CHAPTER 4 -
Modeling Languages at a Glance

Overview

1 Introduction

1.1 Human Cognitive Processes
1.2 Models
1.3 Model Engineering

2 MDSE Principles

2.1 MDSE Basics
2.2 The MD* Jungle of Acronyms
2.3 Modeling Languages and Metamodeling
2.4 Overview Considered Approaches
2.5 Tool Support
2.6 Criticisms of MDSE

3 MDSE Use Cases

3.1 MDSE applications
3.2 USE CASE1 – Model driven development
3.3 USE CASE2 – Systems interoperability
3.4 USE CASE3 – Model driven reverse engineering

Overview

4 Modeling Languages at a Glance

5.1 Anatomy of Modeling Languages

5.2 General Purpose vs. Domain-Specific Modeling Languages

5.3 Overview of UML Diagrams

5.4 UML Behavioural or Dynamic Diagrams

5.5 UML Extensibility: The Middle Way Between GPL and DSL

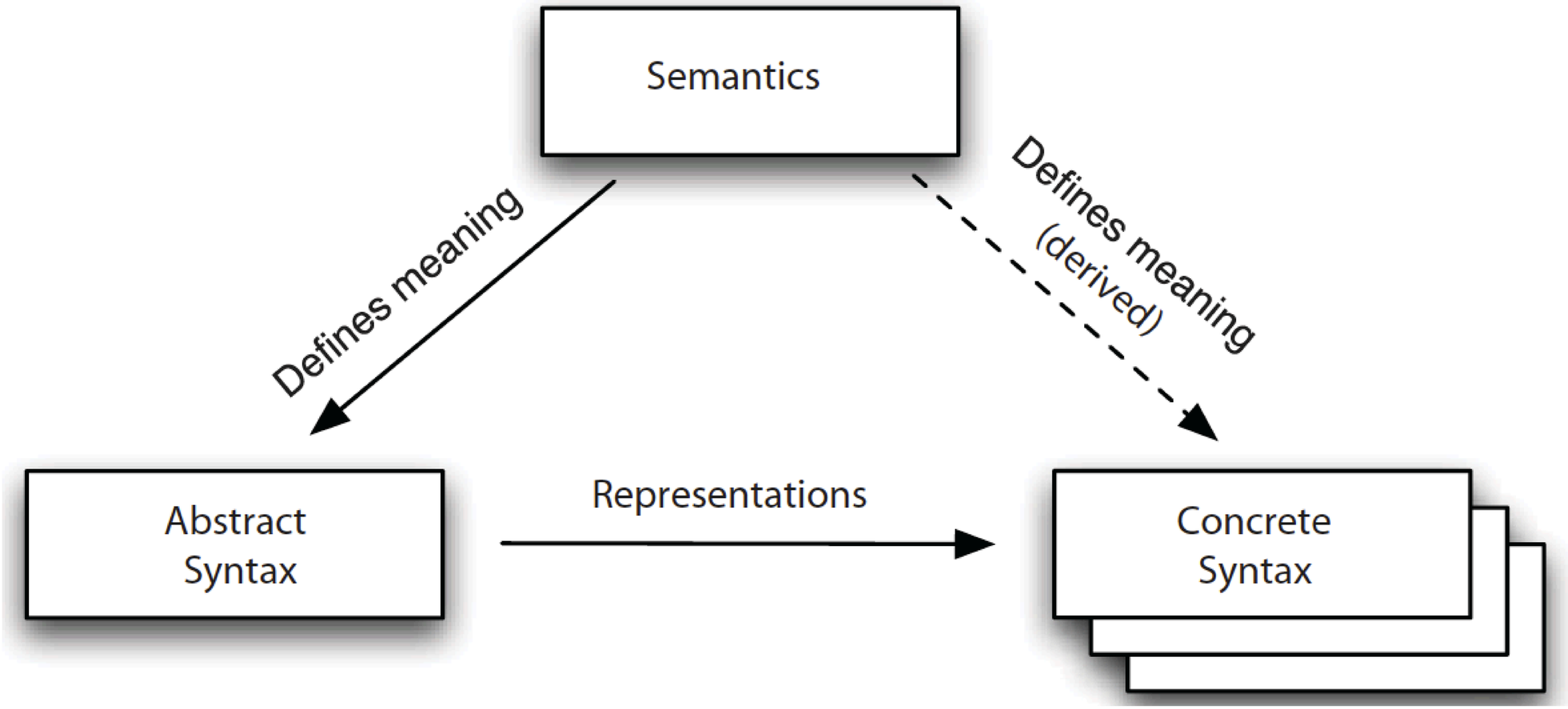
5.6 Domain Specific Languages

5.7 Defining Modeling Constraints (OCL)

Anatomy of a Modeling Language

- Abstract syntax: Describes the structure of the language and the way the different primitives can be combined together, independently of any particular representation or encoding.
- Concrete syntax: Describes specific representations of the modeling language, covering encoding and/or visual appearance
- Semantics: Describing the meaning of the elements defined in the language and the meaning of the different ways of combining them.

Anatomy of a Modeling Language



DSL vs. GPL

- First distinction is between
  - » General Purpose languages (GPL or GPML) and
  - » Domain Specific languages (DSL or DSML)
  - » (already discussed in Chapter 2)

- We take UML as an exemplary case of GPL

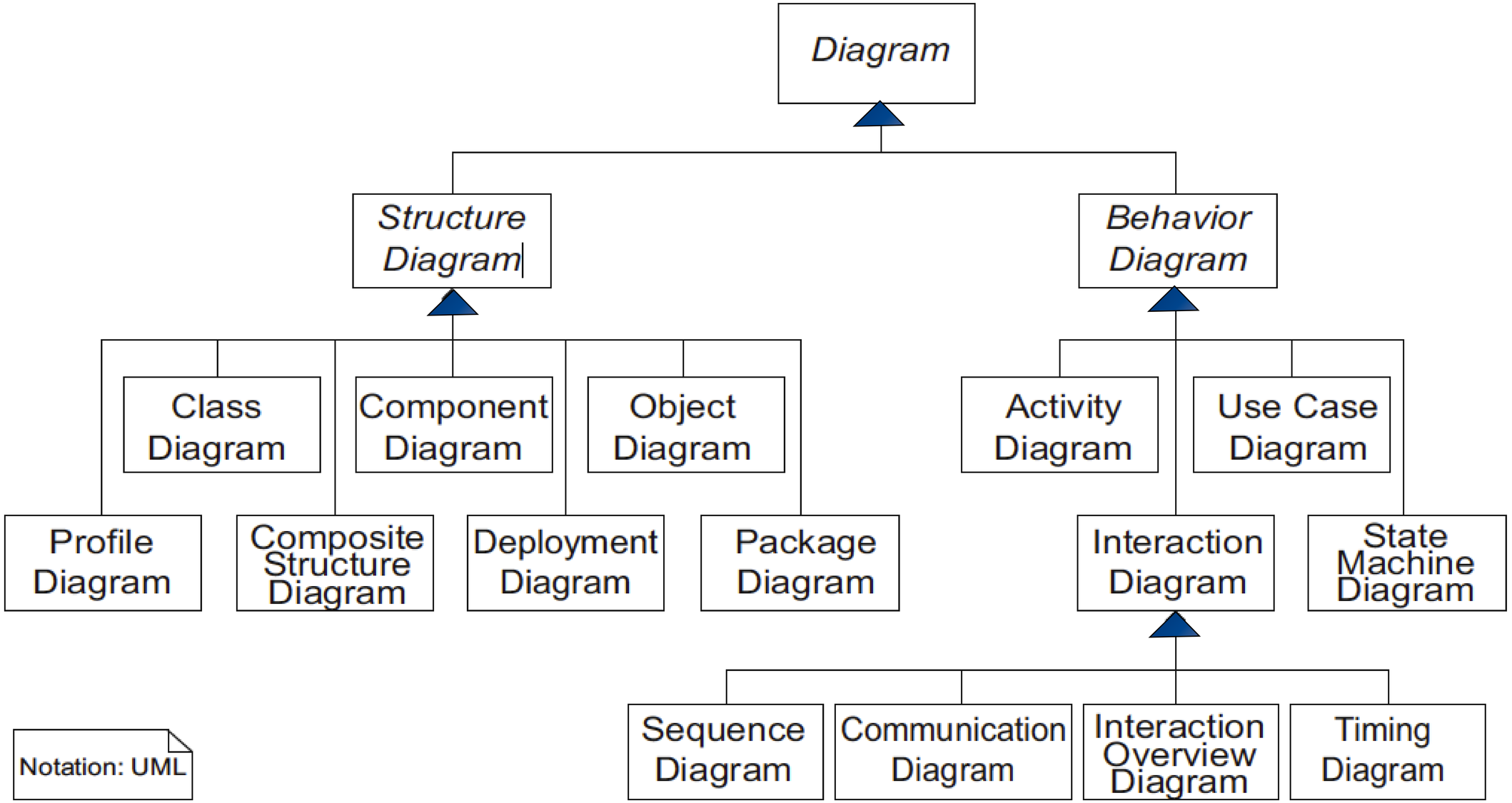
Overview of UML Diagrams

- 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 UML 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.

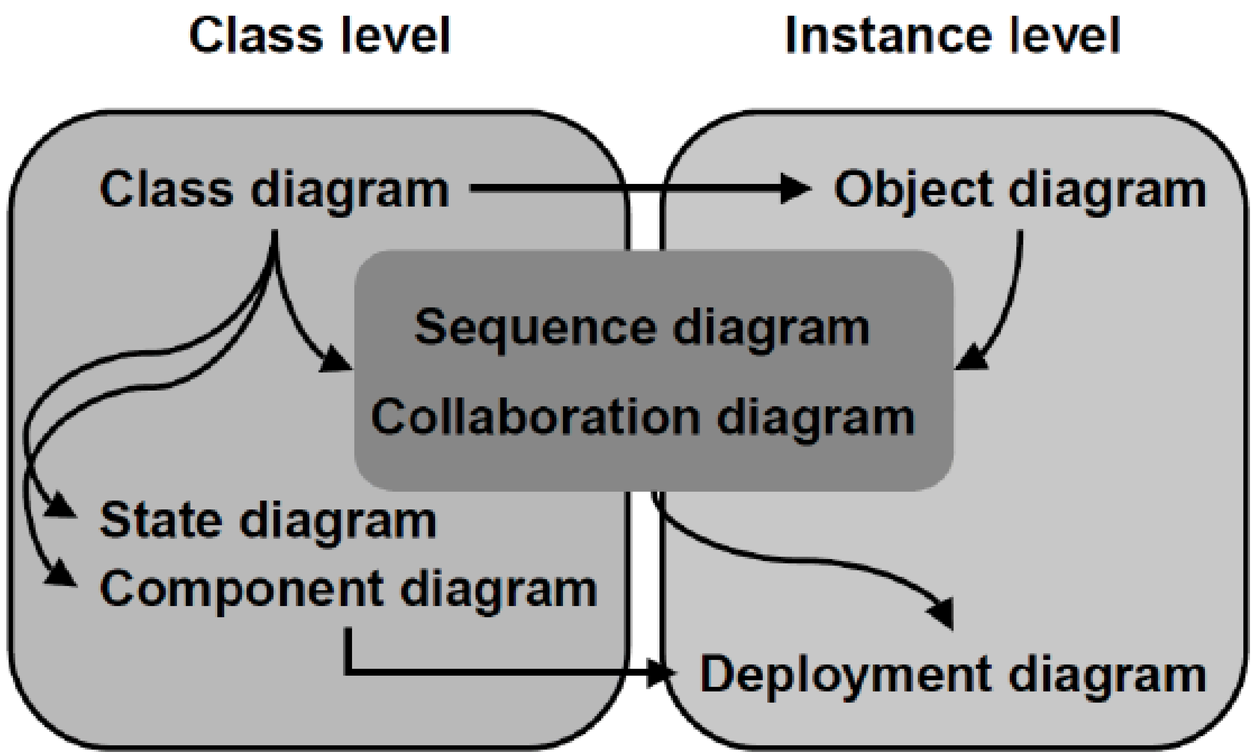
Overview of the UML diagrams



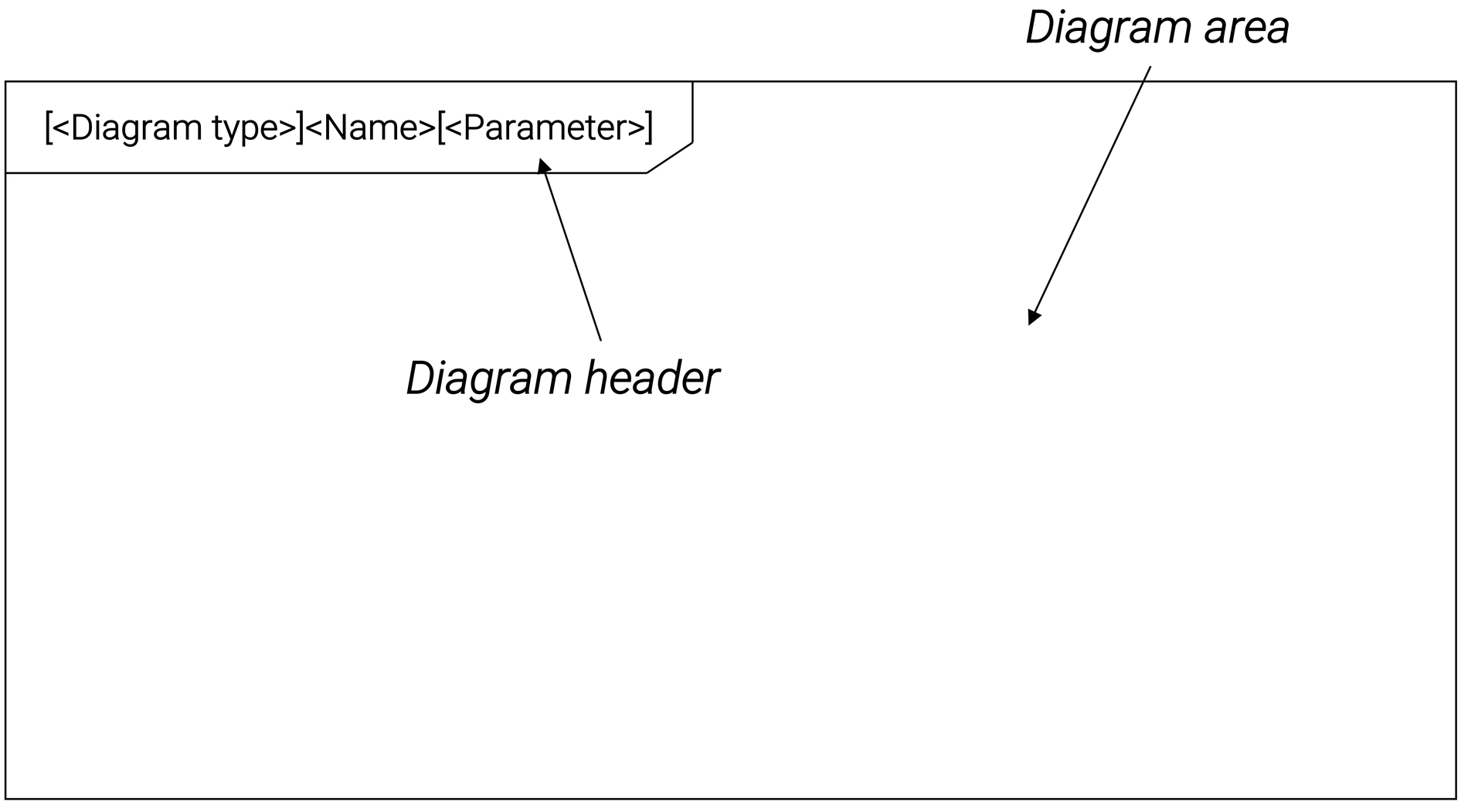
UML Design practices

- Pattern-based design: A set of very well-known design patterns, defined by the so-called Gang of Four
- Using several integrated and orthogonal models together: UML comprises a suite of diagrams that share some symbols and allow cross-referencing
- Modeling at different levels of detail: UML allows eliding details in diagrams when needed. Choose the right quantity of information to include in diagrams
- Extensibility: UML provides a good set of extensibility features which allow to design customized modeling languages if needed

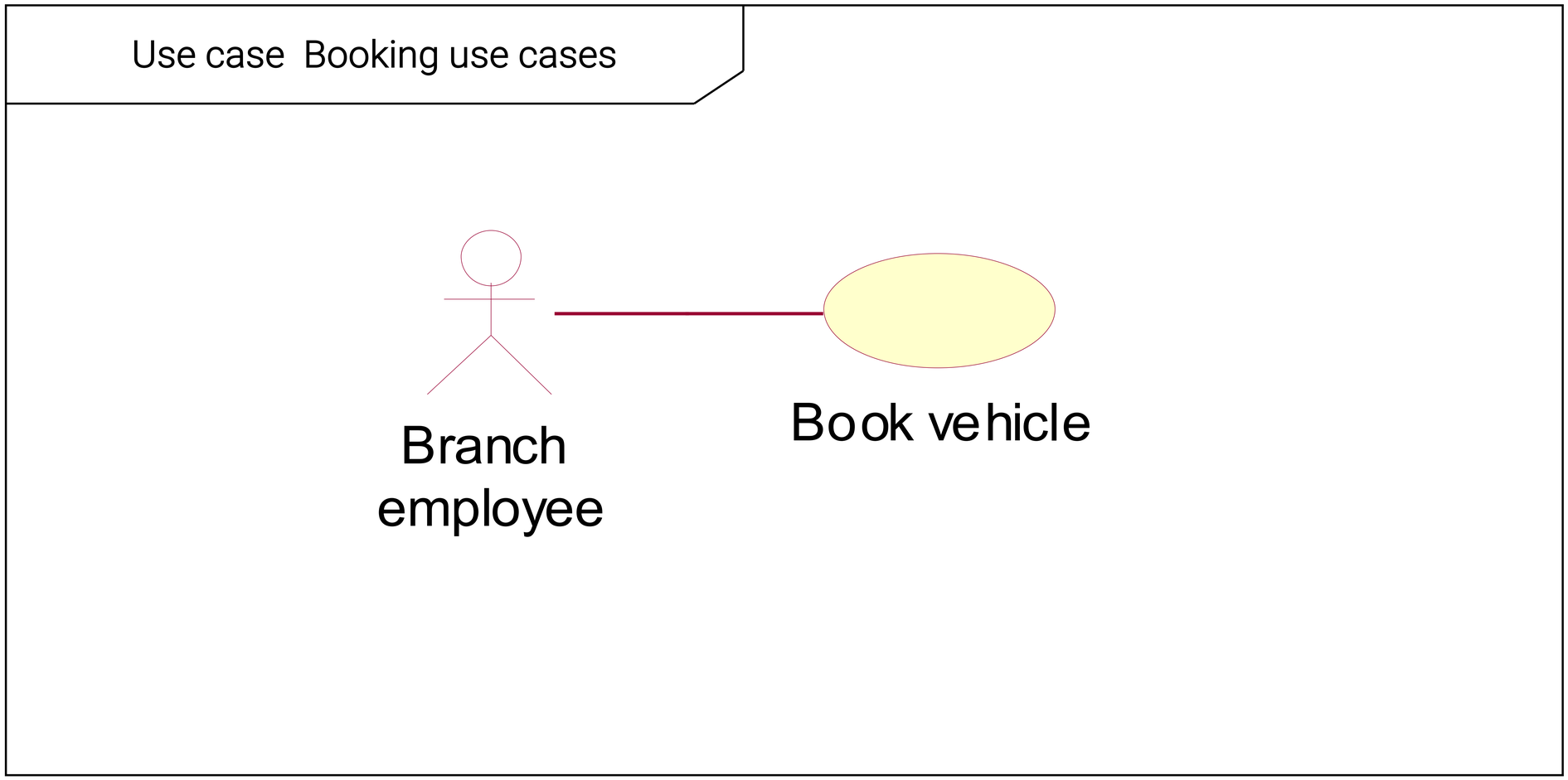
Class vs. instance in diagrams



Basic notation for diagrams



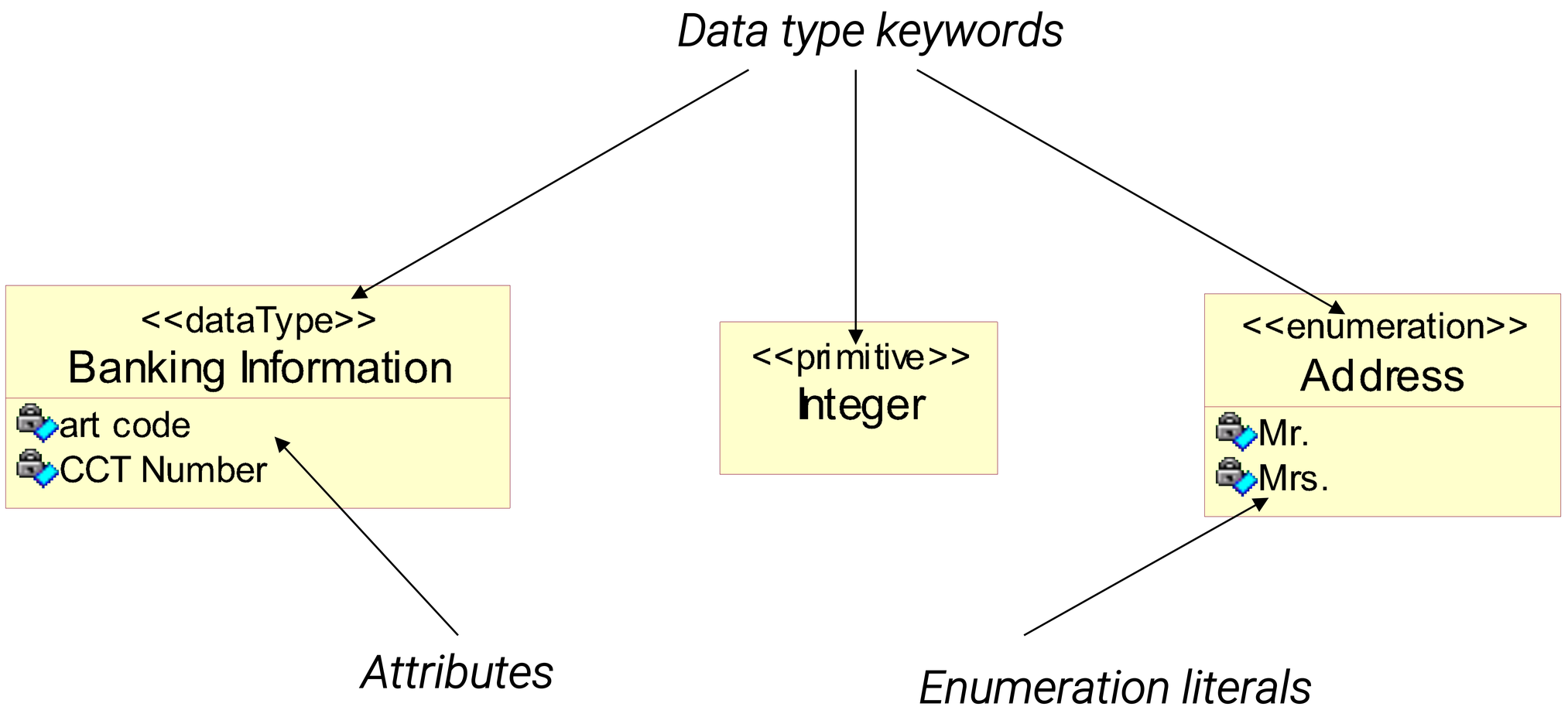
Example of a use case diagram



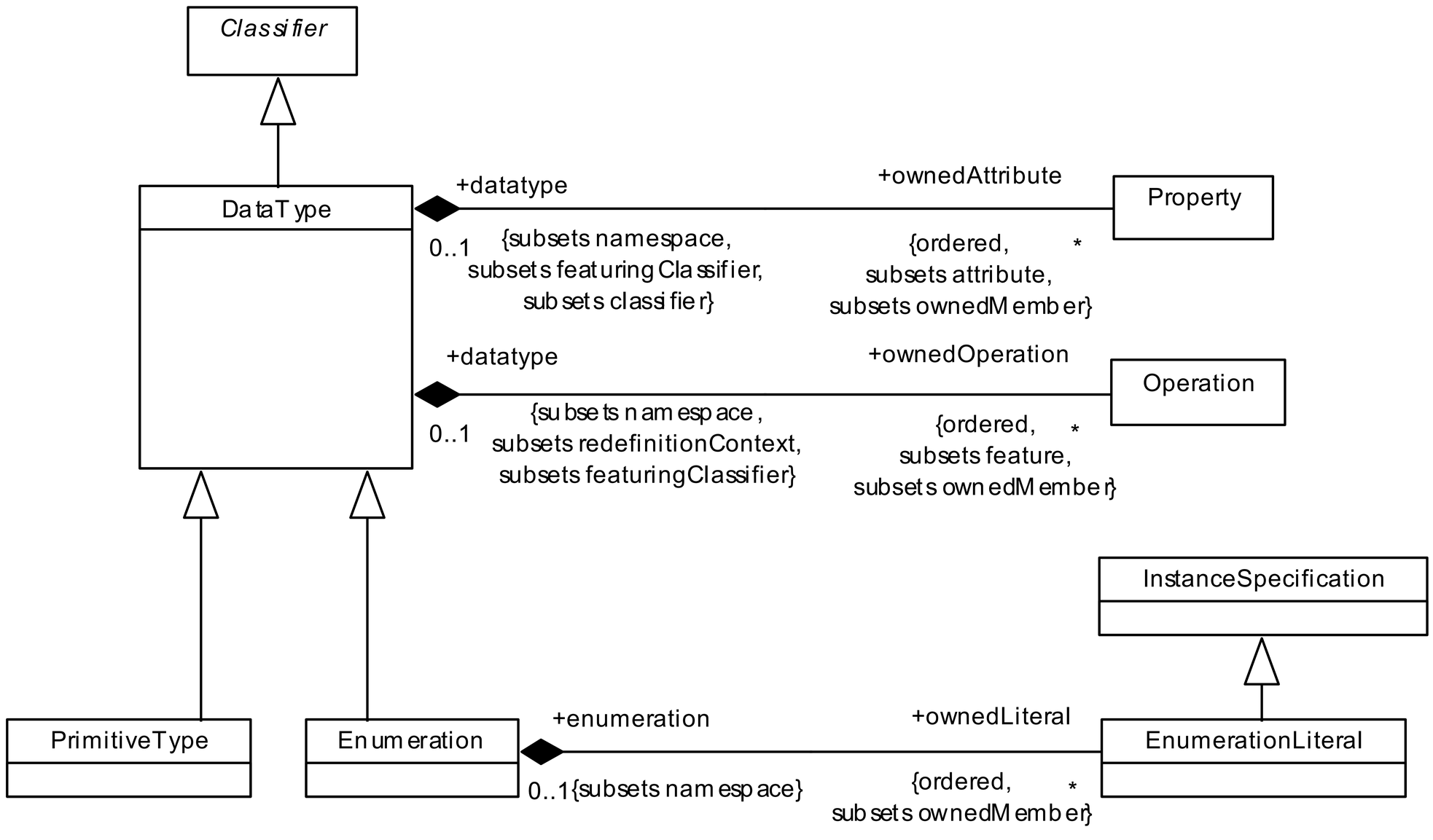
Datatypes

- 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



Structure diagrams (1)

- Emphasize the static description of the elements that must be present in the system being modeled
- 1. The conceptual items of interest for the system
  - » Class diagram: Describes the structure of a system by showing the classes of the systems, their attributes, and the relationships among the classes
  - » Composite structure diagram: Describes the internal structure of a class and the collaborations
  - » Object diagram: A view of the structure of example instances of modeled concepts

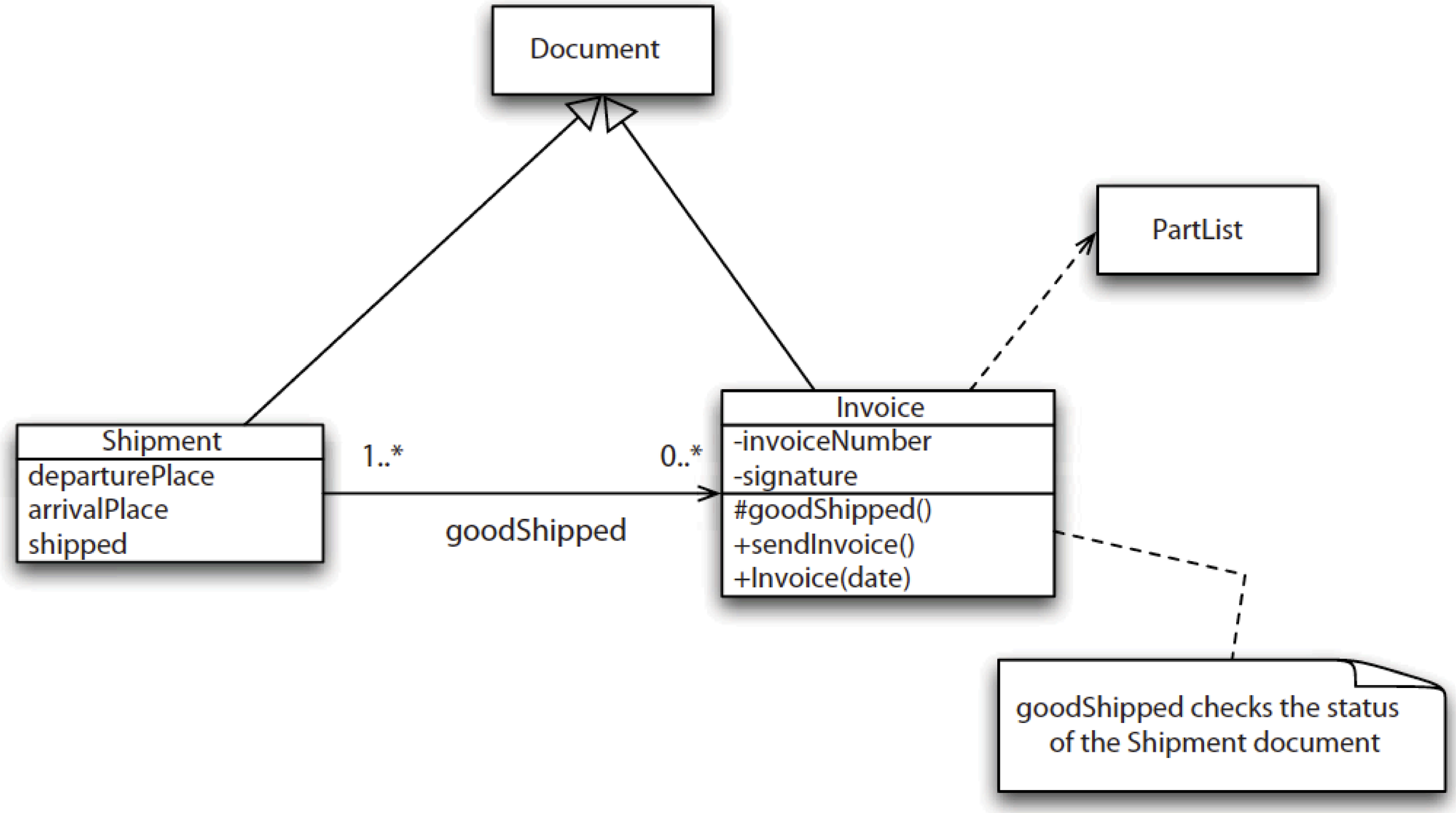
Structure diagrams (2)

- 2. The architectural organization and structure of the system.
  - » Package diagram: Describes how a system is split up into logical groupings
  - » Component diagram: Describes how a software system is split up into components
  - » Object diagram: A view of the structure of example instances of modeled concepts

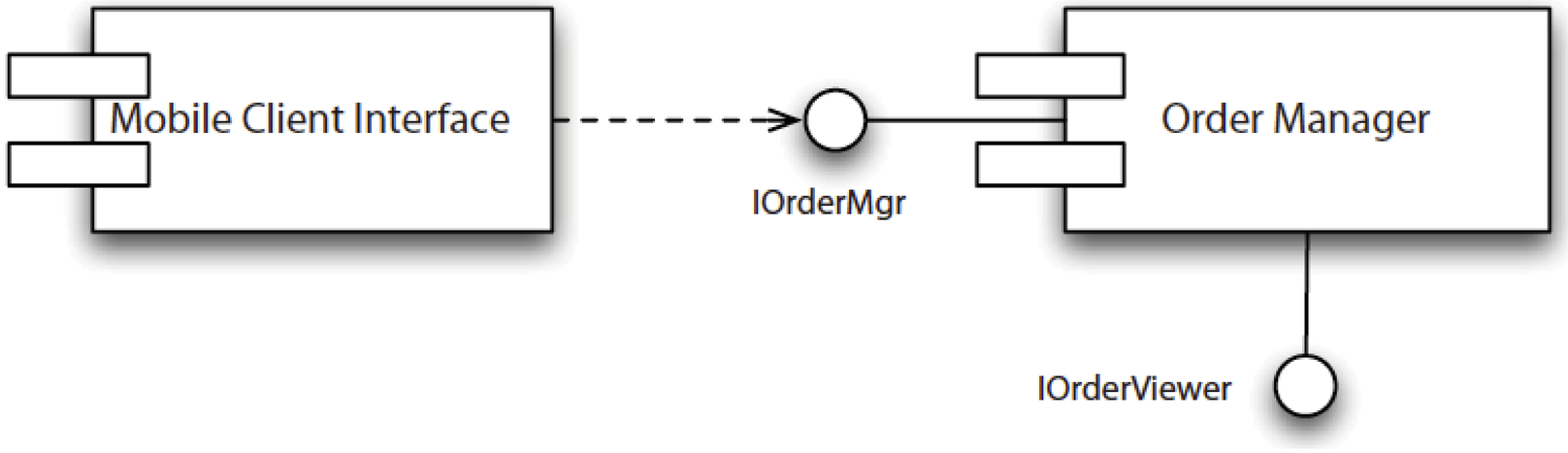
Class diagrams basic concepts

- The basis of UML is described in the Kernel package of the UML 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.

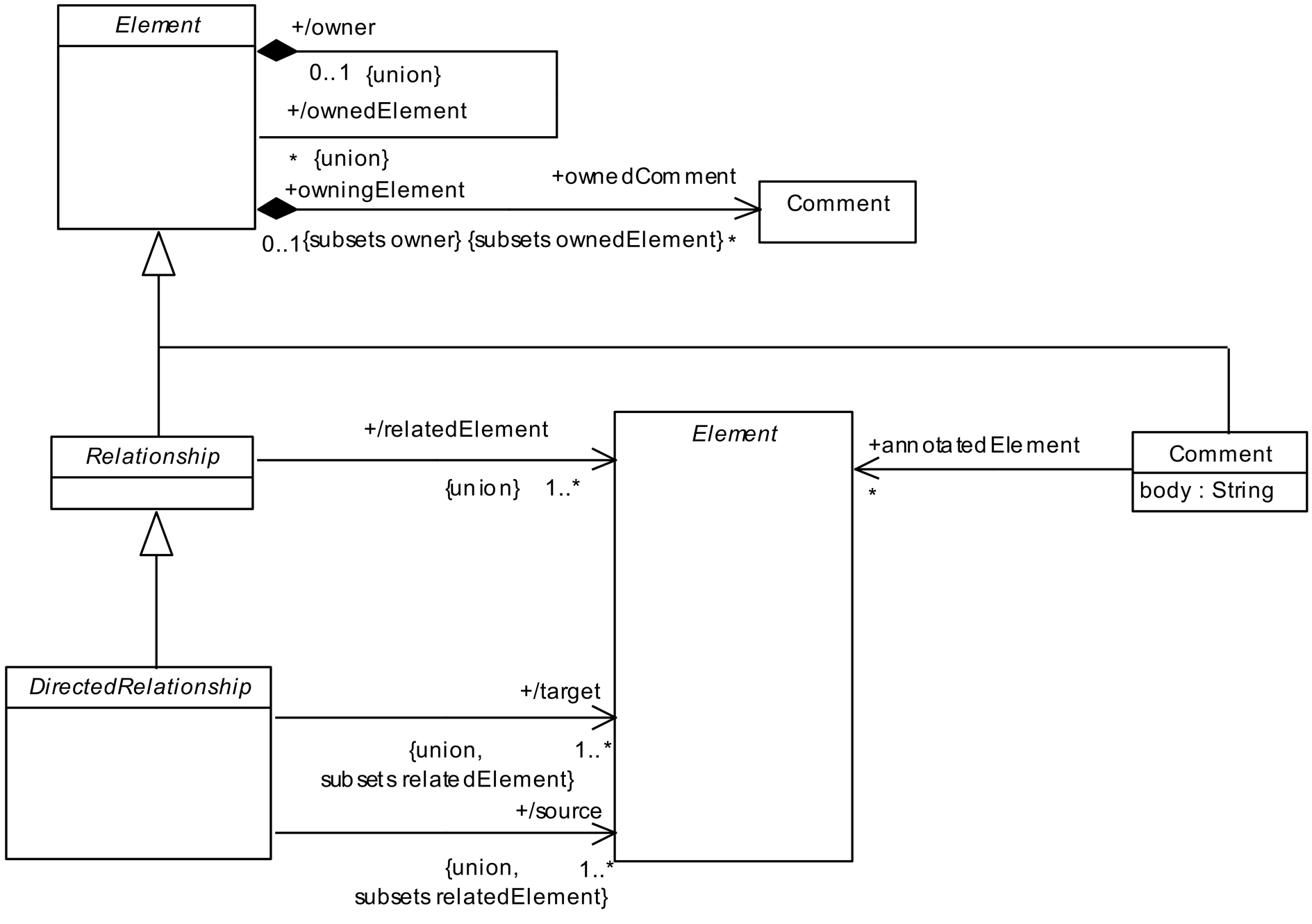
Class diagram example



Component Diagram example



The basic concepts in the UML metamodel for class diagrams



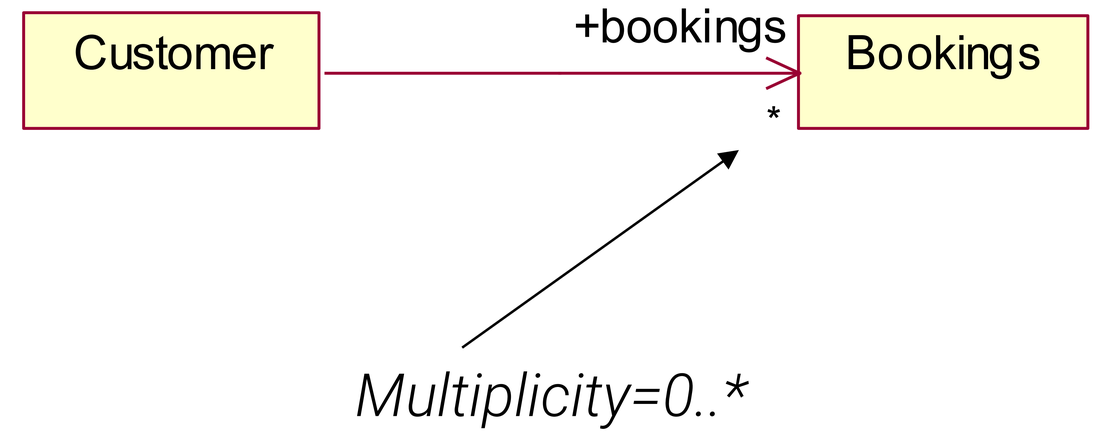
Named elements

- 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

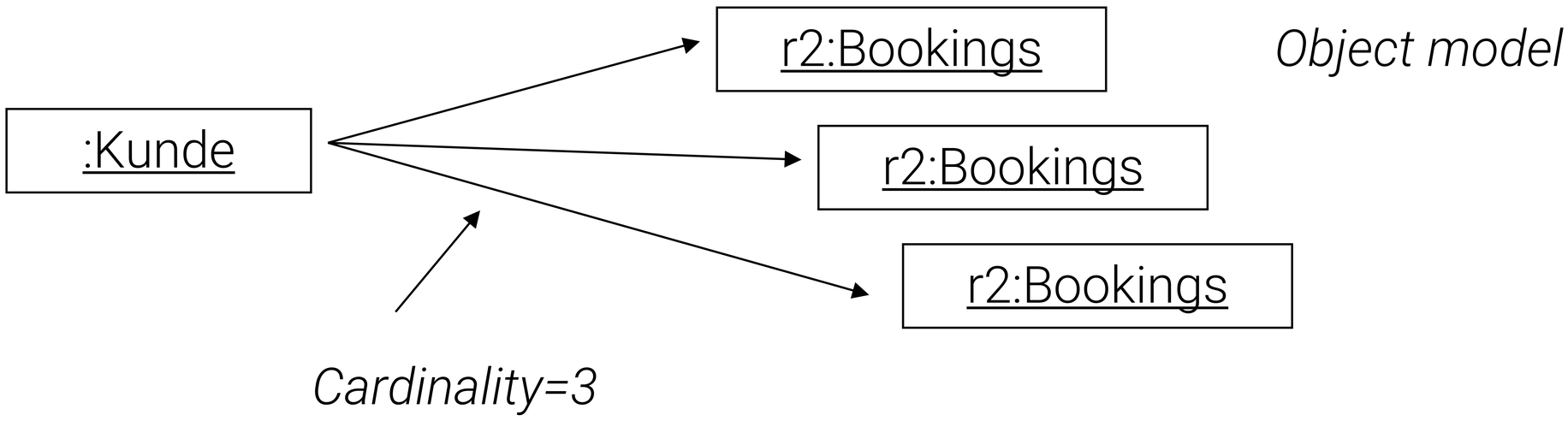
Multiplicities

- 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



Class model



Object model

UML BEHAVIOURAL OR DYNAMIC DIAGRAMS

Software Methodologies
for Distributed Systems

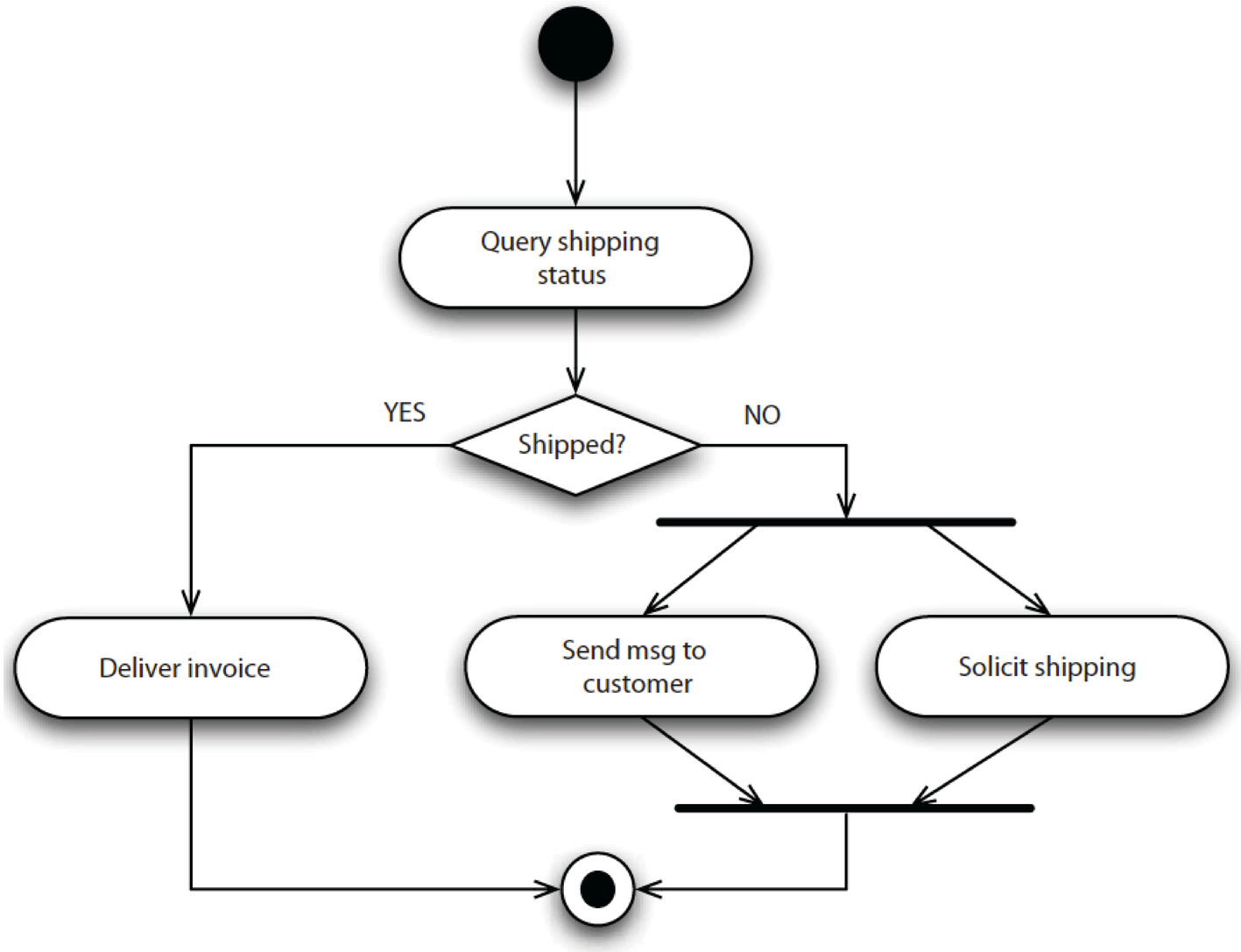
UML Behavioural Diagrams (1)

- Use case diagram: Describes the functionality provided by a system in terms of actors external to the system and their goals in using the system
- Activity diagram: Describes the step-by-step workflows of activities to be performed in a system for reaching a specific goal
- State machine diagram (or statechart): Describes the states and state transitions of the system, of a subsystem, or of one specific object.

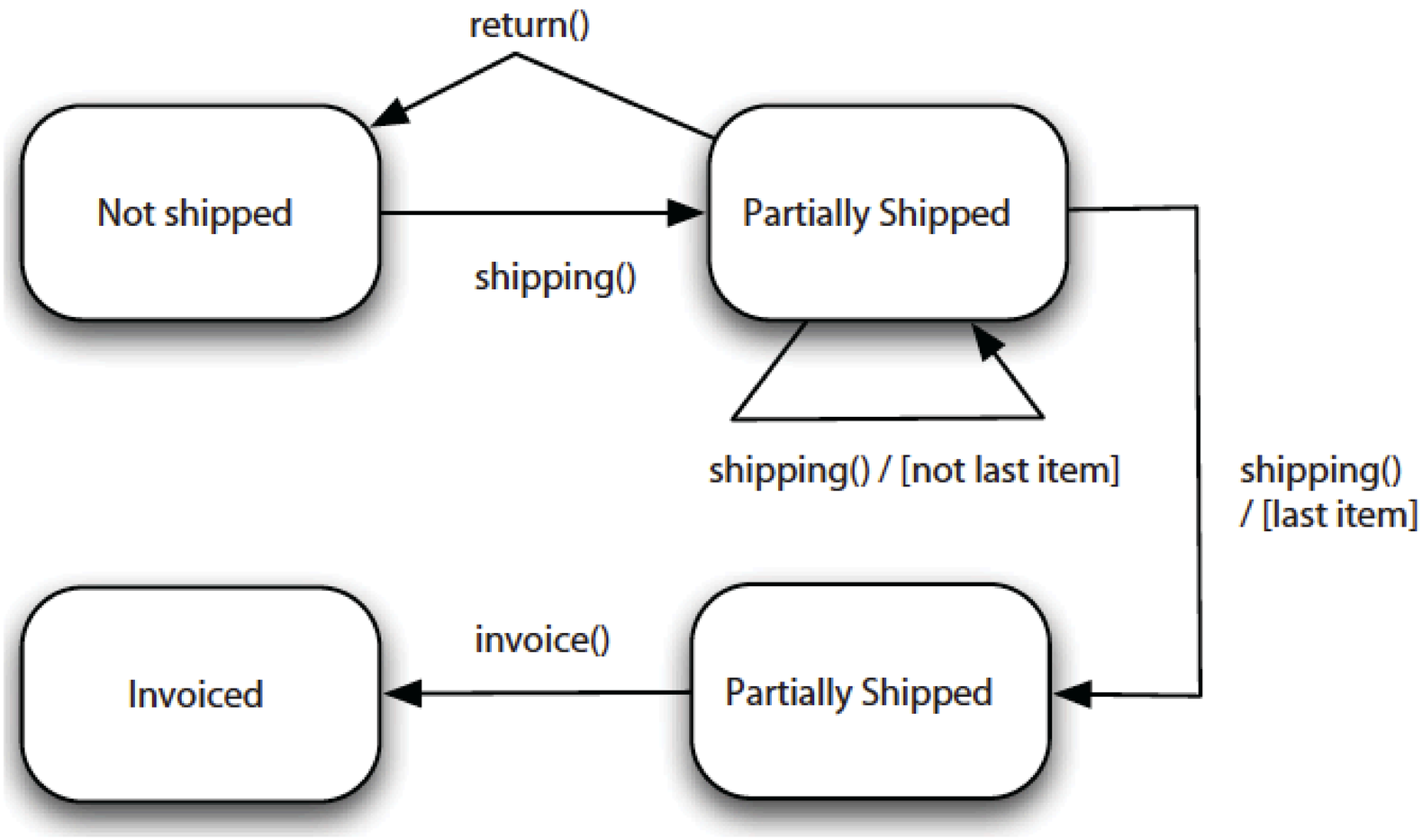
UML Behavioural Diagrams (2): Interaction diagrams

- A subset of behavior diagrams, emphasize the flow of control and data among the elements of the system.
- Sequence diagram: Shows how objects communicate with each other in terms of a temporal sequence of messages
- Communication or collaboration diagram: Shows the interactions between objects or classes in terms of links and messages that flow through the links
- Interaction overview diagram: Provides an overview in which the nodes represent interaction diagrams
- Timing diagrams: A specific type of interaction diagram where the focus is on timing constraints

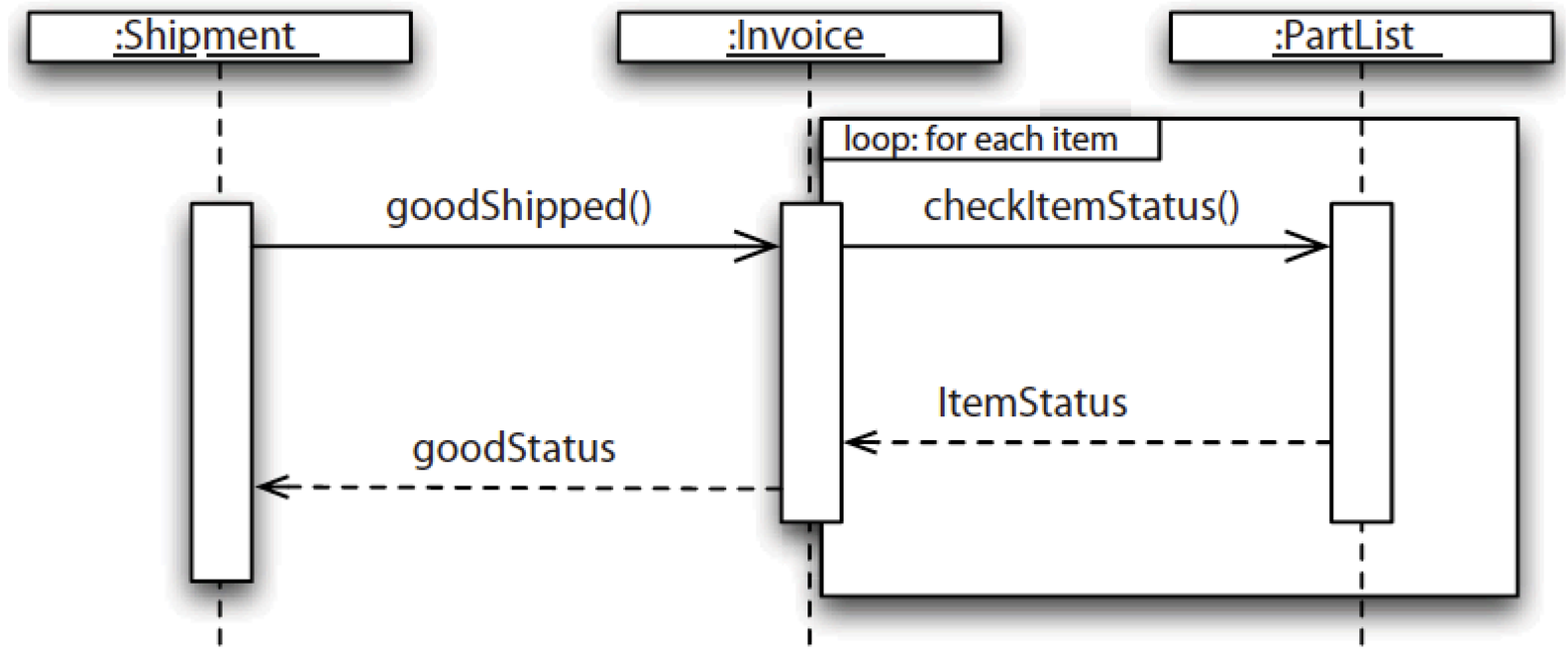
Activity diagram example



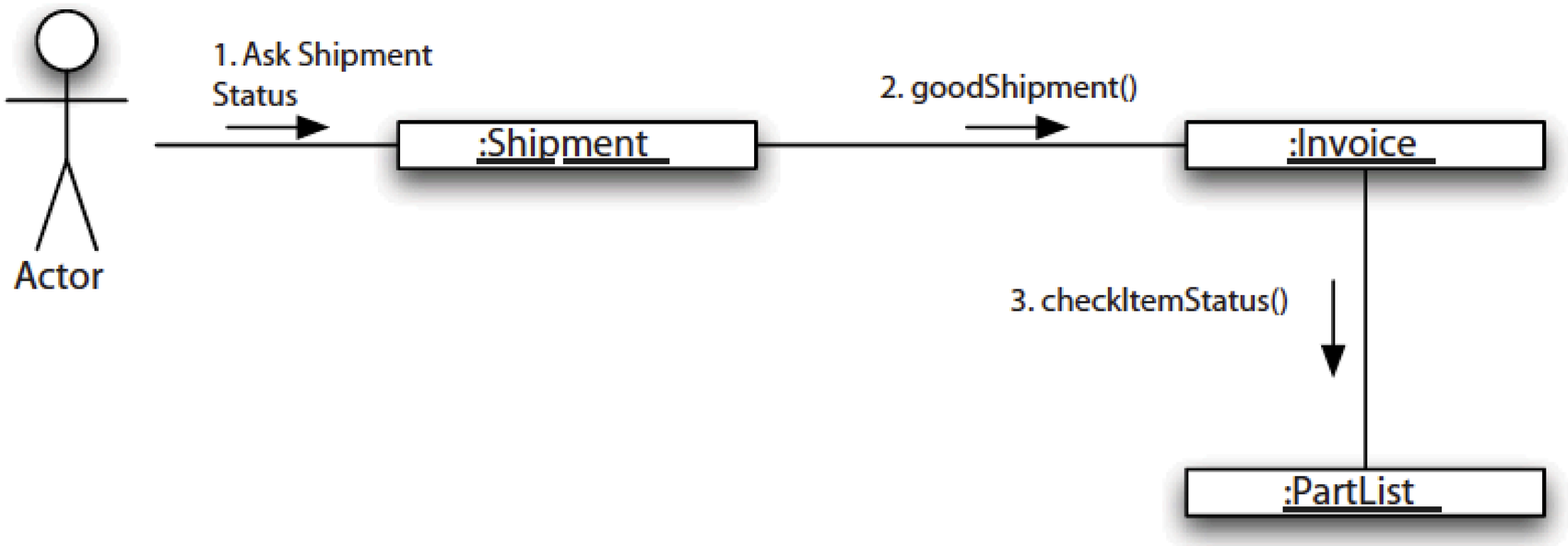
State Diagram example



Sequence diagram example



Collaboration diagram example



UML EXTENSIBILITY

Extensibility: Stereotype 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.

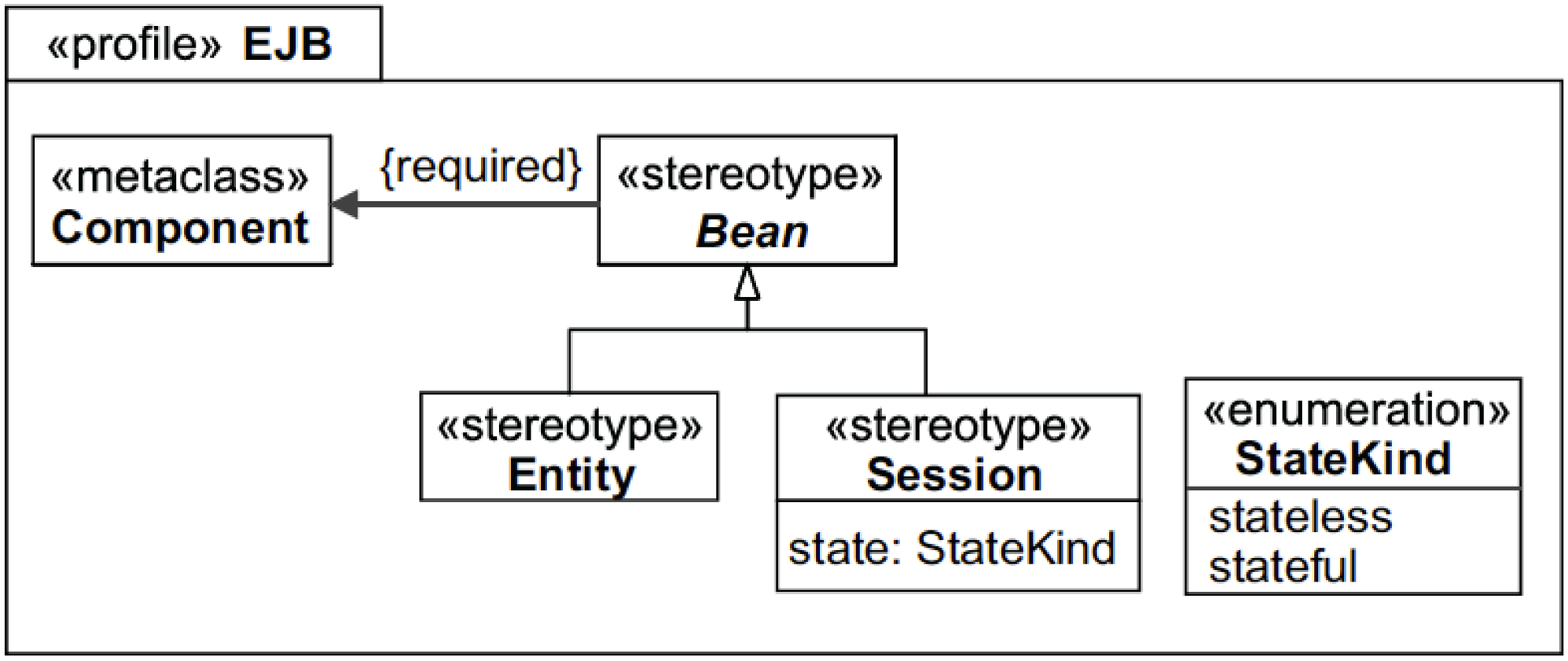
Multiple stereotyping

- 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.

UML Extensibility: profile example



DOMAIN-SPECIFIC LANGUAGES

Principles for Domain Specific Languages

- The language must provide good abstractions to the developer, must be intuitive, and make life easier, not harder
- The language must not depend on one-man expertise for its adoption and usage. Its definition must be shared and agreed upon
- The language must evolve and must be kept updated based on the user and context needs, otherwise it is doomed to die.
- The language must come together with supporting tools and methods
- The language should be open for extensions and closed for modifications (open-close principle)

Classification of DSLs (1): FOCUS. Horizontal vs. Vertical

- Vertical DSLs aim at a specific industry or field.
- Examples: configuration languages for home automation systems, modeling languages for biological experiments, analysis languages for financial applications.

- Horizontal DSLs have a broader applicability and their technical and cover concepts that apply across a large set of fields. They may refer to a specific technology but not to a specific industry.
- Examples: SQL, Flex, WebML.

Classification of DSLs (2): STYLE. Declarative vs. Imperative

- Declarative DSLs: specification paradigm that expresses the logic of a computation without describing its control flow.
- The language defines what the program should accomplish, rather than describing how to accomplish it.
- Examples Web service choreography, SQL.

- Imperative DSLs: define an executable algorithm that states the steps and control flow that needs to be followed.
- Examples: service orchestrations (start-to-end flows), BPMN process diagrams, programming languages like Java or C/C++.

Classification of DSLs (3): NOTATION. Graphical vs. Textual

- Graphical DSLs: the outcomes of the development are visual models and the development primitives are graphical items such as blocks, arrows and edges, containers, symbols, and so on.

- Textual DSLs comprise several categories, including XML-based notations, structured text notations, textual configuration files, and so on.

Classification of DSLs (4): INTERNALITY. Internal vs. External

- External DSLs have their own custom syntax, with a full parser and self-standing, independent models/programs.

- Internal DSLs consist in using a host language and give it the feel of a particular domain or objective, either by embedding pieces of the DSL in the host language or by providing abstractions, structures, or functions upon it

Classification of DSLs (5): EXECUTABILITY. Model Interpretation vs. Code Generation

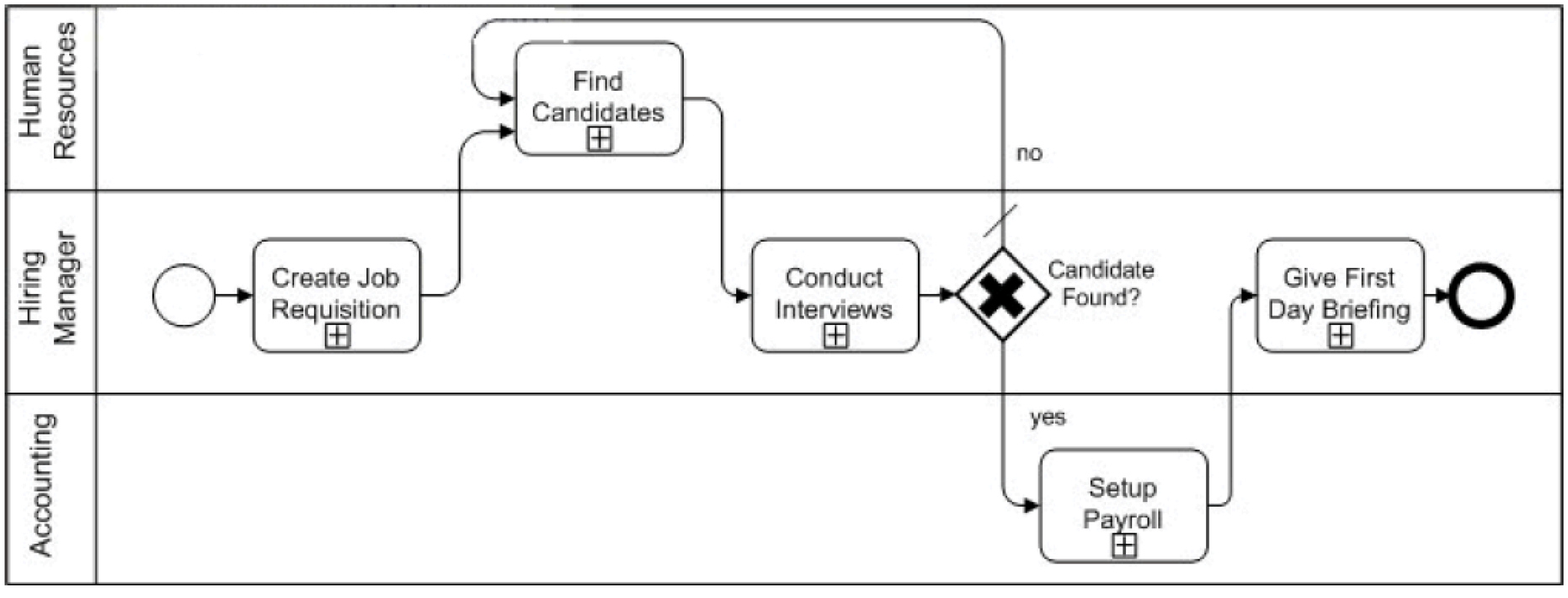
- Model interpretation: reading and executing the DSL script at runtime one statement at a time, exactly as programming languages interpreters do.

- Code-generation: applying a complete model-to-text (M2T) transformation at deployment time, thus producing an executable application, as compilers do for programming languages.

- See Chapter 2 for model executability details.

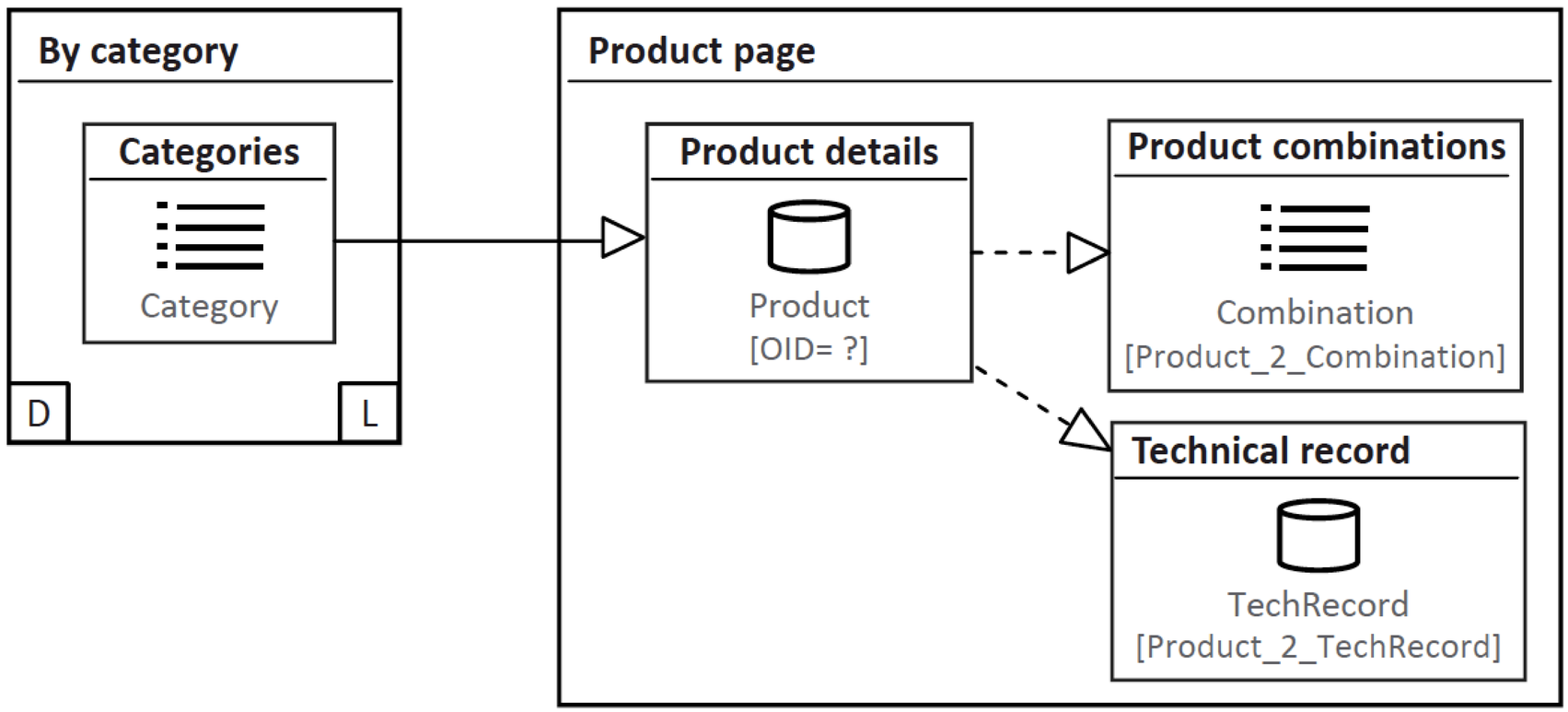
DSL example (1): BPMN process model

- Graphical, external, imperative, horizontal DSL for specifying business processes



DSL example (2): WebML hypertext model

- Declarative, graphical, horizontal DSL for modeling Web navigation UIs. Supporting tool WebRatio applies a full code generation approach for executing the models.



DSL example (3): VHDL specs

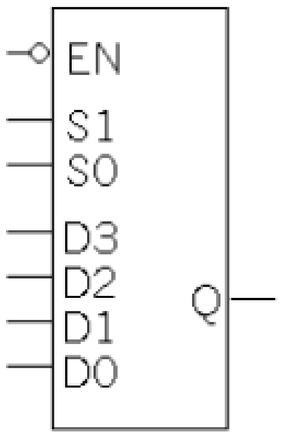
- Textual, external, declarative, vertical DSL for specifying the behaviour of electronic components.
- Example: definition of a Multiplexer in VHDL

```
entity mux4_to_1 is
      port (I0,I1,I2,I3: in std_logic_vector(7 downto 0);
       SEL: in std_logic_vector (1 downto 0);
        OUT1: out std_logic_vector(7 downto 0));
end mux4_to_1;
```

- Multiplexer (MUX): electronic component that selects one of several analog or digital input signals and forwards the selected input into a single output line.
- See more at: http://en.wikipedia.org/wiki/Multiplexer

DSL Example (3). Cont.d

- Alternative representation of the multiplexer, according to different notations (which could be seen as DSLs themselves):
- Electronic block diagram, truth table, output boolean expression



EN'	S1	S0	Q
0	0	0	D0
0	0	1	D1
0	1	0	D2
0	1	1	D3
1	x	x	1

$$Q = S1' \, S0' \, D0 + S1' \, S0 \, D1 + S1 \, S0' \, D2 + S1 \, S0 \, D3$$