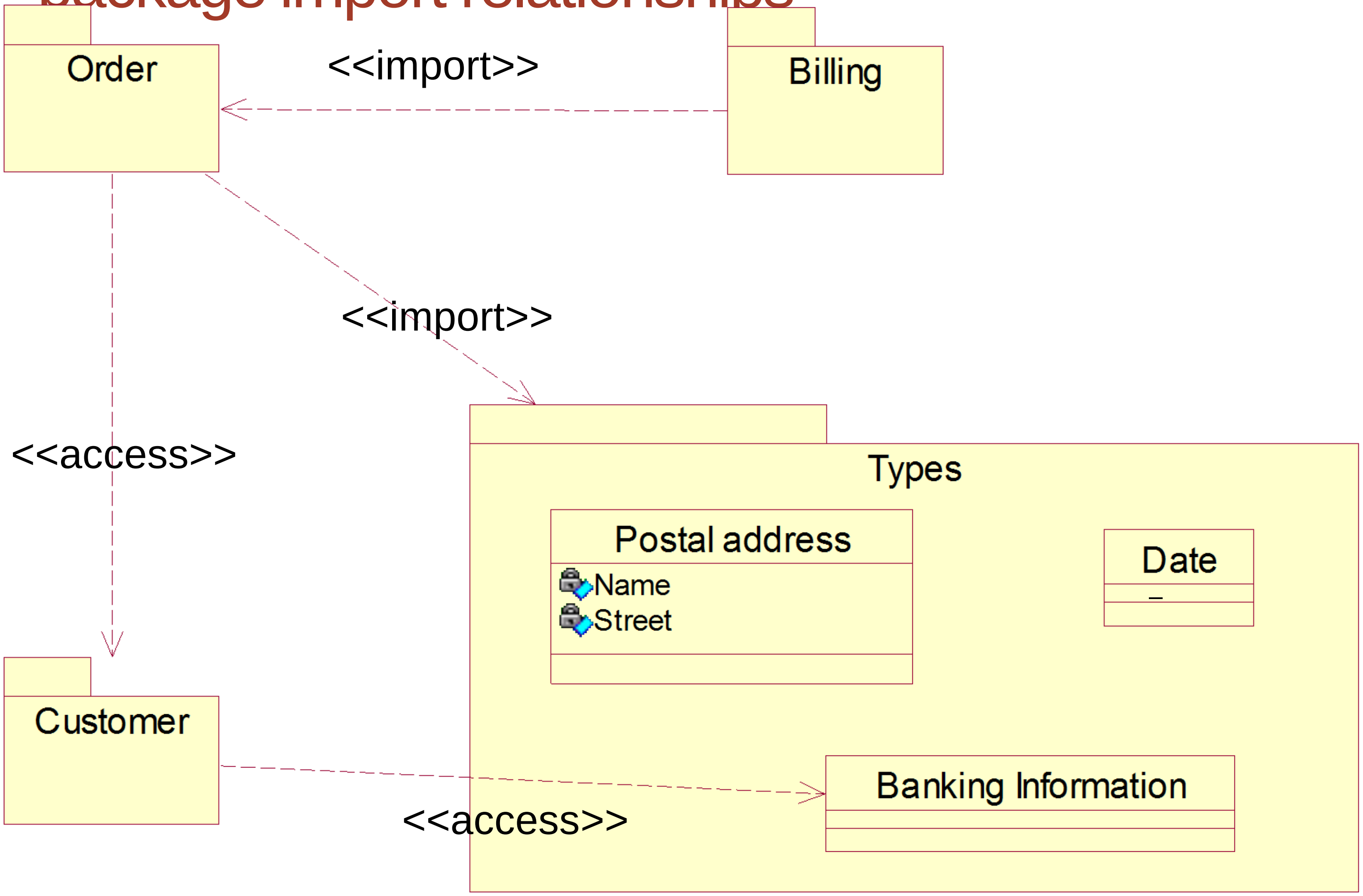
PackageImport

- 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

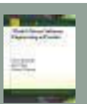


Example of element and package import relationships



<<access>> and <<import>>

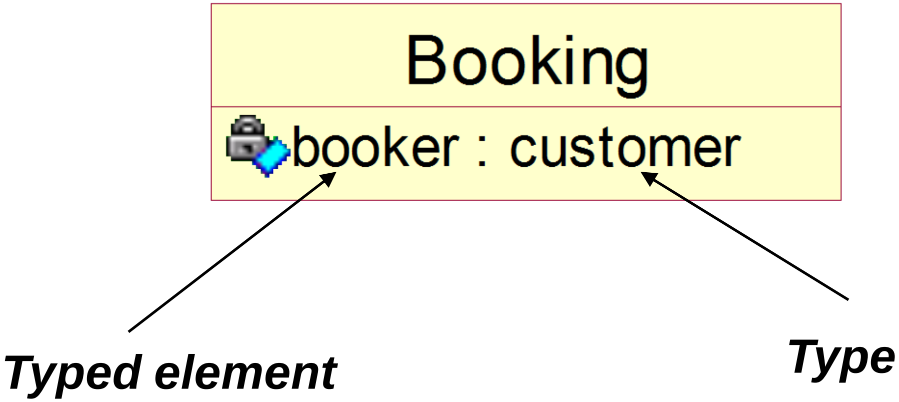
- **<<import>>**: The visibility is ***public***; for example, the postal address for *Order*. The public import is a transitive relationship: if *A* imports *B* and *B* imports *C*, then *A* is indirectly importing *C* too.
- **<<access>>**: The visibility is ***private***, not public: *Customer* is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



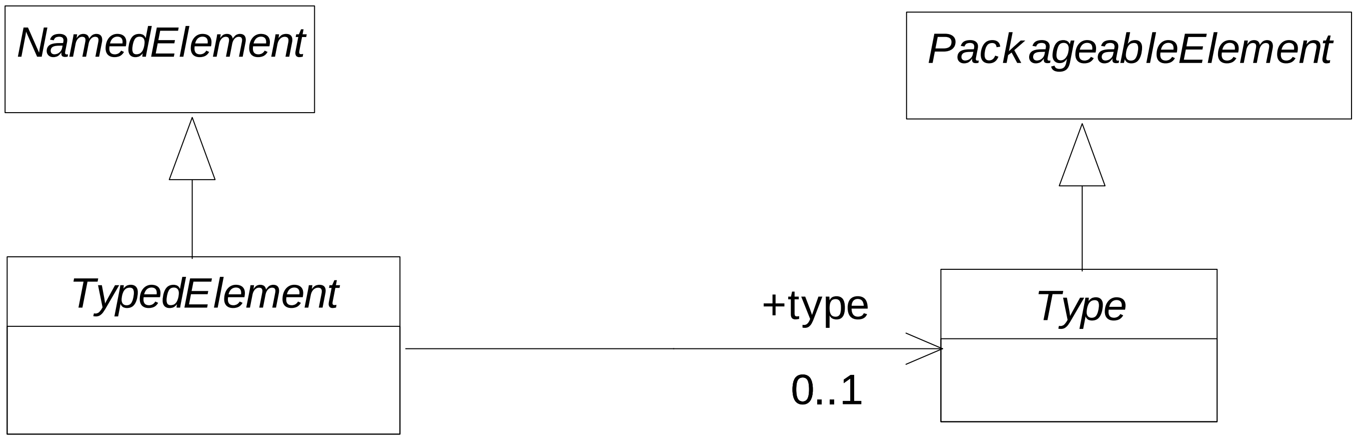
Typed elements

- 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.-** Symple data types and classes are types.

Example – typed element & type



Typed elements metamodel

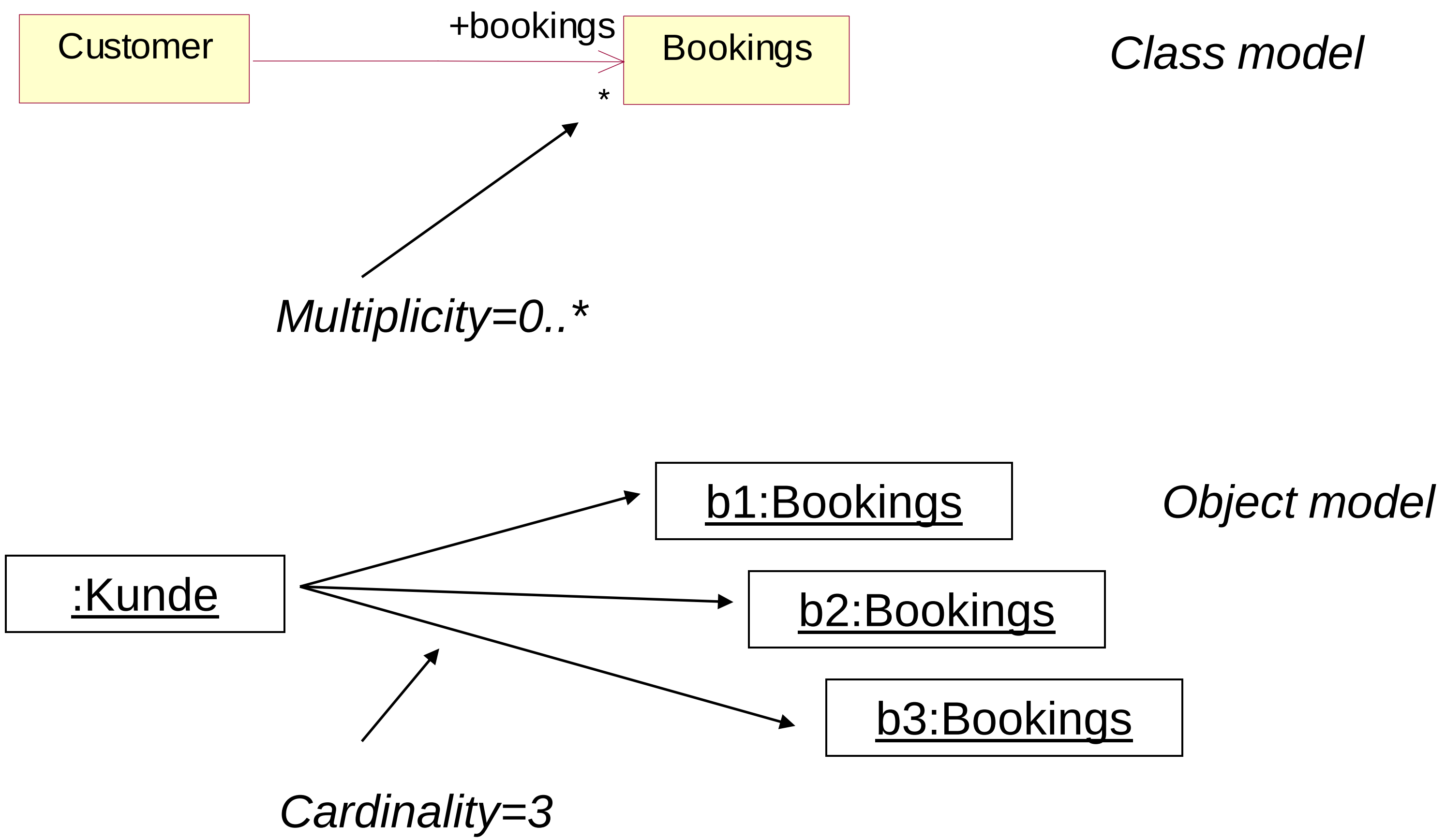


Type and typed element are abstract classes.
They have no properties

Multiplicities

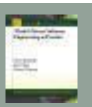
- A ***multiplicity element*** is the definition of an interval of positive integers to specify allowable cardinalities.
- A ***cardinality*** is a concrete number of elements in a set.
- A multiplicity element is often simply called ***multiplicity***; the two terms are synonymous.

Example Multipicity & Cardinality



Multiplicities

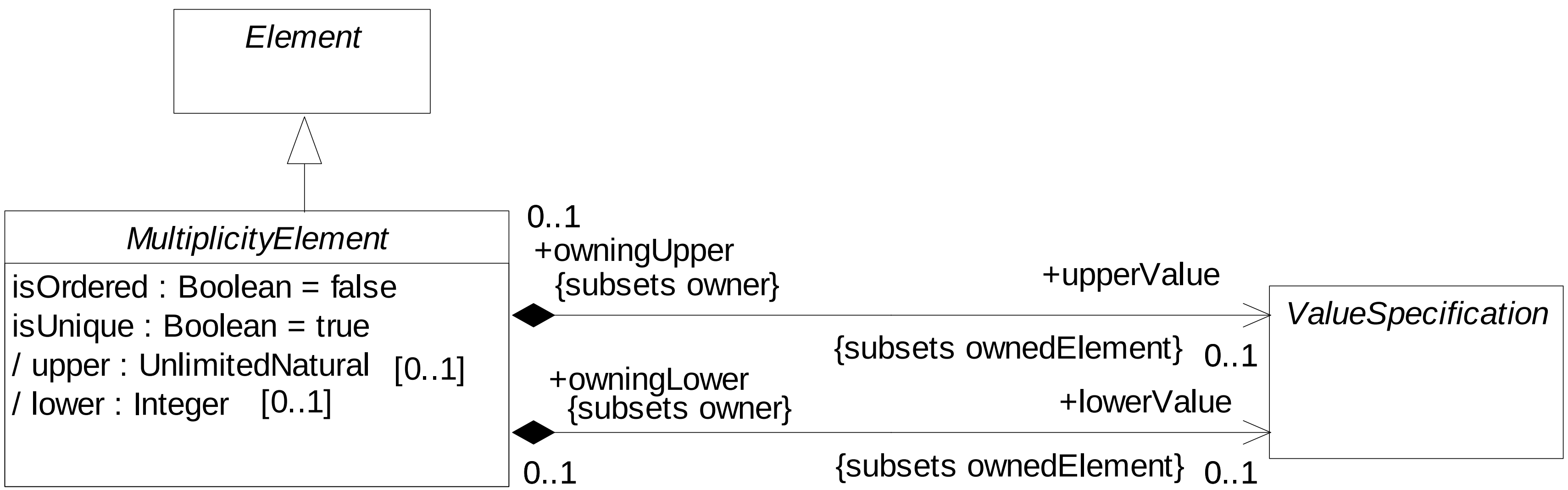
- The ***notation*** for multiplicity is either a *single number* or a *value range*.
- A value range is written by stating the minimum and maximum values, separated by two dots (e.g. 1..5).
- In addtion, you can use the wildcard character ***** to specify an arbitrary number of elements.



Examples of multiplicities

- 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 meaningles, but valid; the lower or upper value, respectively is defined by a *value specification*).

The multiplicity metamodel



Checklist: multiplicities

1. What value range is described by a multiplicity?
2. What is the difference between multiplicity and cardinality?

Value specification

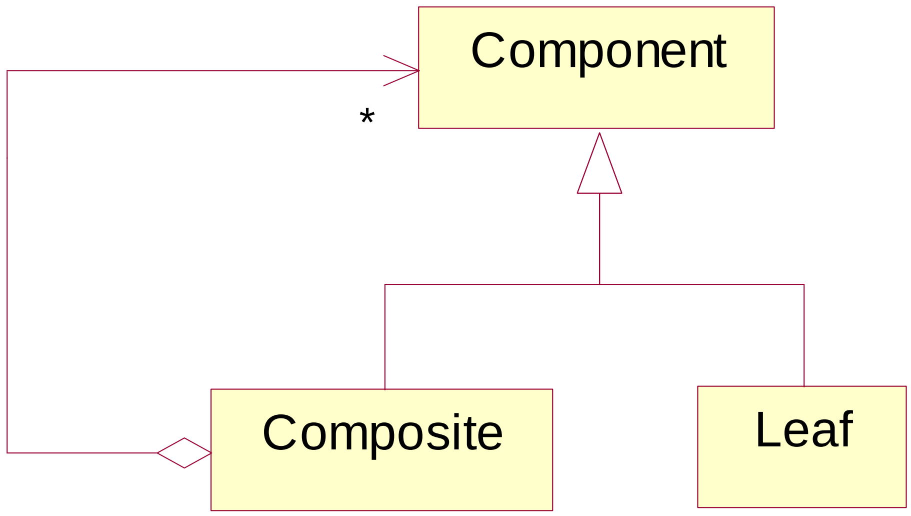
- **Def.-** A ***value specification*** indicates one or several values in a model.
- **Semantics.-** Examples for value specifications include simple, mathematical expressions, such as 4+2, and expressions with values from the object model, Integer::MAX_INT-1

Value specification-semantics

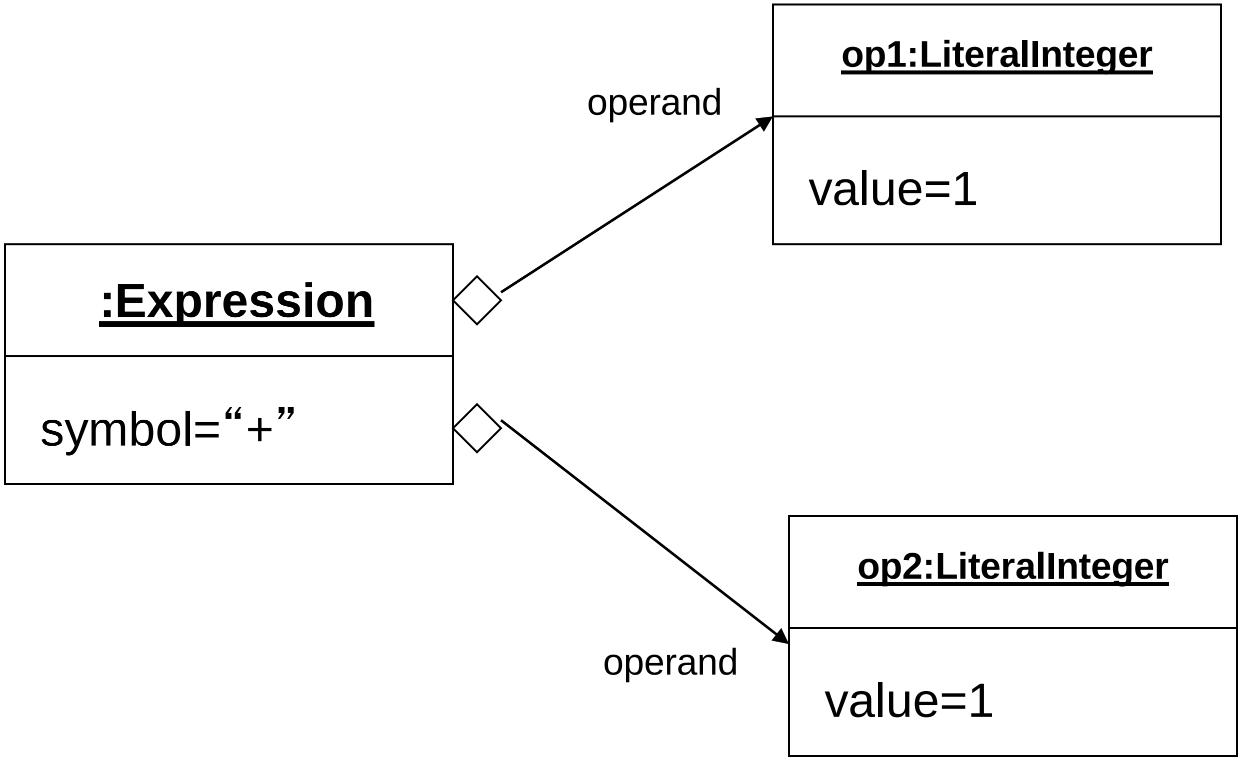
- In addition, there are language-dependent expressions defined by a language statement and the pertaining expression in that language (***opaque expression***), such OCL or Java expression (the language statement can be omitted if the language is implicity defined by the expression or context).

The metamodel and the composite pattern

- The metamodel is based on the *composite pattern*:

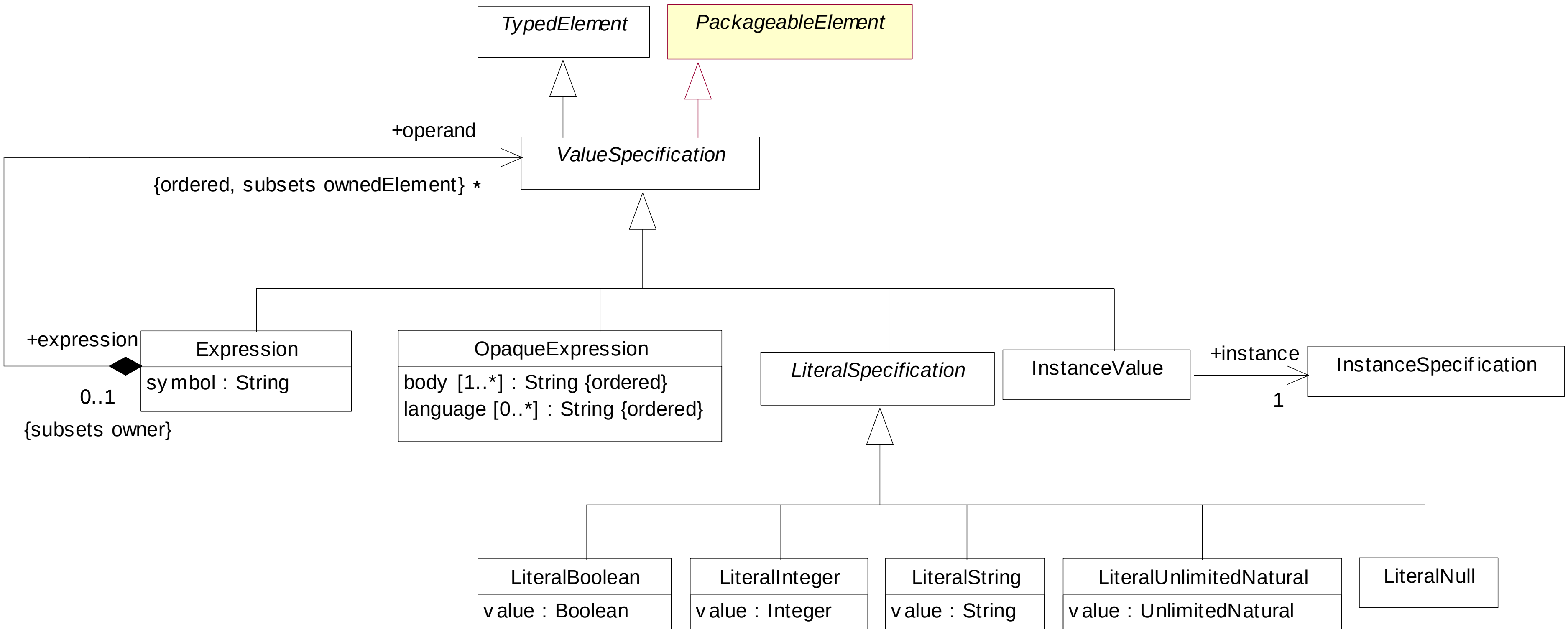


Example

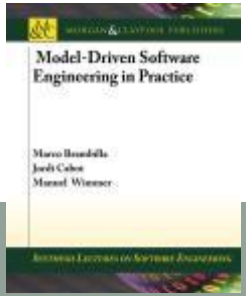


Object Model for 1+1

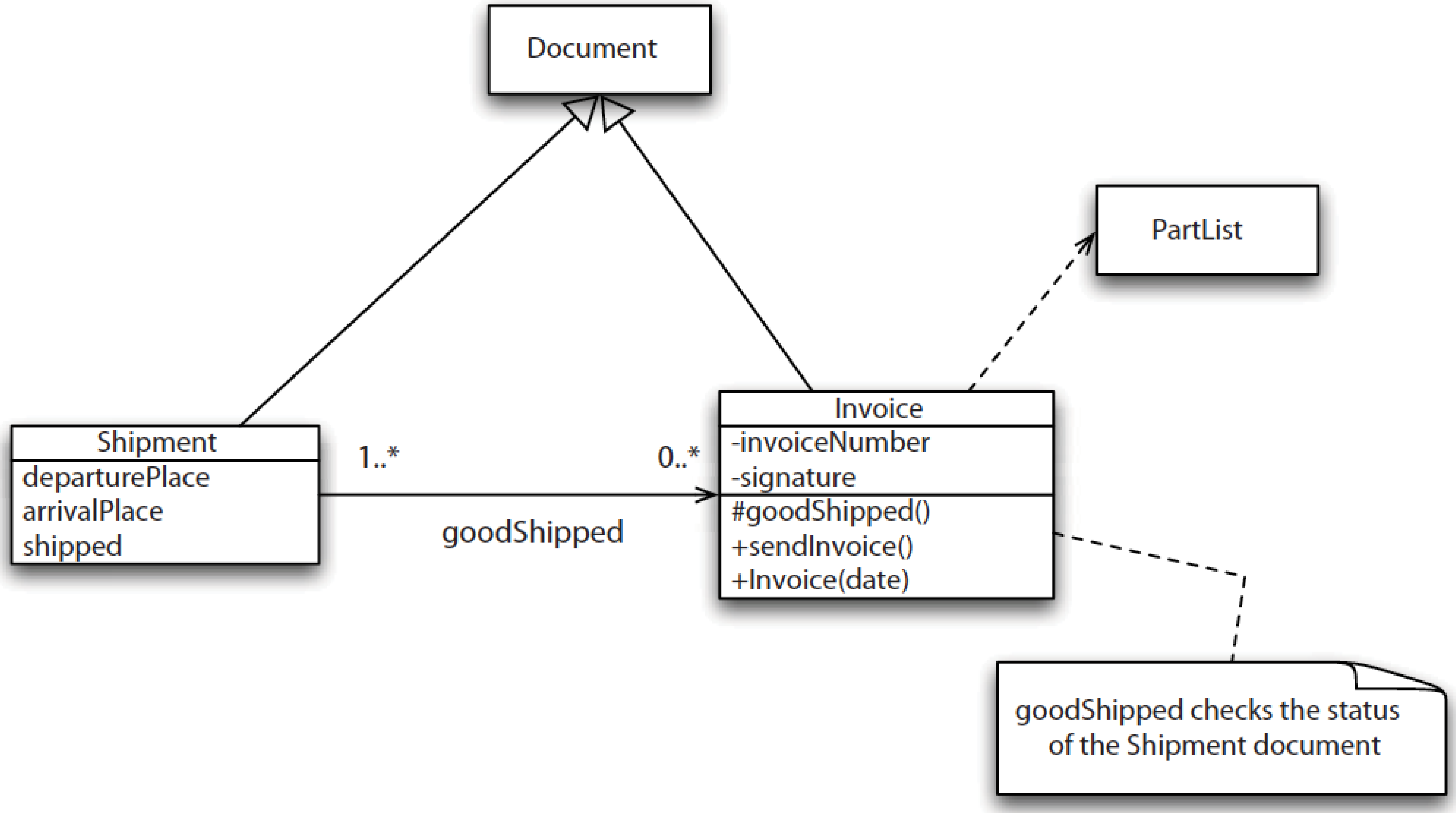
The metamodel for value specifications



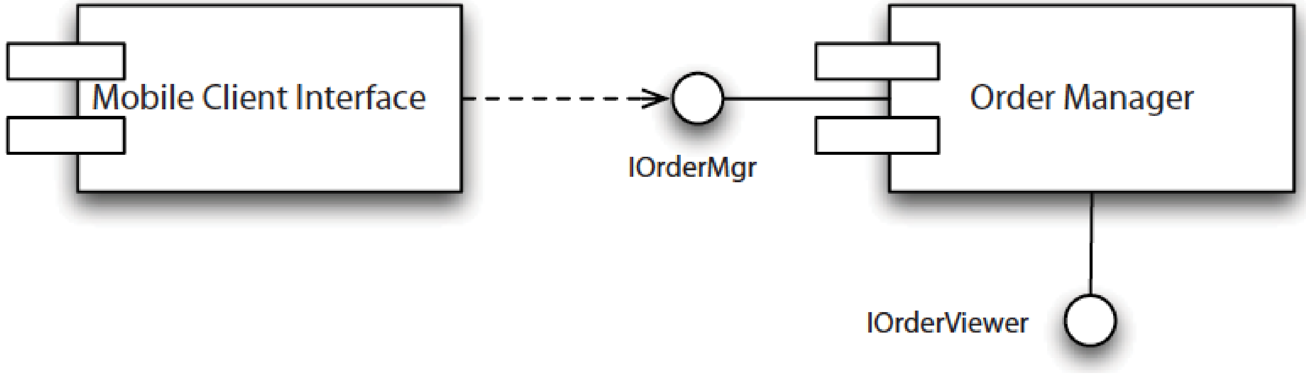
UML EXAMPLES



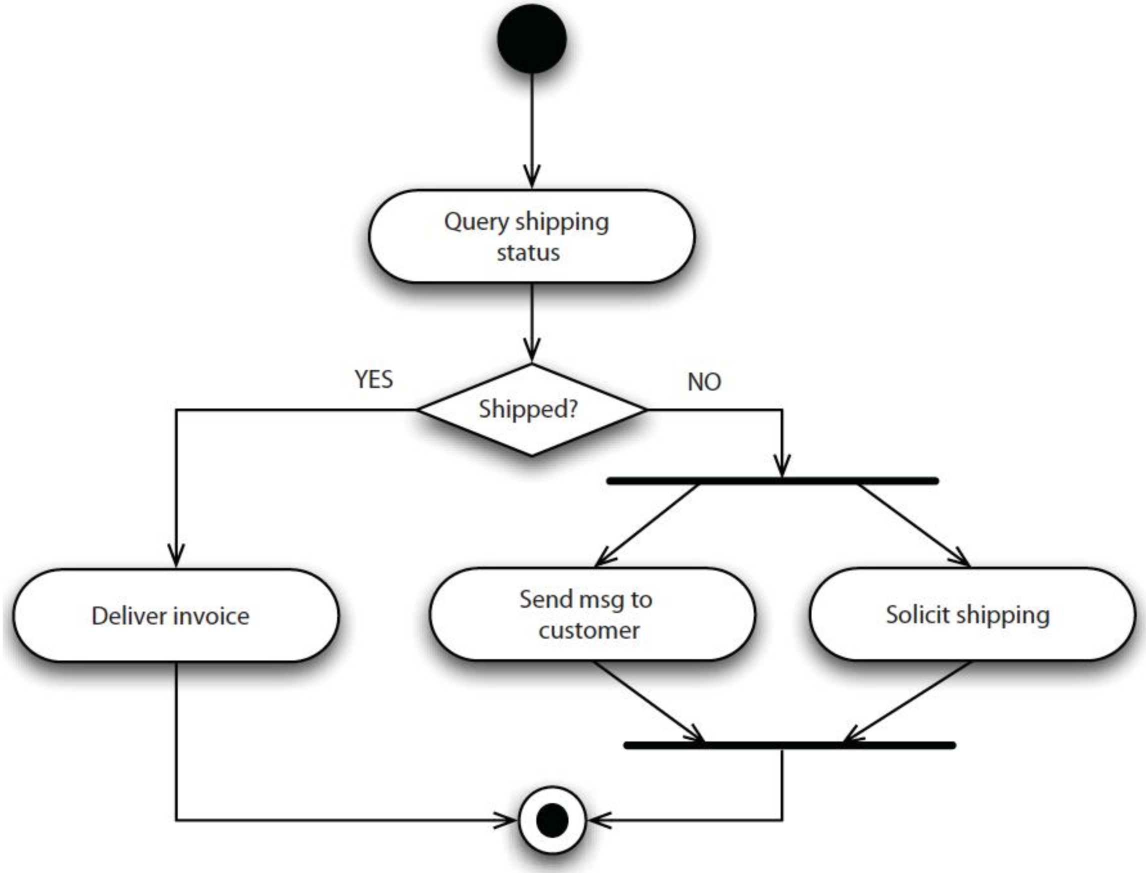
Class diagram



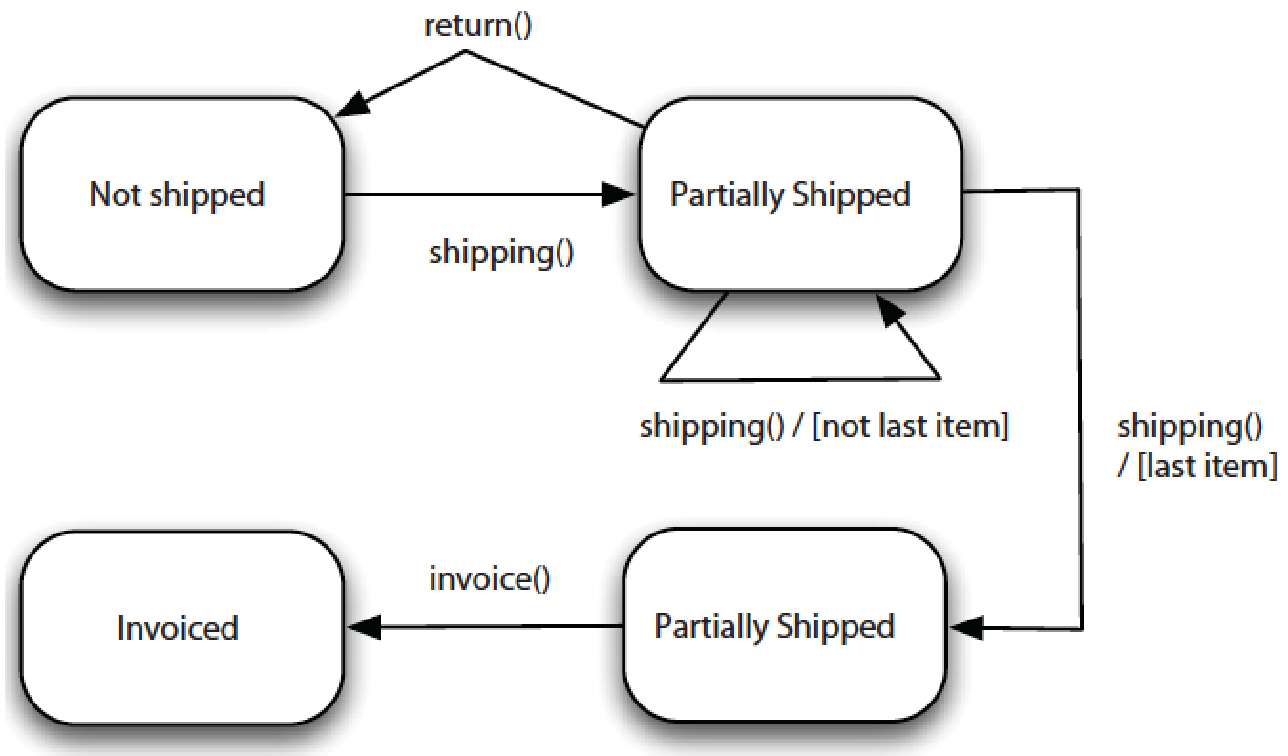
Component Diagram



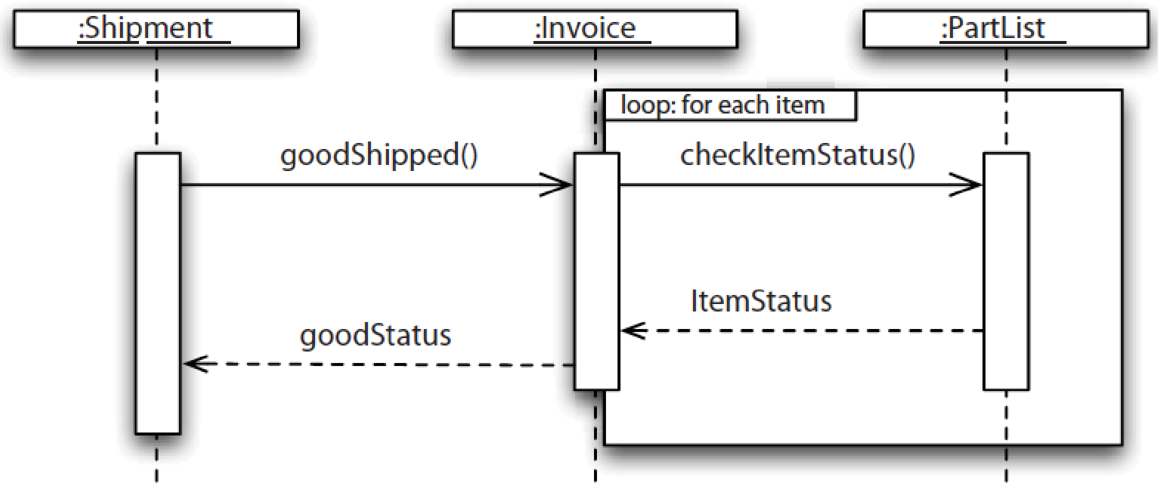
Activity diagram

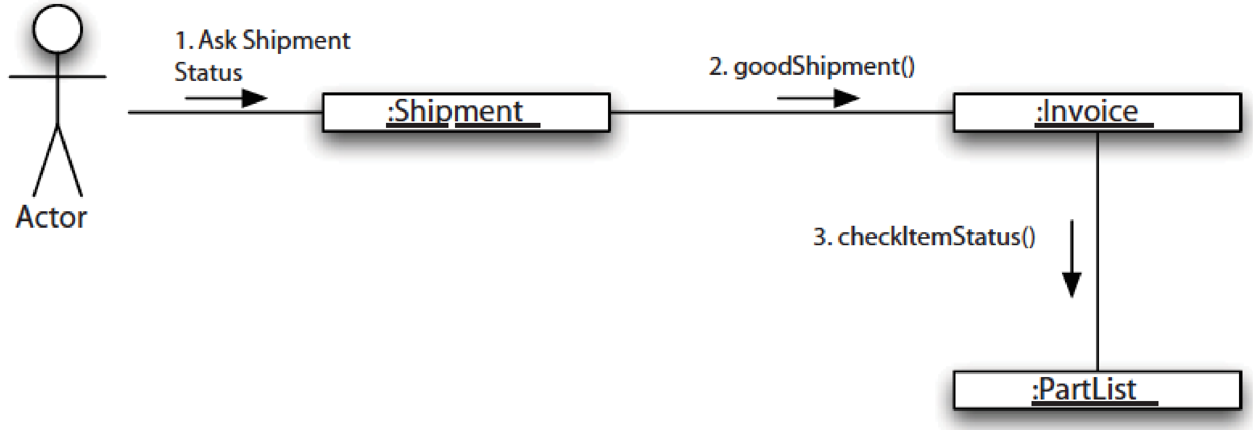


State Diagram

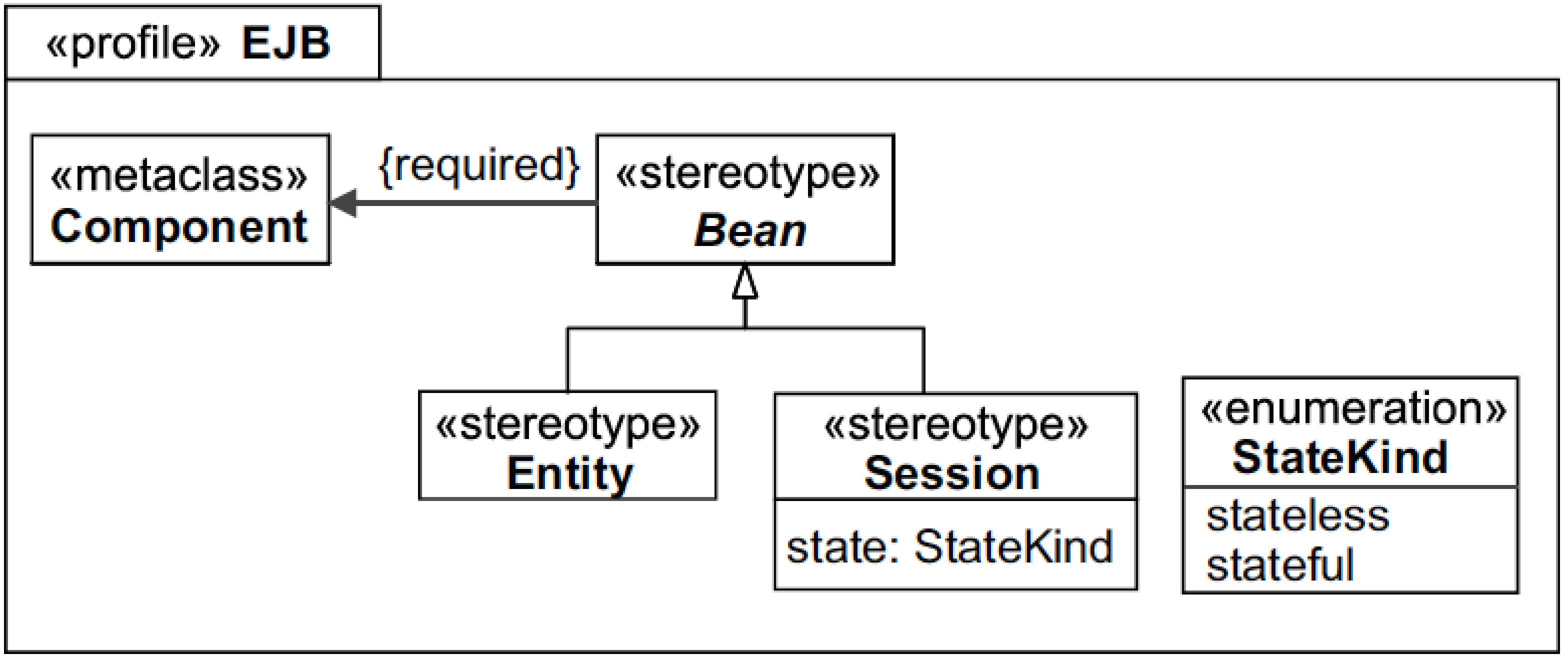


Sequence vs. Collaboration





UML Extensibility: profiles



APPROACHES TO MDA
MDA VS UML

Approaches

MDA with UML

- Problems when using **UML** as PIM/PSM
  - Method bodies?
  - Incomplete diagrams, e.g. missing attributes
  - Inconsistent diagrams
  - *For the usage of the UML in Model Engineering special guidelines have to be defined and adhered to*
- Different requirements to **code generation**
  - get/set methods
  - Serialization or persistence of an object
  - Security features, e.g. Java Security Policy
  - *Using adaptable code generators or PIM-to-PSM transformations*
- **Expressiveness** of the UML
  - UML is mainly suitable for “generic” software platforms like Java, EJB, .NET
  - Lack of support for user interfaces, code, etc.
  - *MDA tools often use proprietary extensions*

Approaches

MDA

- Many **UML tools** are expanded to MDA tools
  - UML profiles and code generators
  - Stage of development partly still similar to CASE: proprietary UML profiles and transformations, limited adaptability
- **Advantages** of MDA
  - Standardization of the Meta-Level
  - Separation of platform independent and platform specific models (reuse)
- **Disadvantages** of MDA
  - No special support for the development of the execution platform and the modeling language
  - Modeling language practically limited to UML with profiles
  - Therefore limited code generation (typically no method bodies, user interface)

Approaches

AC-MDSD

- Efficient reuse of architectures
  - Special attention to the efficient reuse of infrastructures/frameworks (= architectures) for a series of applications
  - Specific procedure model
    - Development of a reference application
    - Analysis in individual code, schematically recurring code and generic code (equal for all applications)
    - Extraction of the required modeling concepts and definition of the modeling language, transformations and platform
  - Software support (www.openarchitectureware.org)
- Basic architecture almost completely covered
  - When using UML profiles there is the problem of the method bodies
  - The recommended procedure is to rework these method bodies not in the model but in the generated code
- Advantages compared to MDA
  - Support for platform- and modeling language development
- Disadvantages compared to MDA
  - Platform independence and/or portability not considered

Approaches

MetaCASE/MetaEdit+

- Free configurable CASE
  - Meta modeling for the development of domain-specific modeling languages (**DSL**s)
  - **The focus** is on the ideal support of the **application area**, e.g. mobile-phone application, traffic light pre-emption, digital clock – Intentional Programming
  - Procedural method driven by the DSL development
- Support in particular for the **modeling level**
  - Strong Support for meta modeling, e.g. graphical editors
  - Platform development not assisted specifically, the usage of components and frameworks is recommended
- **Advantages**
  - Domain-specific languages
- **Disadvantages**
  - Tool support only focuses on graphical modeling

[www.metacase.com]

Approaches

Software Factories

- **Series production** of software products
  - Combines the ideas of different approaches (MDA, AC-MDSD, MetaCASE/DSLs) as well as popular SWD-technologies (patterns, components, frameworks)
  - Objective is the automatically processed development of software product series, i.e., a series of applications with the same application area and the same infrastructure
  - The SW-Factory as a marketable product
- Support of the **complete basic architecture**
  - Refinements in particular on the realization level, e.g. deployment
- **Advantages**
  - Comprehensive approach
- **Disadvantages**
  - Approach not clearly delimited (similar MDA)
  - Only little tool support

[J. Greenfield, K. Short: Software Factories. Wiley, 2004]

Approaches

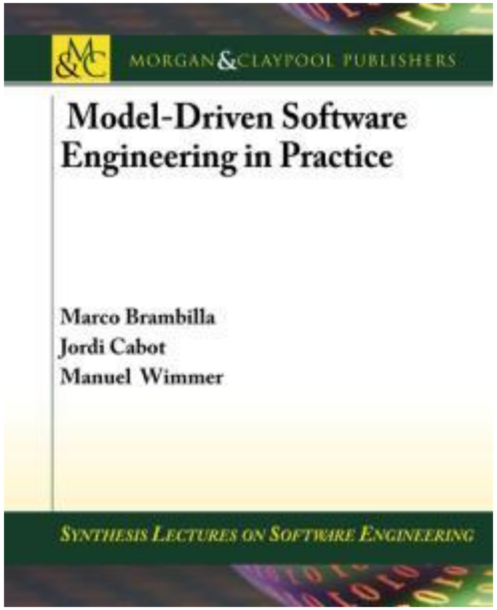
Excursus: OMG Standards

- CORBA - Common Object Request Broker Architecture
  - Language- and platform-neutral interoperability standard (similar to WSDL, SOAP and UDDI)
- UML - Unified Modeling Language
  - Standardized modeling language, industry standard
- CWM - Common Warehouse Metamodel
  - Integrated modeling language for data warehouses
- MOF – Meta Object Facility
  - A standard for metamodels and model repositories
- XMI - XML Metadata Interchange
  - XML-based exchange of models
- QVT – Queries/Views/Transformations
  - Standard language for model-to-model transformations

MORGAN & CLAYPOOL PUBLISHERS

MODEL-DRIVEN SOFTWARE ENGINEERING IN PRACTICE

Marco Brambilla,
Jordi Cabot,
Manuel Wimmer.
Morgan & Claypool, USA, 2012.



www.mdse-book.com
www.morganclaypool.com
or buy it at: www.amazon.com