Τεχνολογία Λογισμικού

Ν.Παπασπύρου, Αν.Καθ. ΣΗΜΜΥ, nickie@softlab.ntua,gr

Β.Βεσκούκης, Αν.Καθ. ΣΑΤΜ, v.vescoukis@cs.ntua,gr

Κ.Σαΐδης, ΠΔ 407, saiko@softlab.ntua.gr

Εισαγωγή στη UML

Unified Modeling Language

OMG Standard, Object Management Group

- Based on work from Booch, Rumbaugh, Jacobson

UML is a modeling language to express and design documents, software, systems and more

- Created with OO analysis and design,
  but has evolved to cover more than software systems
- UML is NOT a methodology, process, etc
- Independent of implementation language

Unified Modeling Language

Open Standard, Graphical notation for Software Systems, from initial conception to detailed design, across the entire software lifecycle

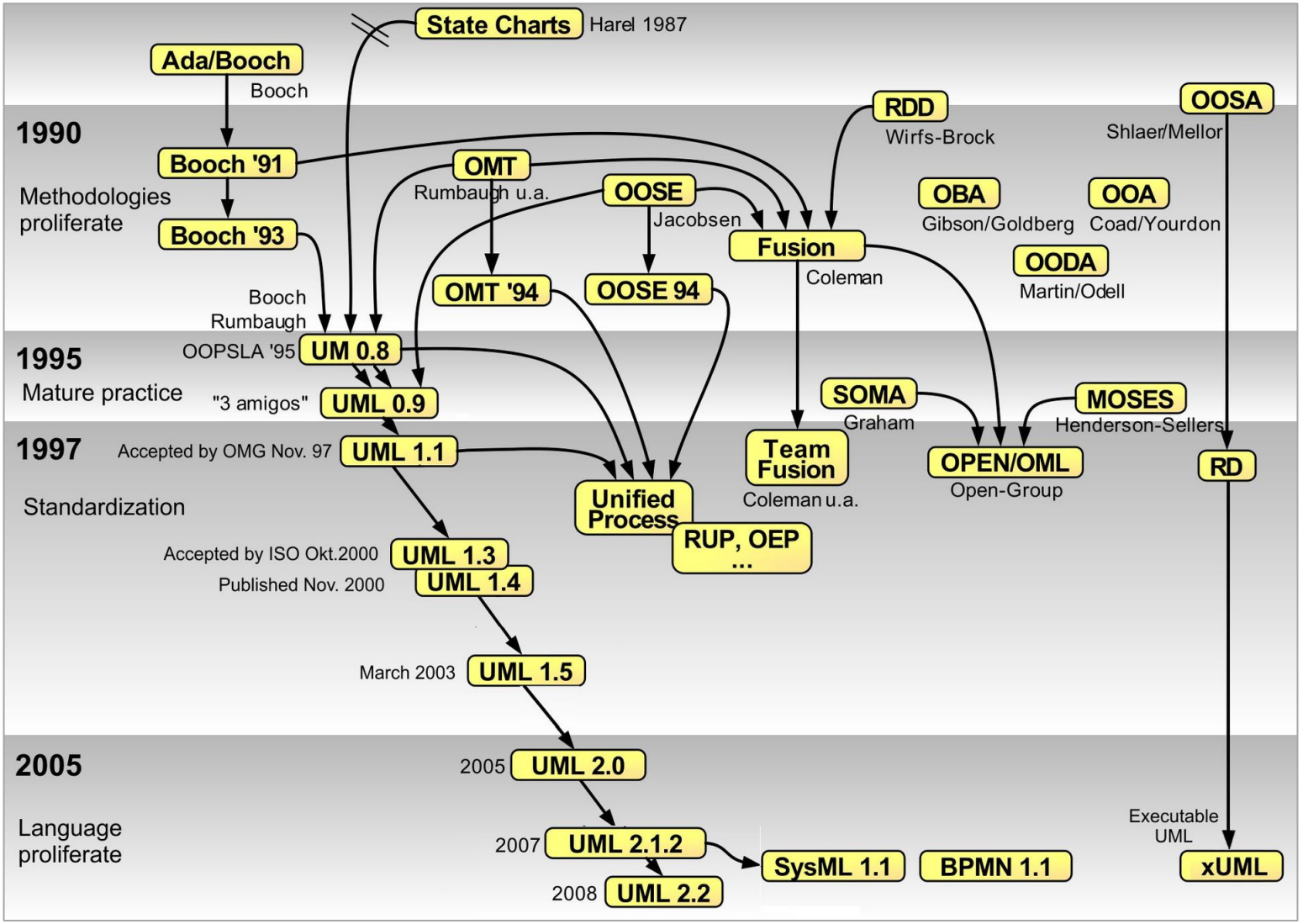
- specification
- visualization
- construction
- documentation

Support understanding of software to customers and developers

Support for diverse application areas

Based upon experience and needs of the user community

History


State Charts
Harel 1987
Ada/Booch
Booch
1990
Booch '91
Methodologies
proliferate
Booch '93
OMT
Rumbaugh u.a.
OOSE
Jacobsen
RDD
Wirfs-Brock
OOSA
Shlaer/Mellor
OBA
Gibson/Goldberg
OOA
Coad/Yourdon
Fusion
Coleman
OODA
Martin/Odell
OMT '94
OOSE 94
Booch
Rumbaugh
1995
OOPSLA '95
UM 0.8
Mature practice
"3 amigos"
UML 0.9
SOMA
Graham
MOSES
Henderson-Sellers
1997
Accepted by OMG Nov. 97
UML 1.1
Team
Fusion
Coleman u.a.
OPEN/OML
Open-Group
RD
Standardization
Unified
Process
RUP, OEP
...
Accepted by ISO Okt.2000
Published Nov. 2000
UML 1.3
UML 1.4
March 2003
UML 1.5
2005
2005
UML 2.0
Language
proliferate
2007
UML 2.1.2
2008
UML 2.2
SysML 1.1
BPMN 1.1
Executable
UML
xUML

UML concepts

Systems, Models, Views

- A model is an abstraction describing a subset of a system
- A view depicts selected aspects of a model
- A notation is a set of graphical or textual rules for depicting views
- Views and models of a single system may overlap each other

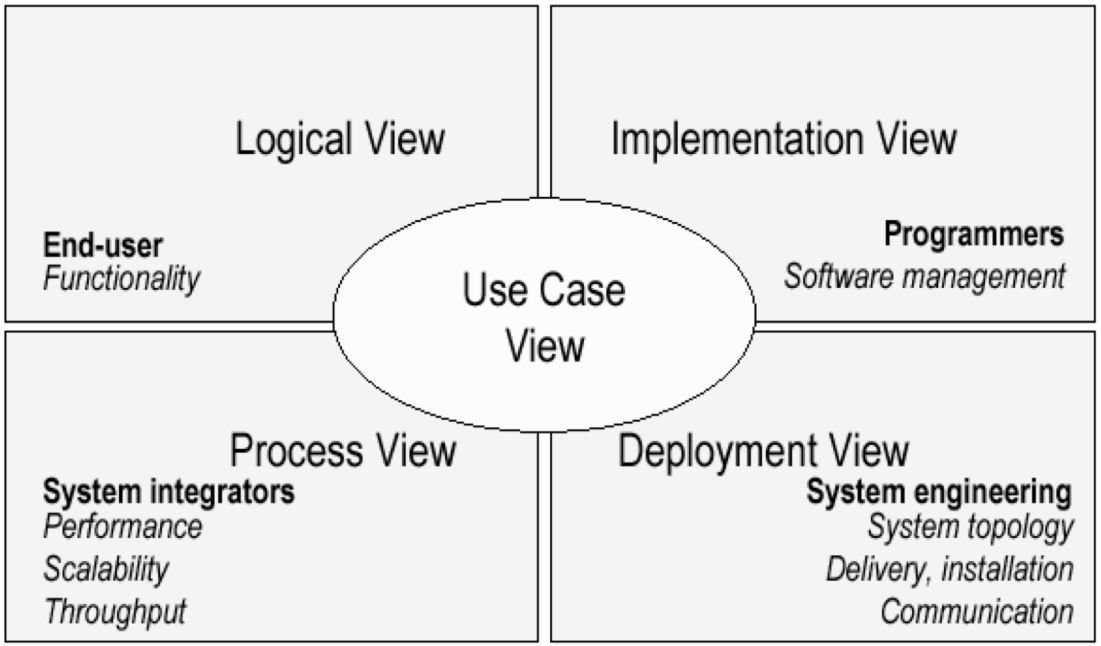
Example

- System: Aircraft
- Models: Flight simulator, scale model
- Views: All blueprints, electrical wiring, fuel system

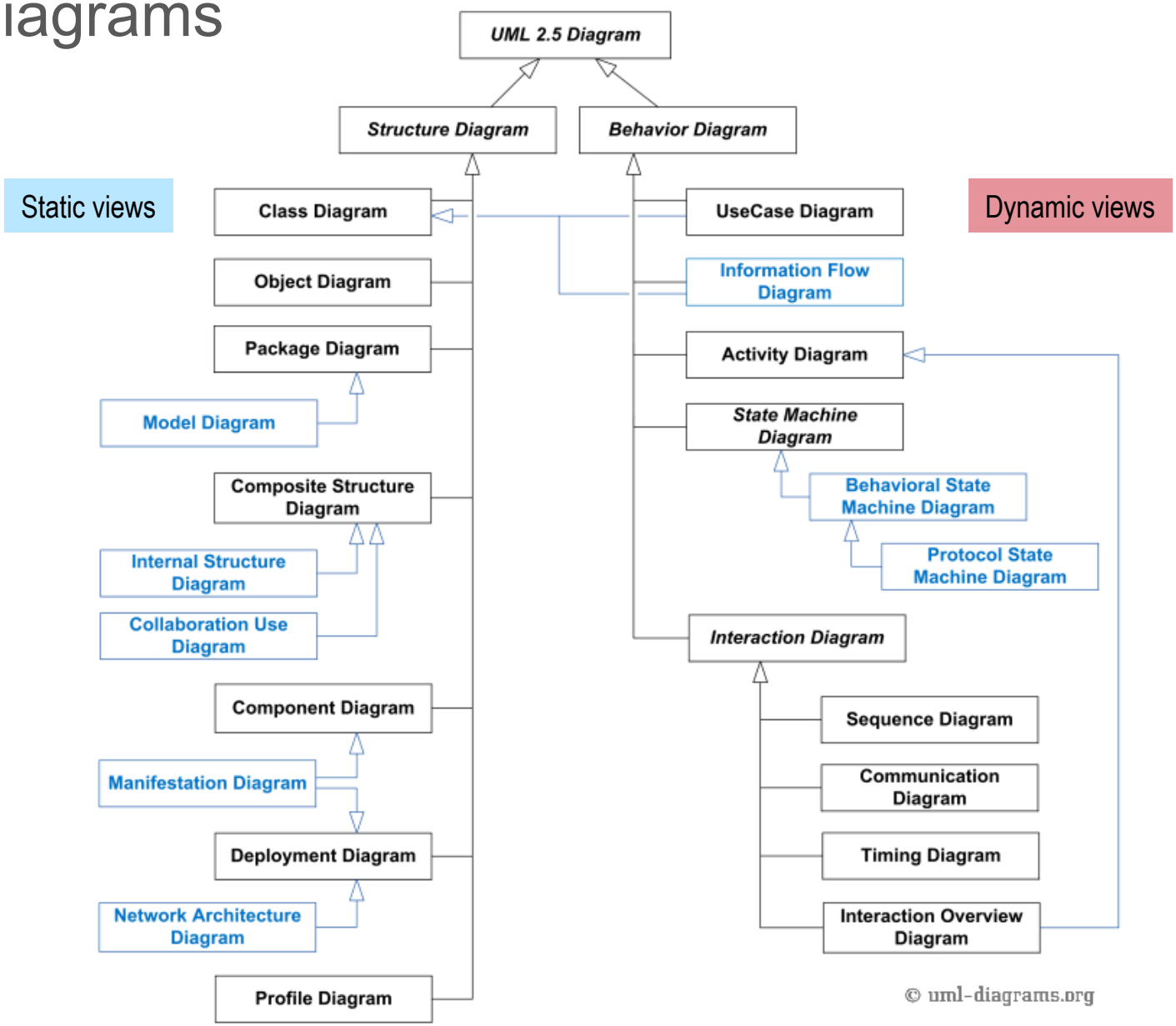
UML models, views, diagrams

UML defines many diagrams, each of which is a view into a model

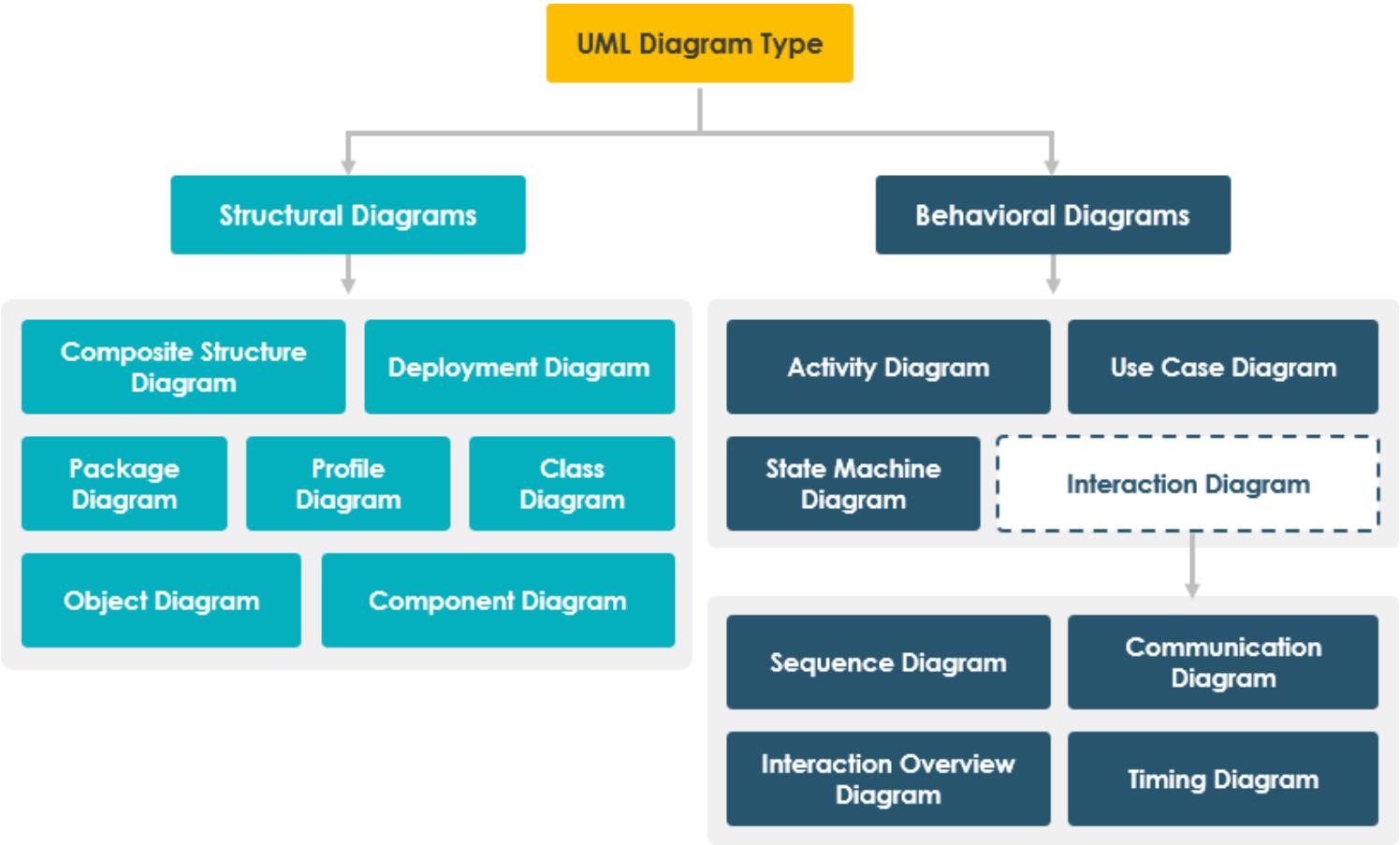
- Diagram presented from the aspect of a particular stakeholder
- Provides a partial representation of the system
- Is semantically consistent with other views



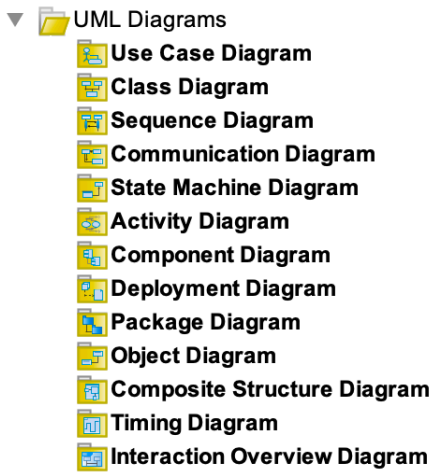
UML diagrams



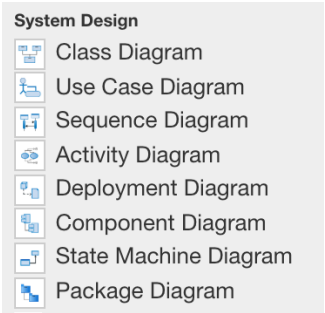
UML diagrams



Visual Paradigm CE



Visual Paradigm online



UML views: focus on what's needed

Not all systems require all views

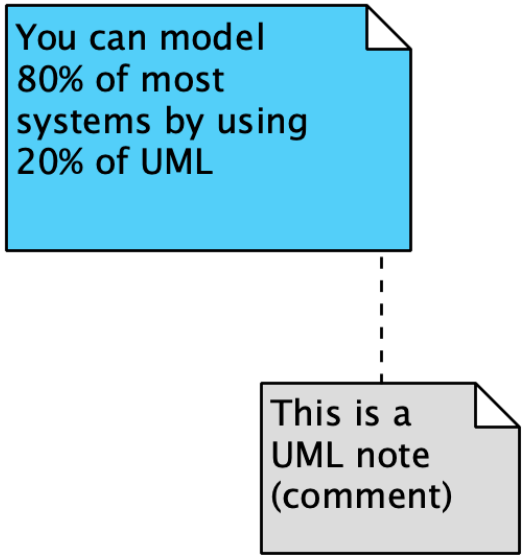
- Single execution node: drop deployment view
- Single process: drop process view
- Very small program: drop implementation view

A system might need additional views

- Data view, security view, …

Identification of "useful" views depends on the context and intended use of the UML model of a system

- Communication with the client
- System specification
- System design



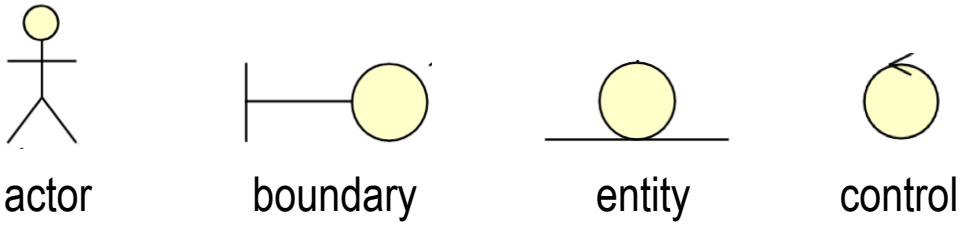
A key concept: stereotypes

Stereotype:
A mechanism for extending the vocabulary (and thus, the expressive power) of UML

Why extend the vocabulary?

- Ecosystem- / stack- / framework- specific terminology
- Comprehensive architecture visualization

Use with measure!



Basic UML modeling

Use Cases

- Capture requirements

Domain Model

- Capture process, key classes

Design Model

- Capture details and behaviors of use cases and domain objects
- Add classes that do the work and define the architecture

Basic UML modeling

Use Case Diagrams

Class Diagrams / Package Diagrams

Interaction Diagrams

- Sequence Diagrams
- Collaboration (a.k.a. Communication Diagrams)

Activity Diagrams / State Transition Diagrams

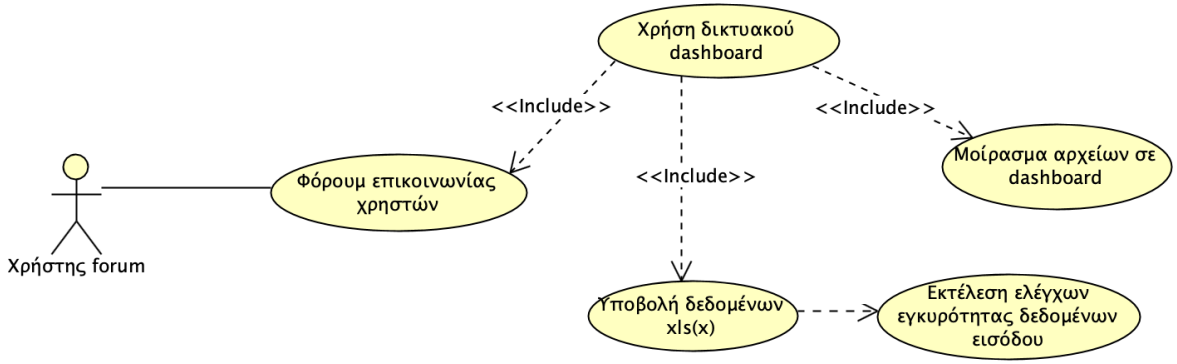
Component Diagrams / Deployment Diagrams

Use Case diagrams

What is a Use Case – key concepts

- **Use cases** represent a sequence of interaction(s) for a type of functionality
- **Actors** represent roles. A role is a type of user of the system, and can even be another system (external system)
- Used during requirements elicitation to represent external behavior

The use case model is the set of all use cases. It is a complete description of the functionality of the system and its environment



Use cases vs. Requirements

A Use Case usually groups some requirements together in the context of an interaction of the system with some external entity.

The granularity of the requirements' definition determines the level of grouping requirements in use cases

By: Transitor

(133) Requirement

	Αίτημα αλλαγής κατάστασης υποβολ	Δημιουργία πρότυπων αναφορών μι	Δημιουργία πρότυπων αναφορών ορ	Διαχείριση οργάνωσης δεδομένων BI	Διαχείριση στοιχείων φορέων	Διαχείριση υποχρεώσεων – αυτόματ	Διαχείριση χρηστών – δικαιωμάτων	Διαχείριση χρηστών – δικαιωμάτων	Δυναμικά ερωτήματα – διαδραστική	Εισαγωγή εξωτερικών δεδομένων σ	Εκτέλεση ελέγχων εγκυρότητας δεδο	Έλεγχος διάθεσης web services	Ενοποίηση με εργαλεία office	Επεξεργασία templates ΔΕ	Μοίραζμα αρχείων σε dashboard	Παραγωγή αναφορών – infographics	Παραγωγή αναφορών στοχοθεσίας	Παραγωγή ΔΕ	Παραγωγή ομάδων αναφορών – δευ	Παραμετροποίηση – αντιστοίχιση λο	Παραμετροποίηση web services	Παραμετροποίηση έλέγχων εγκυρότ	Παραμετροποίηση ομαδοποίησης αν	Παραμετροποίηση στοχοθεσίας	Πραμετροποίηση μεταχηματισμών	Υποβολή – διαχείριση λογιστικών σχ	Φόρουμ επικοινωνίας χρηστών	Χρήση δικτυακού dashboard
BICS21: Λειτουργία παραμετροποίησης από τους διαχειριστές συνθηκών εγκυρότητας ...																						✓					
BICS22: Ελεγχος εγκυρότητας μηνιαίων δεδομένων ενδοκυβερνητικών συναλλαγών											✓																
BICS23: Λειτουργία παραμετροποίησης από τους διαχειριστές συνθηκών εγκυρότητας ...																						✓					
BICS24: Ελεγχος εγκυρότητας μηνιαίων δεδομένων ισοζυγίου											✓																
BICS25: Λειτουργία παραμετροποίησης από τους διαχειριστές συνθηκών εγκυρότητας ...																						✓					
BICS26: Ελεγχος εγκυρότητας δημοσιευμένων ισολογισμών											✓																
BICS27: Λειτουργία παραμετροποίησης από τους διαχειριστές συνθηκών εγκυρότητας ...																						✓					
BICS28: Ελεγχος εγκυρότητας σχεδίων προϋπολογισμών											✓																
BICS29: Λειτουργία παραμετροποίησης από τους διαχειριστές συνθηκών εγκυρότητας ...																						✓					
BICS30: Ελεγχος εγκυρότητας αρχικών/τροποποιημένων προϋπολογισμών											✓																
BICS31: Λειτουργία παραμετροποίησης από τους διαχειριστές συνθηκών εγκυρότητας ...																						✓					
BICS32: Ελεγχος εγκυρότητας σχεδίων ΜΠΔΣ											✓																
BICS33: Λειτουργία παραμετροποίησης από τους διαχειριστές συνθηκών εγκυρότητας ...																						✓					

Use Cases and Actors

An actor models an external entity which communicates with the system and triggers some of its functionality:

- User
- External system
- Physical environment

An actor has a unique name and an optional description

Examples:

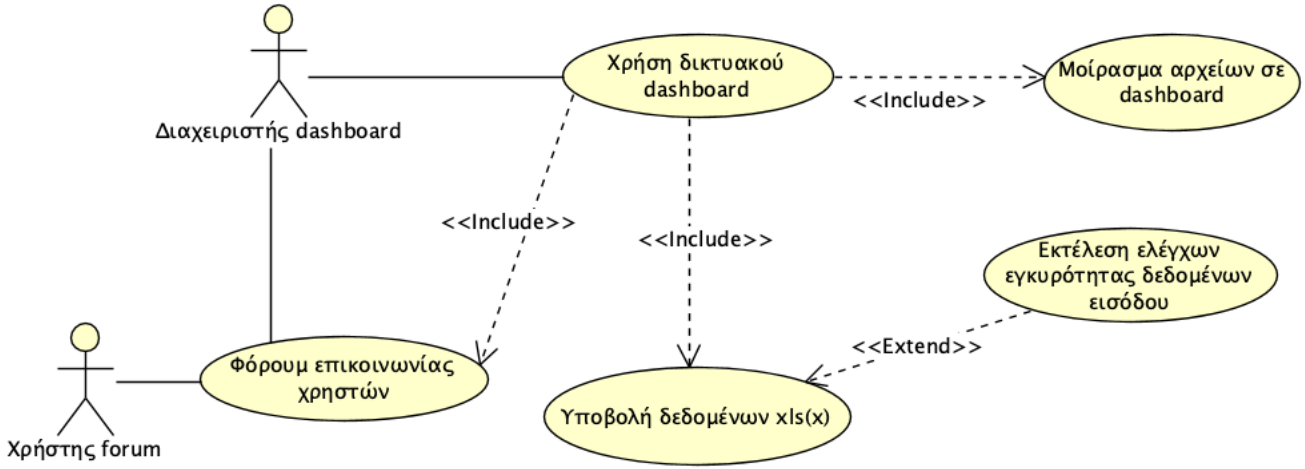
- Passenger: A person issuing a ticket
- GPS device: Provides the system with GPS coordinates

Use Cases and Actors

A use case represents a class of functionality provided by the system as an event flow

A use case consists of:

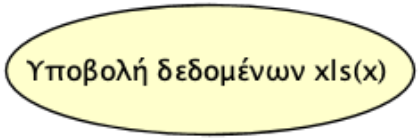
- Unique name
- Participating actors
- Entry conditions
- Flow of events
- Exit conditions
- Special requirements



Use Case: example

Unique name

- Υποβολή δεδομένων xls(x)



Participating actors

- Διαχειριστής dashboard

Entry conditions

- xls(x) file is available; server has enough disk space free

Flow of events

- User drags file to designated area; file is uploaded to the server

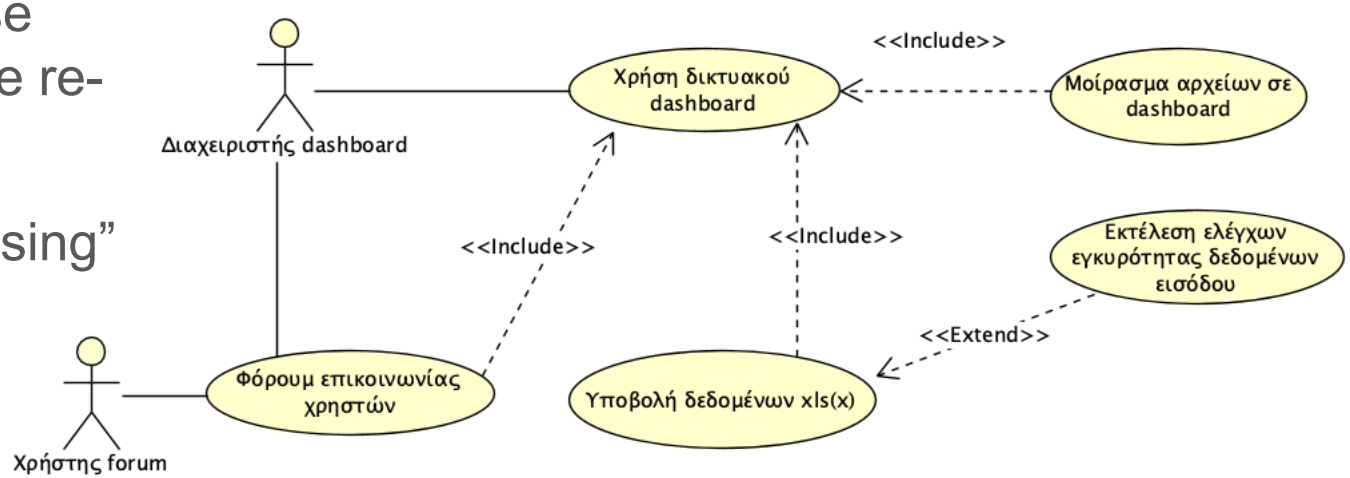
Exit conditions

- File is saved on the server

Use Case diagrams: <<include>> and <<extend>>

Include:

- Behavior that has been factored out of the Use Case, so that it can be re-used
- Arrow points to the ”using” Use Case



Extends

- Exceptional, rarely invoked Use Cases
- Arrow points to the extended Use Case

Use Case Diagrams are useful for…

Determining requirements

- New use cases often generate new requirements as the system is analyzed and the design takes shape.

Communicating with clients

- Their notational simplicity makes use case diagrams a good way for developers to communicate with clients.

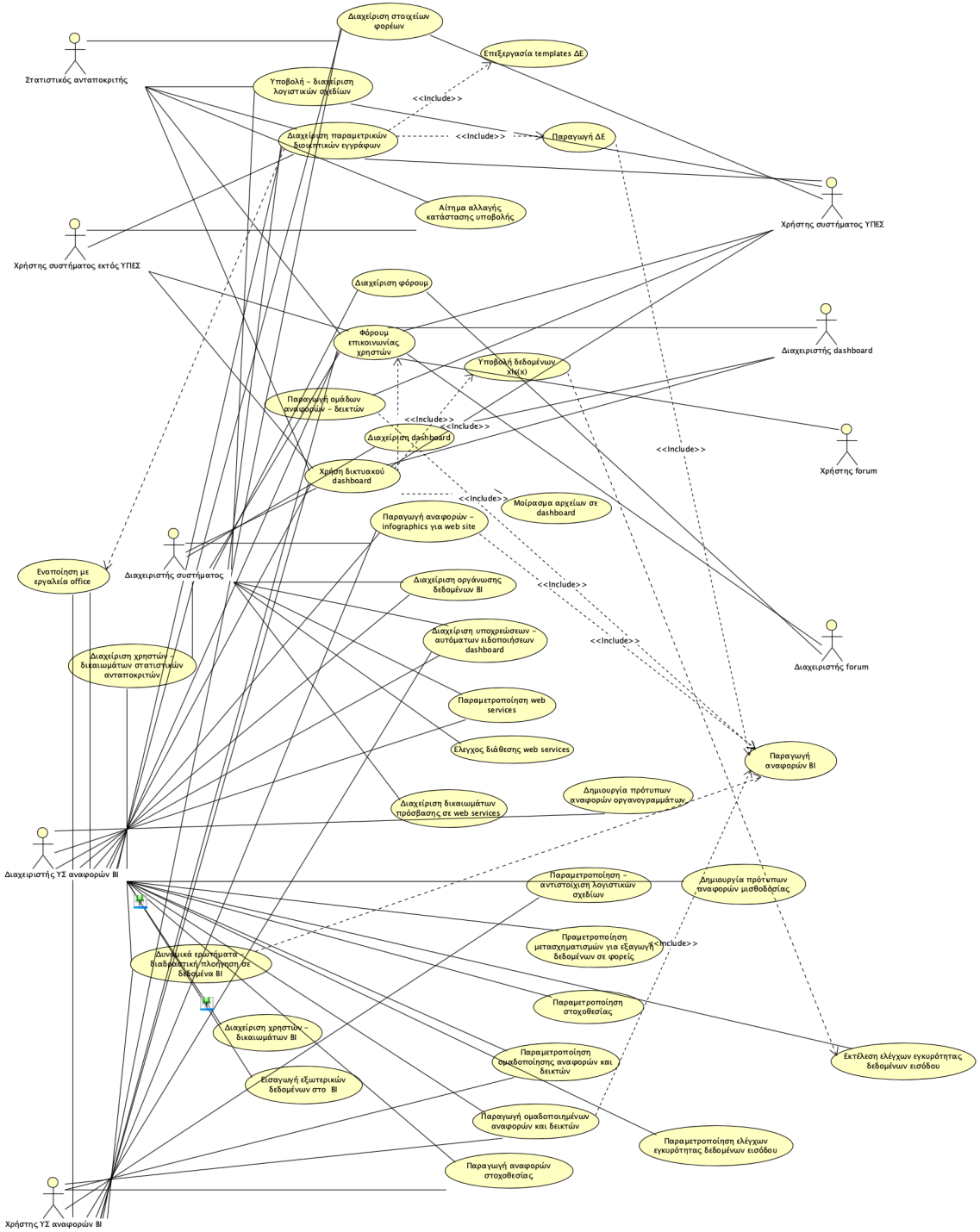
Generating test cases

- The collection of scenarios for a use case may suggest a suite of test cases for those scenarios.

Use case descriptions provide the info needed: not use case diagrams!
All use cases need to be described for the model to be useful.

Use Case Diagrams

A complete Use Case model (diagram)



Class Diagrams

A Class Diagram…

Gives an overview of a system by showing its classes and the relationships among them.

- class diagrams are static
- they display what interacts but not what happens when interactions occur

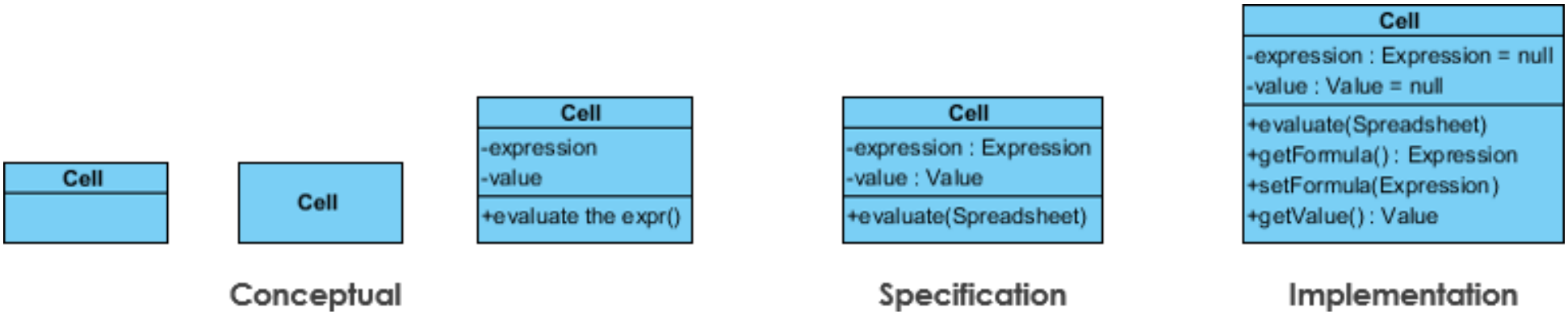
Also shows attributes and operations of each class

Good way to describe the overall architecture of system components

Class Diagram: Perspectives

We draw Class Diagrams under three perspectives

- Conceptual
  - Software independent
  - Language independent
- Specification
  - Focus on the interfaces of the software
- Implementation
  - Focus on the implementation of the software



Classes: Not Just for Code

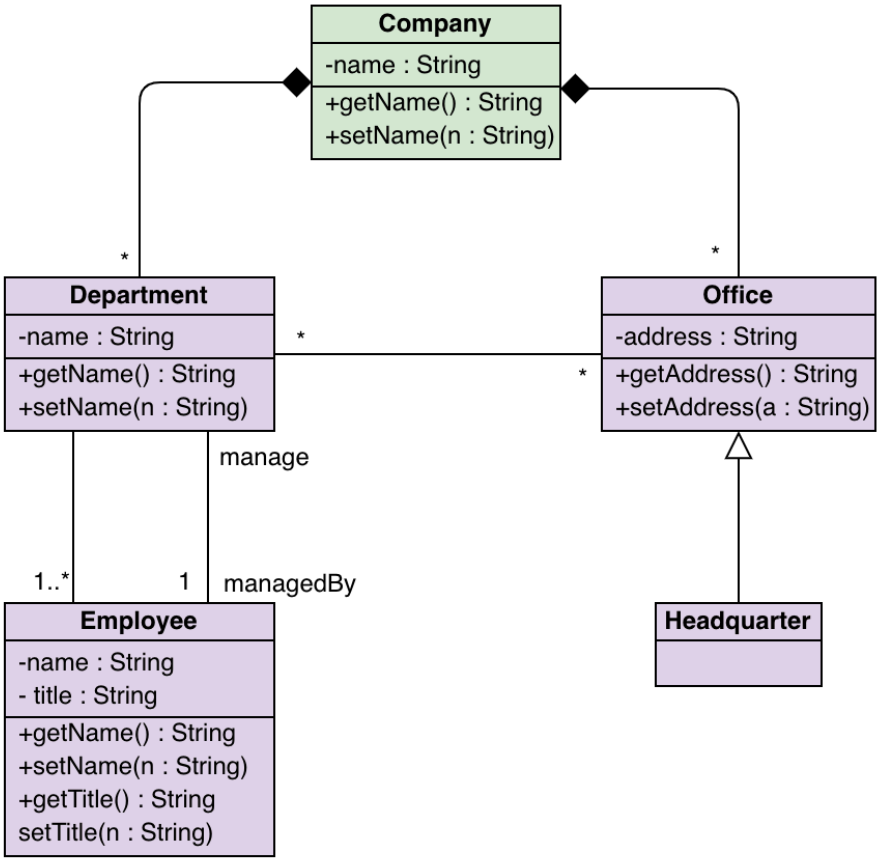
A class represent a concept

A class encapsulates state (attributes) and behavior (operations).

Each attribute has a type.

Each operation has a signature.

The class name is the only mandatory information.

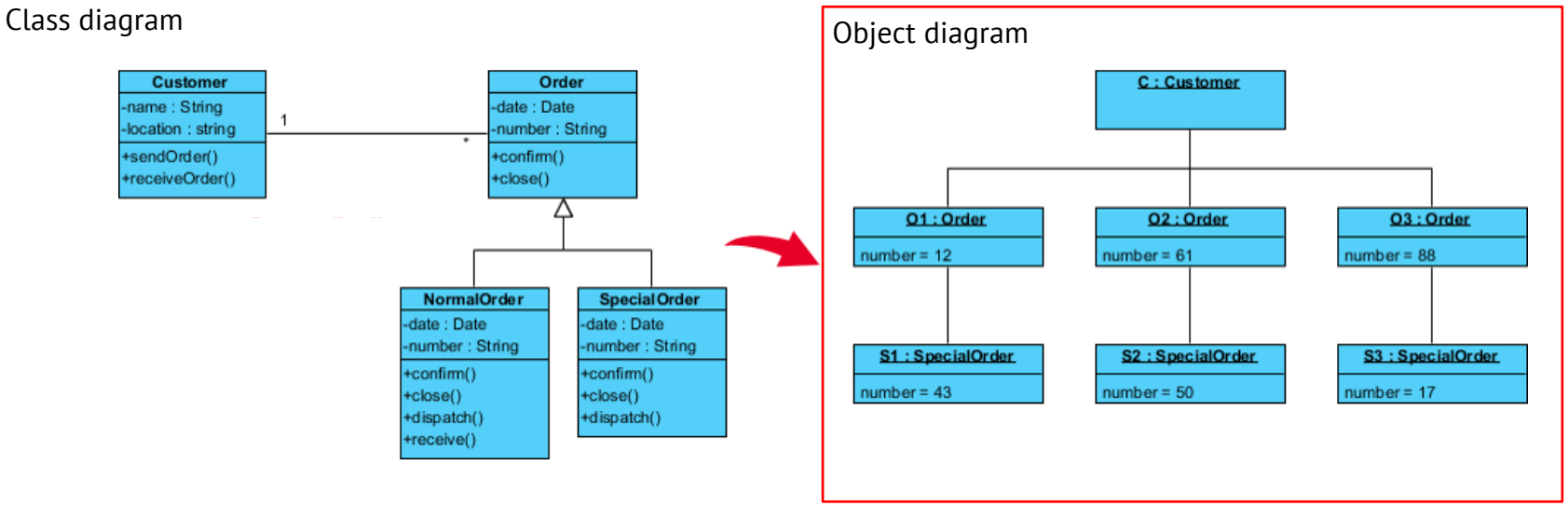


Instances

An ***instance*** represents a phenomenon (= a specific object).

The name of an instance is underlined and can contain the class of the instance.

The attributes are represented with their values.



UML Class Notation

A class is a rectangle divided into three parts

- Class name
- Class attributes (i.e. fields, variables)
- Class operations (i.e. methods)

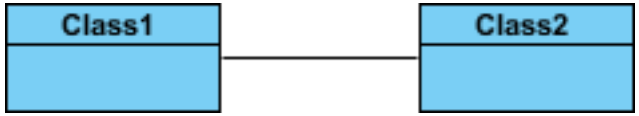
Modifiers

- Private: -
- Public: +
- Protected: #
- Static: Underlined (i.e. shared among all members of the class)

Abstract class: name in italics

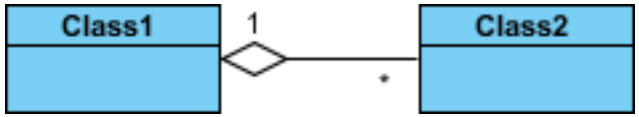
UML Class Notation: Relationships

Association



- A relationship between instances of two classes, where one class must know about the other to do its work, e.g. client communicates to server
- Indicated by a straight line or arrow

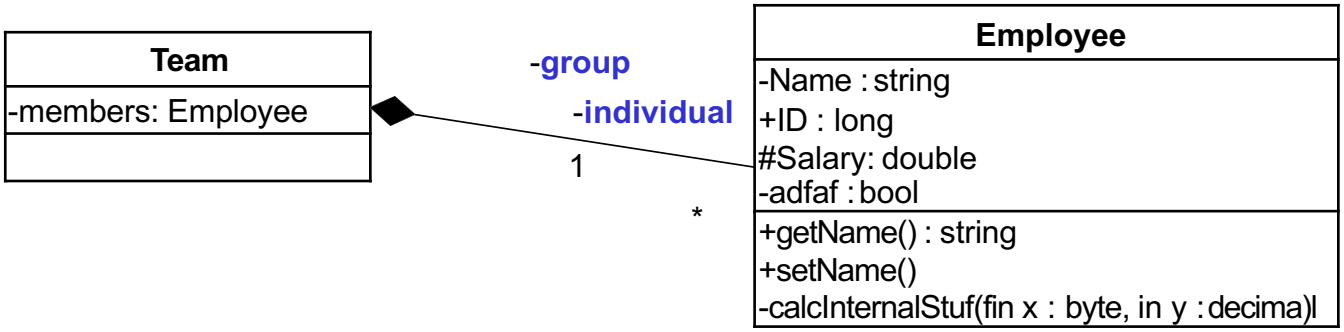
Aggregation



- An association where one class belongs to a collection
- Indicated by an empty diamond on the side of the collection
- Members can exist independently of the aggregate ("parent") e.g.: students exist even if there is no class scheduled

Association Details

Can assign names to the ends of the association to give further information

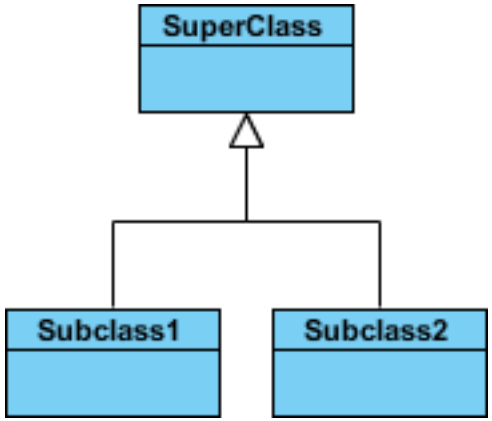


UML Class Notation

Composition

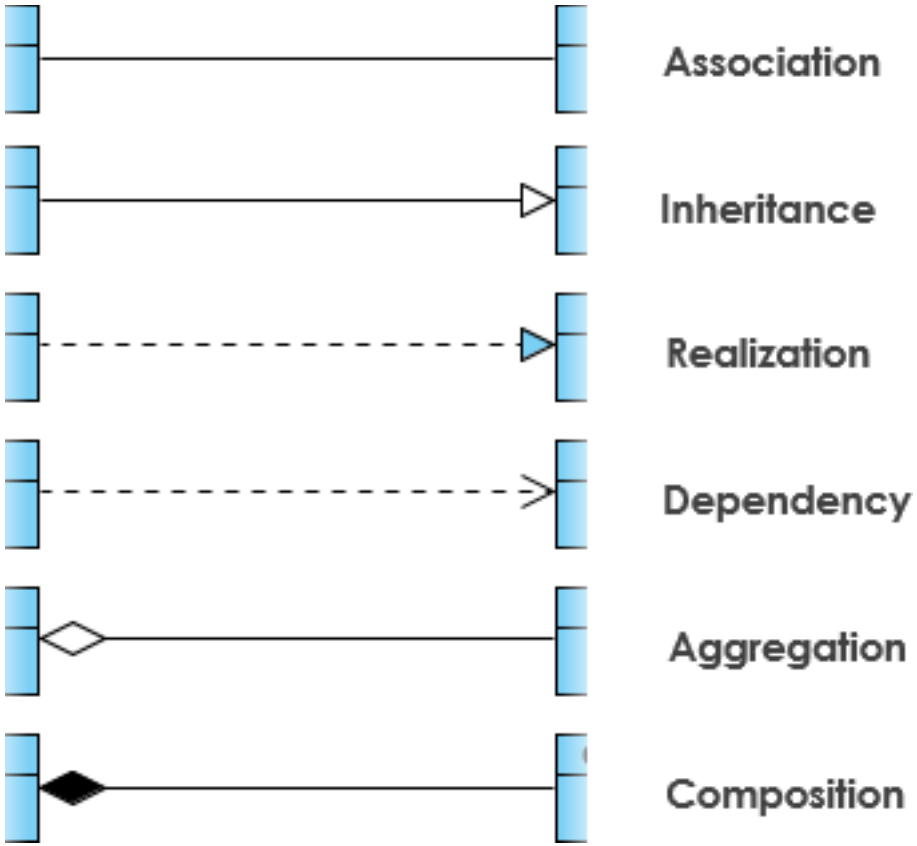
- Strong form of Aggregation
- Lifetime control: components cannot exist without the aggregate (e.g.: parts of an aircraft)
- Indicated by a solid diamond on the side of the collection

Inheritance



- Inheritance represents a "is-a" relationship
- Key element of object orientation
- Indicated by a hollow arrowhead pointing to the superclass ("parent")

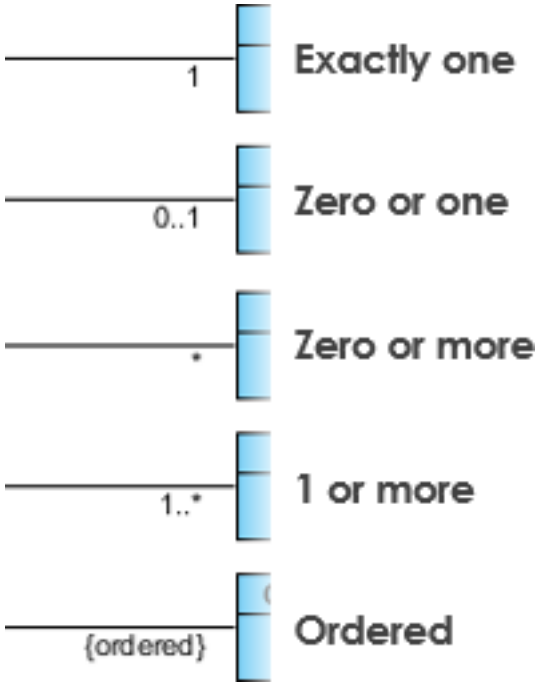
UML Class diagram notation



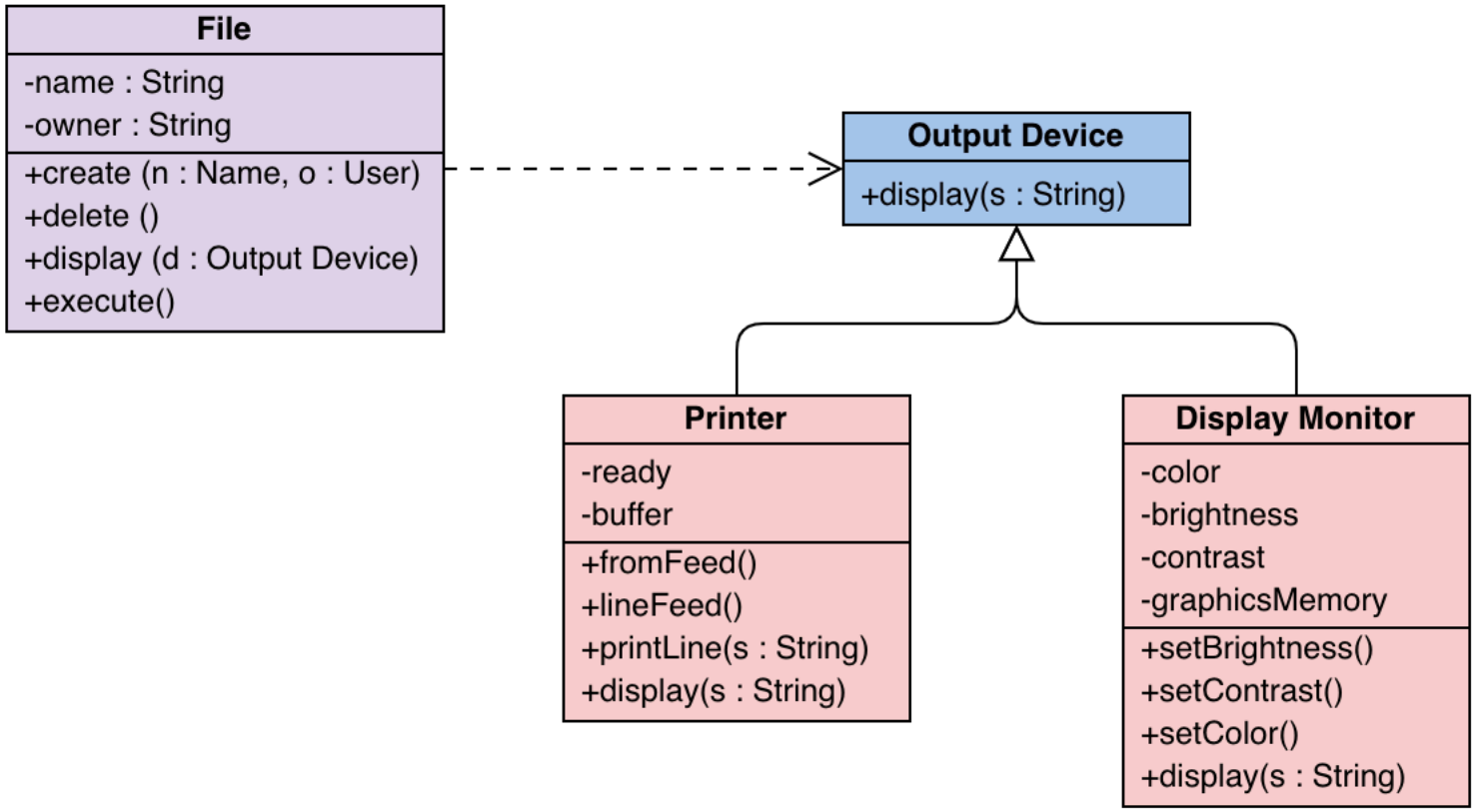
UML Multiplicities

Links on associations to specify more details about the relationship

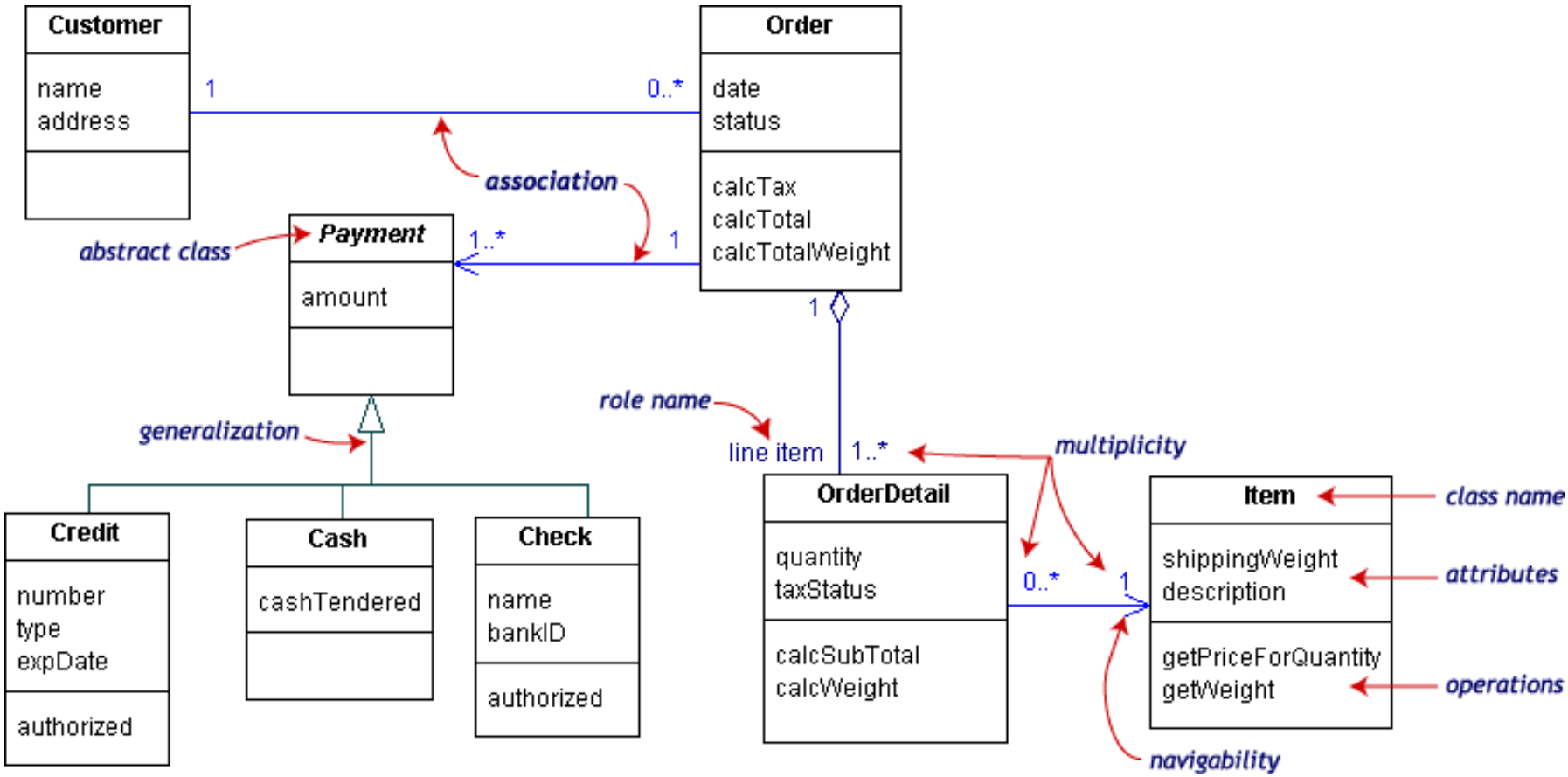
Multiplicities	Meaning
0..1	zero or one instance. ***n . . m*** indicates ***n*** to ***m*** instances.
0..* *or* *	zero to unlimited instances
1	exactly one instance
1..*	at least one instance

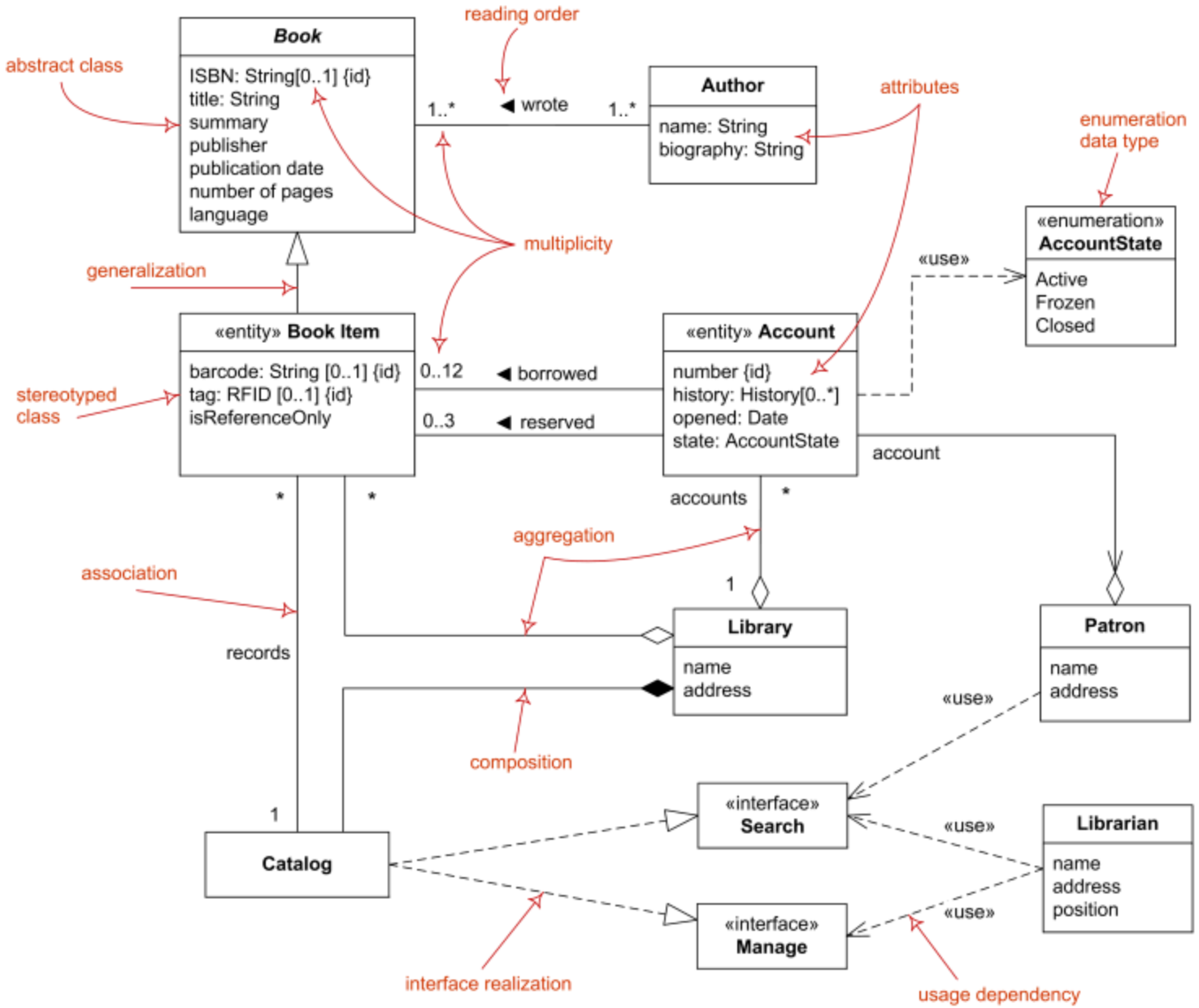


UML Class Diagram example

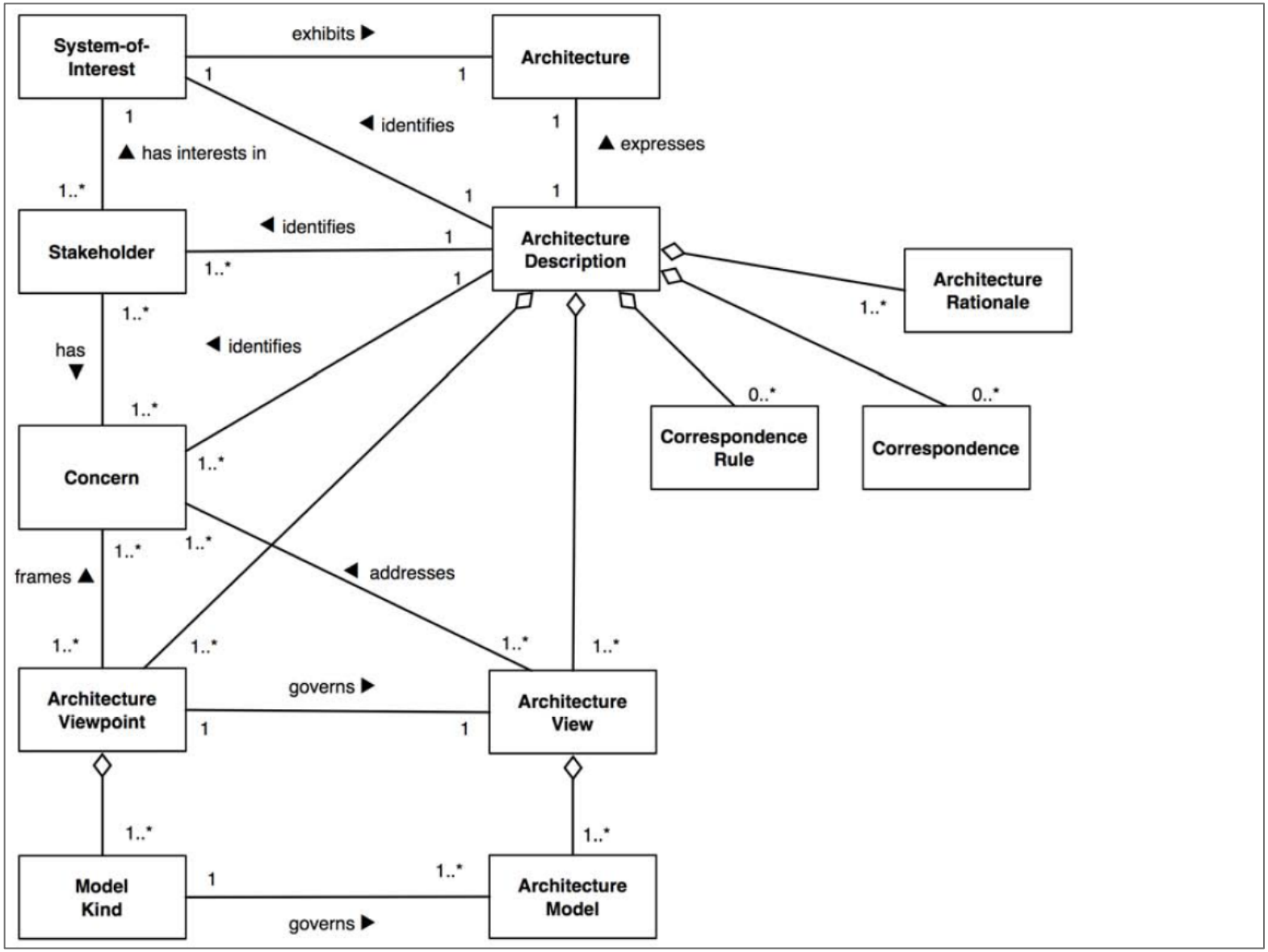


UML Class Diagram Example

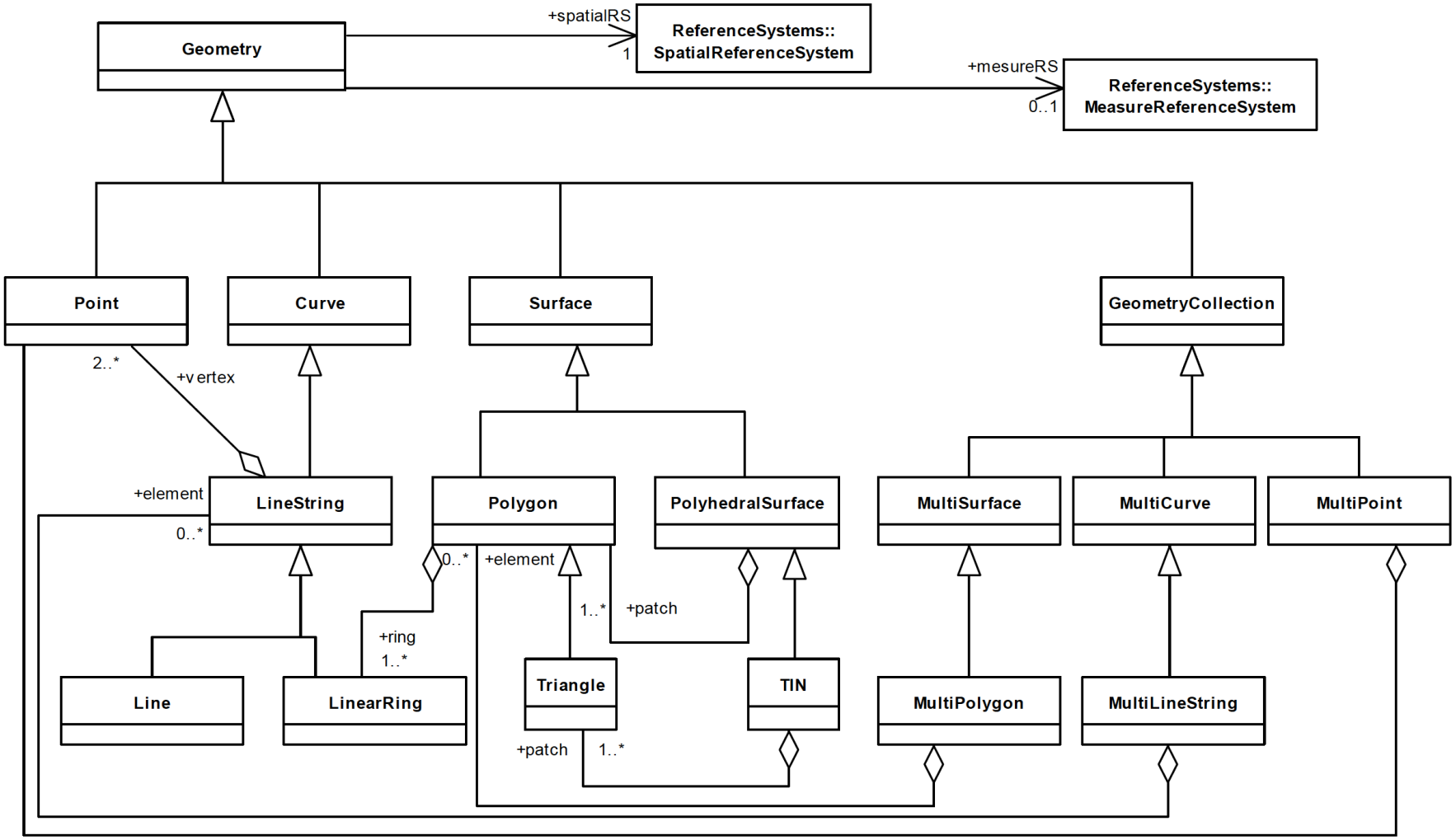



abstract class
Book
ISBN: String[0..1] {id}
title: String
summary
publisher
publication date
number of pages
language
reading order
1..*
wrote
1..*
Author
name: String
biography: String
attributes
enumeration
data type
«enumeration»
AccountState
Active
Frozen
Closed
«use»
multiplicity
generalization
«entity» Book Item
barcode: String [0..1] {id}
tag: RFID [0..1] {id}
isReferenceOnly
stereotyped
class
0..12
borrowed
0..3
reserved
«entity» Account
number {id}
history: History[0..*]
opened: Date
state: AccountState
account
*
*
accounts
*
aggregation
association
1
Library
name
address
Patron
name
address
records
composition
«use»
1
«interface»
Search
«use»
Librarian
name
address
position
Catalog
«interface»
Manage
«use»
interface realization
usage dependency

Class diagram: Software architecture



Class diagram: OCG Simple Features Std



Static vs. Dynamic Design

Static design describes code structure and object relations

- Class relations
- Objects at design time
- Doesn’t change

Dynamic design shows communication between objects

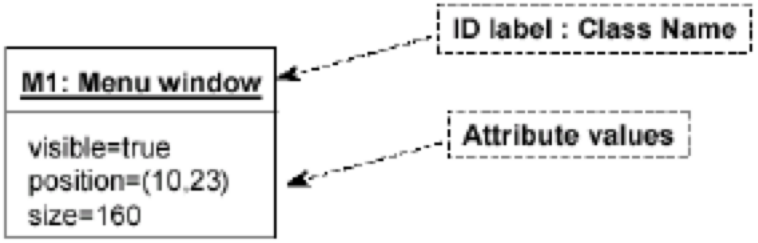
- Similarity to class relations
- Can follow sequences of events
- May change depending upon execution scenario
- Called Object Diagrams

Object diagrams

Shows instances of Class Diagrams and links among them:
An object diagram is a snapshot of the objects in a system at a point in time

Object diagrams focus on representing...

- Interactions – Sequence diagram
- Message passing – Collaboration diagram
- Operation – Deployment diagram



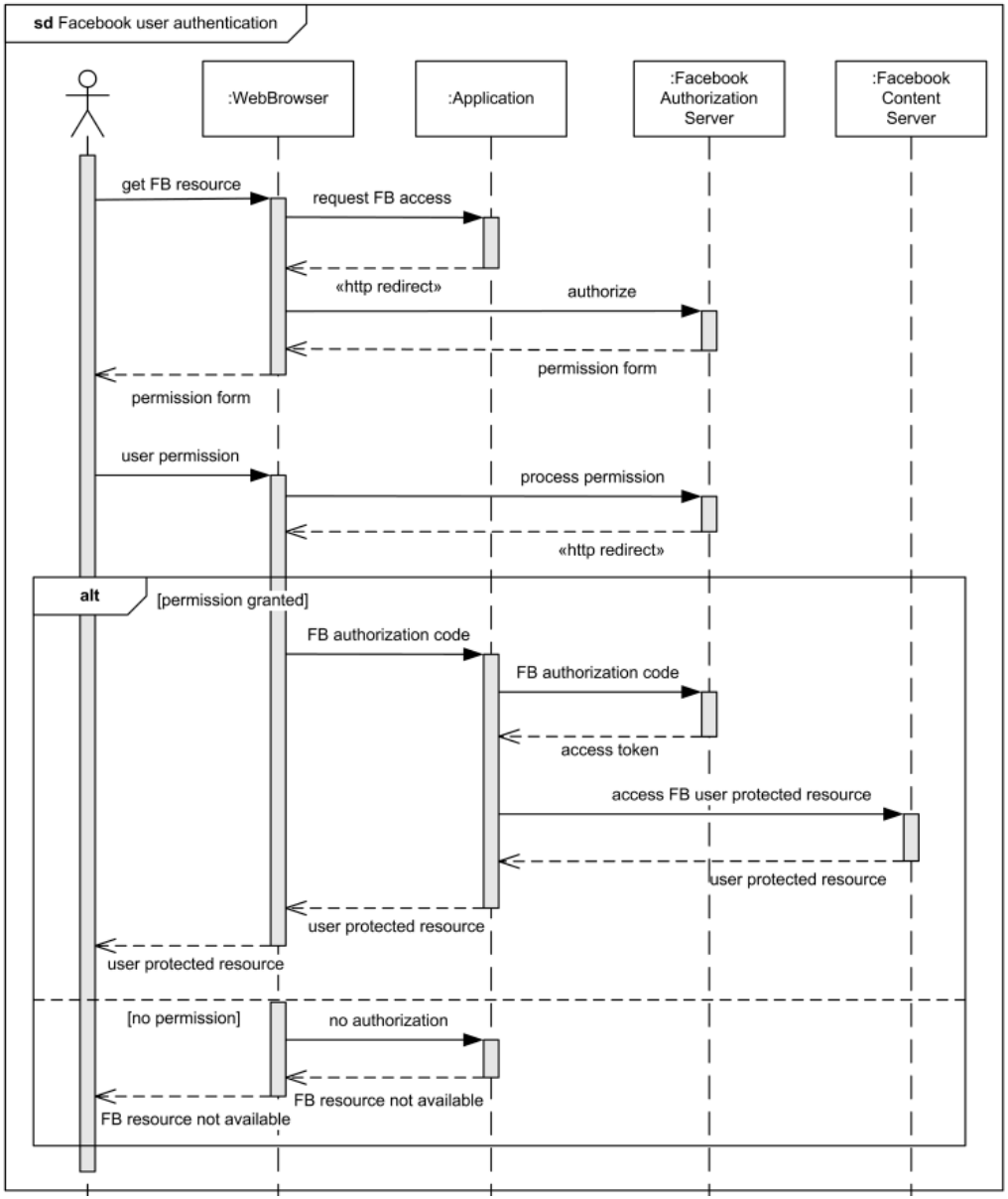
Interactions modeling: Sequence diagrams

Interaction diagrams describe how objects collaborate.

A Sequence Diagram:

- Indicates what messages are sent and when
- Time progresses from top to bottom
- Objects involved are listed left to right
- Messages are sent left to right between objects in sequence

Interactions modeling: Sequence diagrams



Interactions modeling: Sequence diagrams

Actor

Lifeline

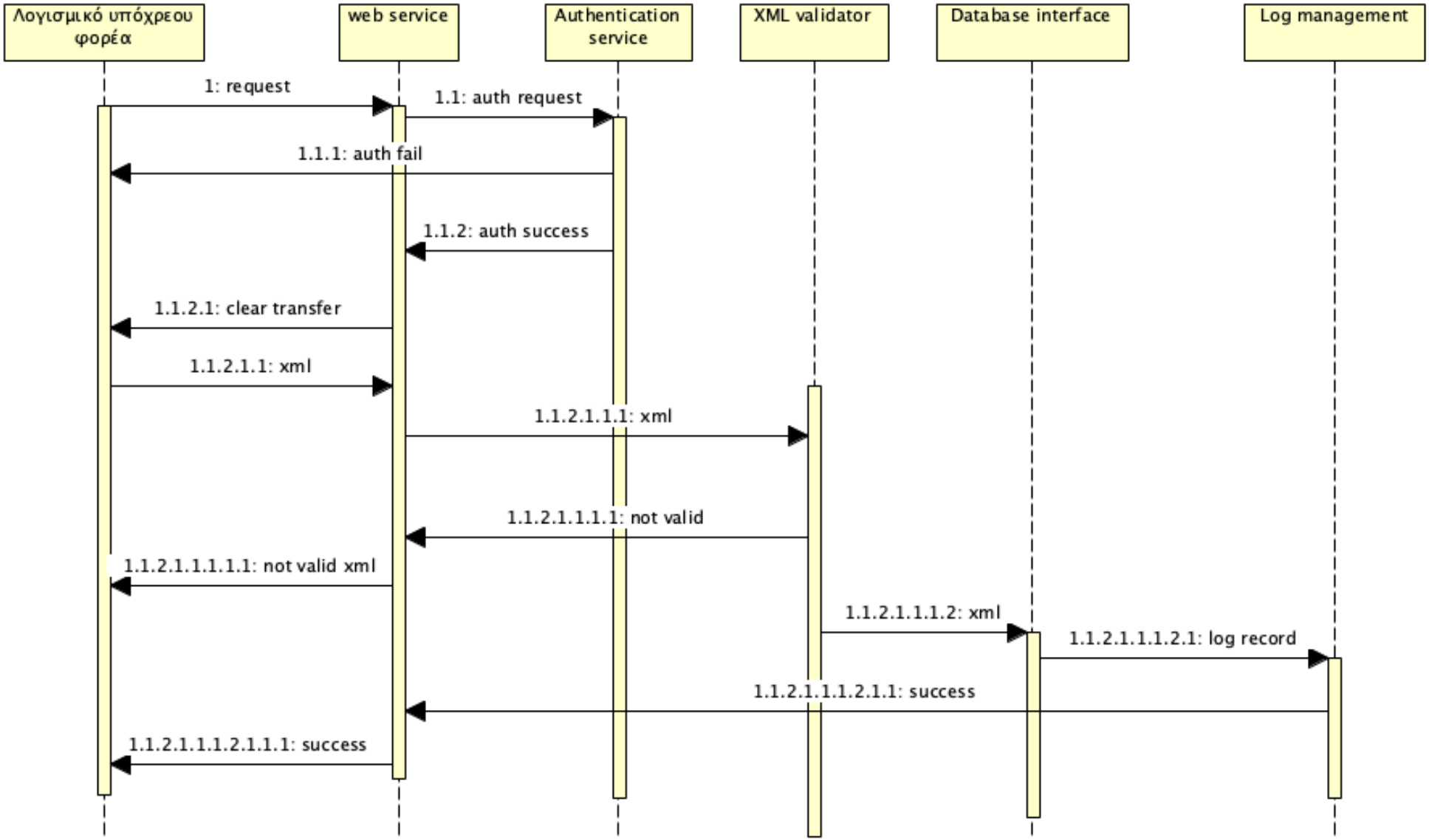
Activation

Messages: call, return, self

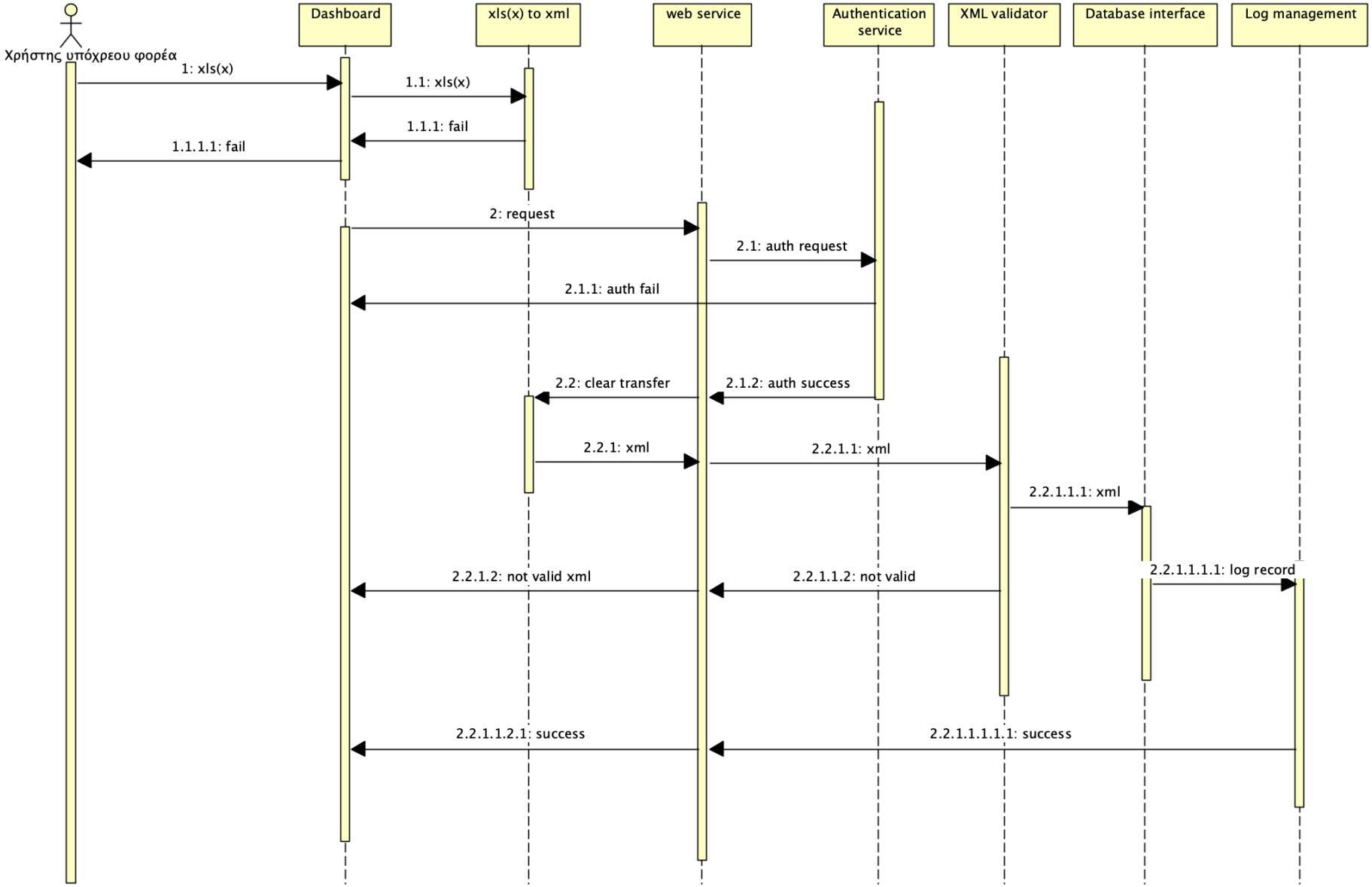
Focus of control: also called execution occurrence

- A tall, thin rectangle on a lifeline
- It represents the period during which an element is performing an operation. The top and the bottom of the rectangle are aligned with the initiation and the completion time respectively.

Interactions modeling: Sequence diagrams



Interactions modeling: Sequence diagram



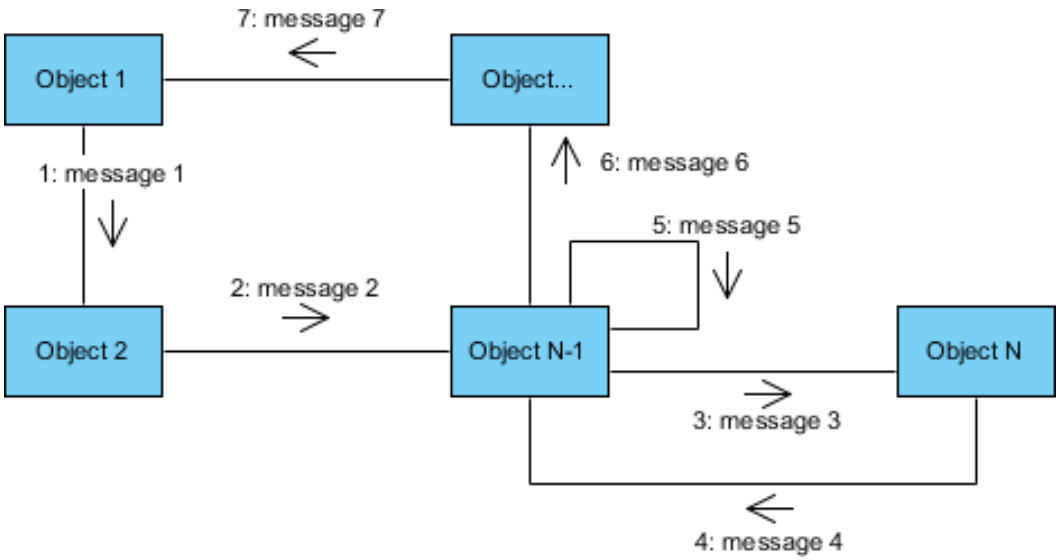
Interactions modeling: Collaboration diagrams

Collaboration Diagrams show similar information to sequence diagrams, BUT the vertical sequence is missing. Instead sequence diagrams use:

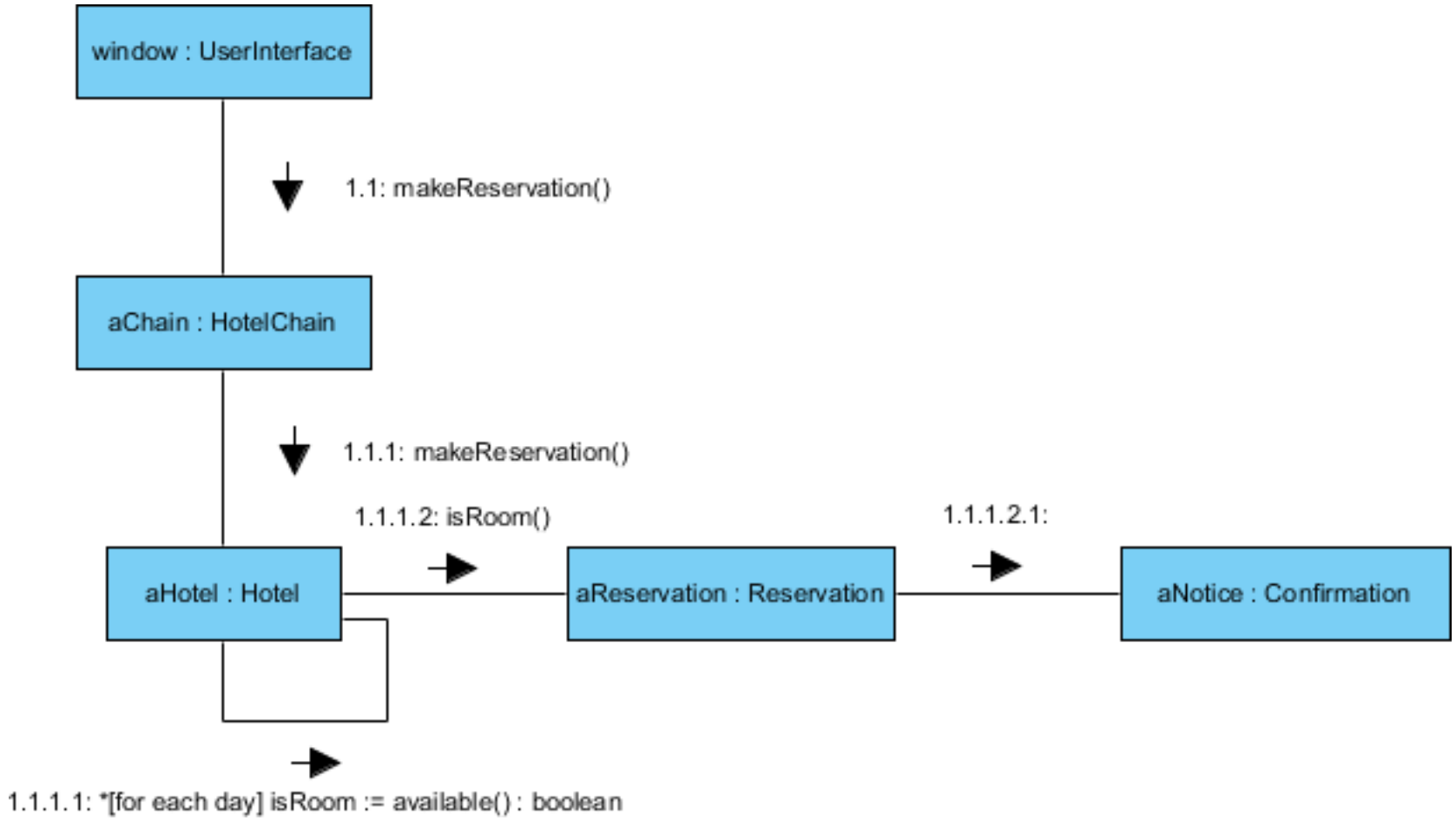
- Object Links - solid lines between the objects that interact
- On the links are Messages - arrows with one or more message name that show the direction and names of the messages sent between objects

Emphasis is on static links as opposed to sequence (= timing, order of things) in the sequence diagram

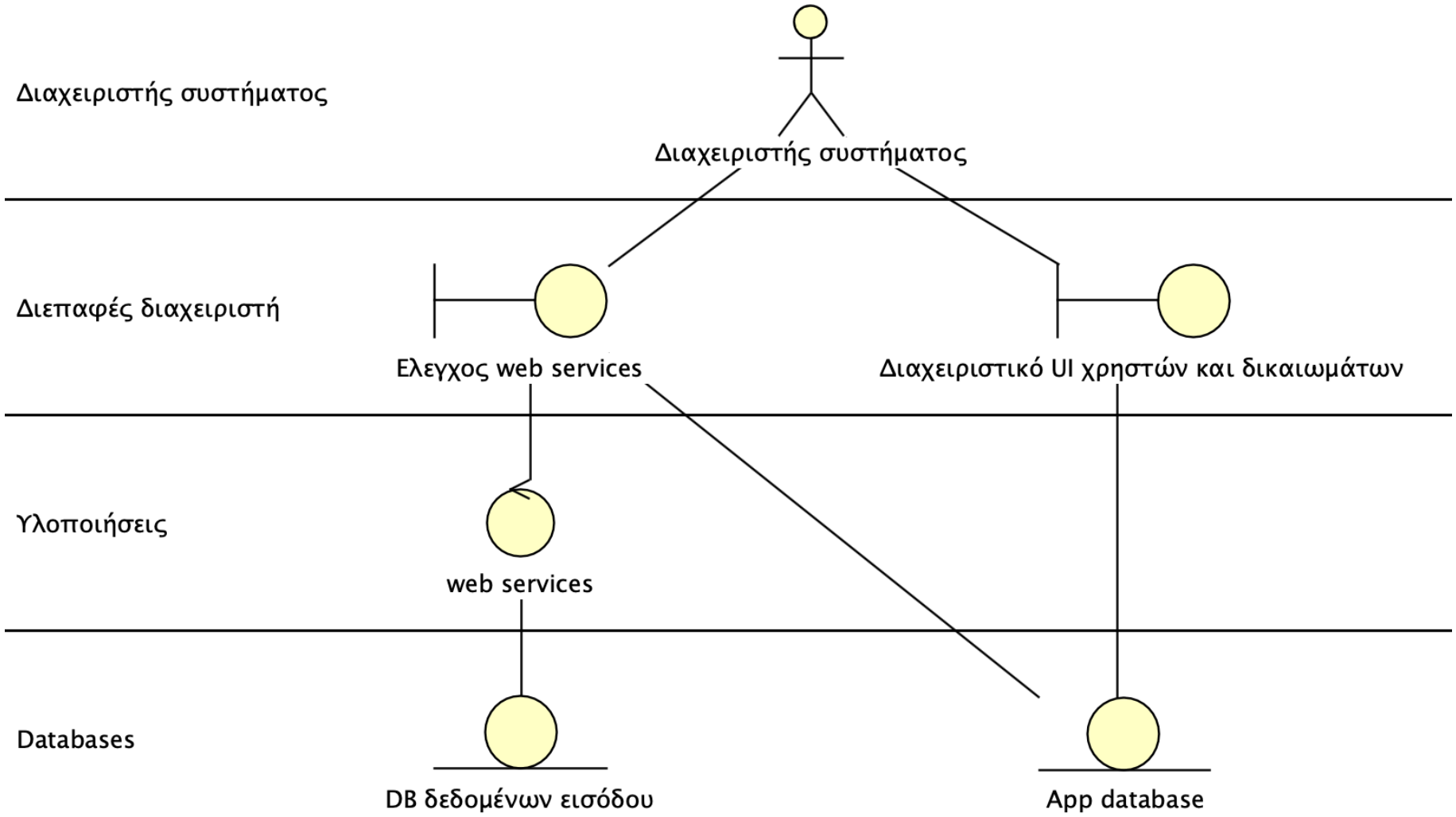
a.k.a. Communication diagrams



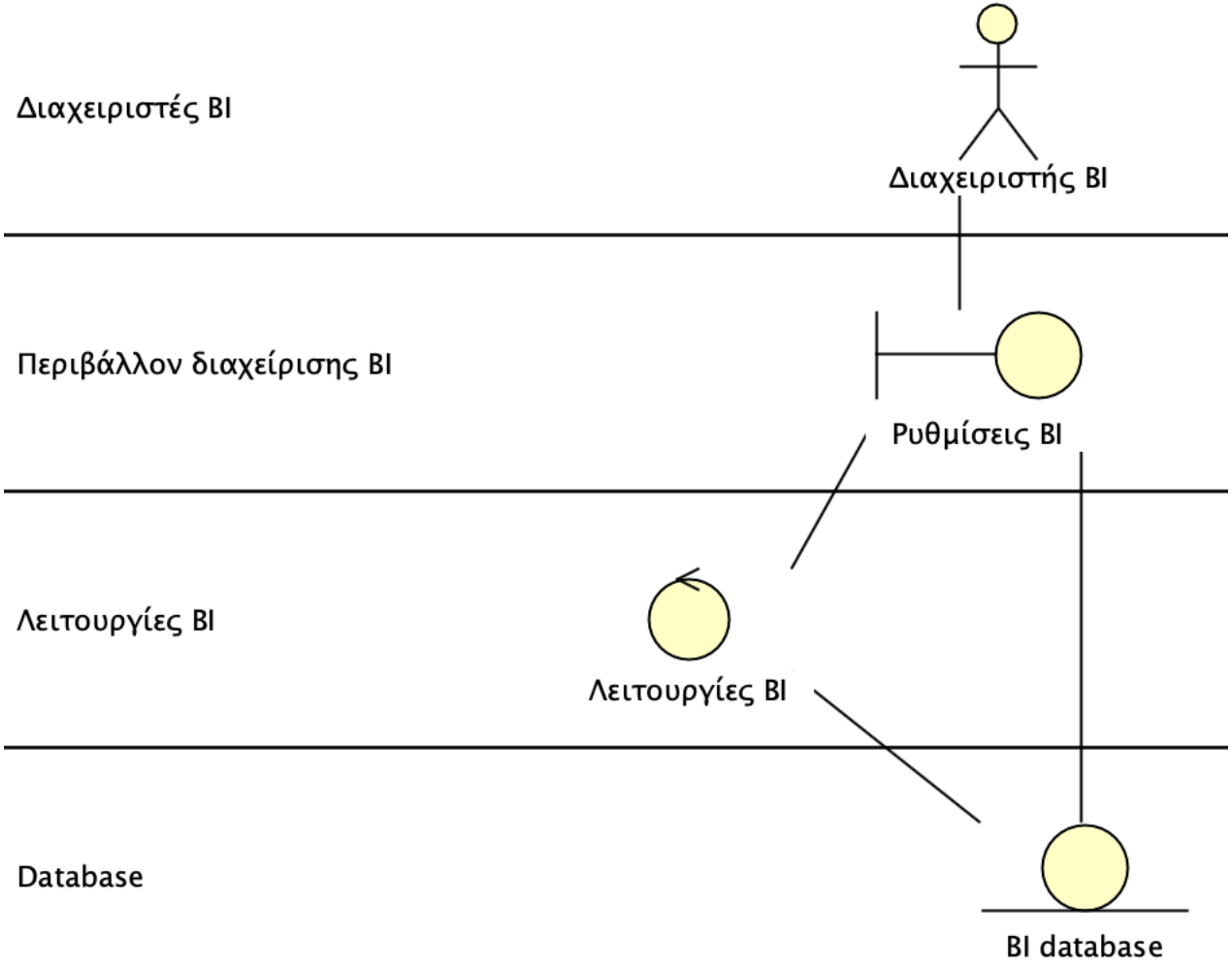
Collaboration diagram example



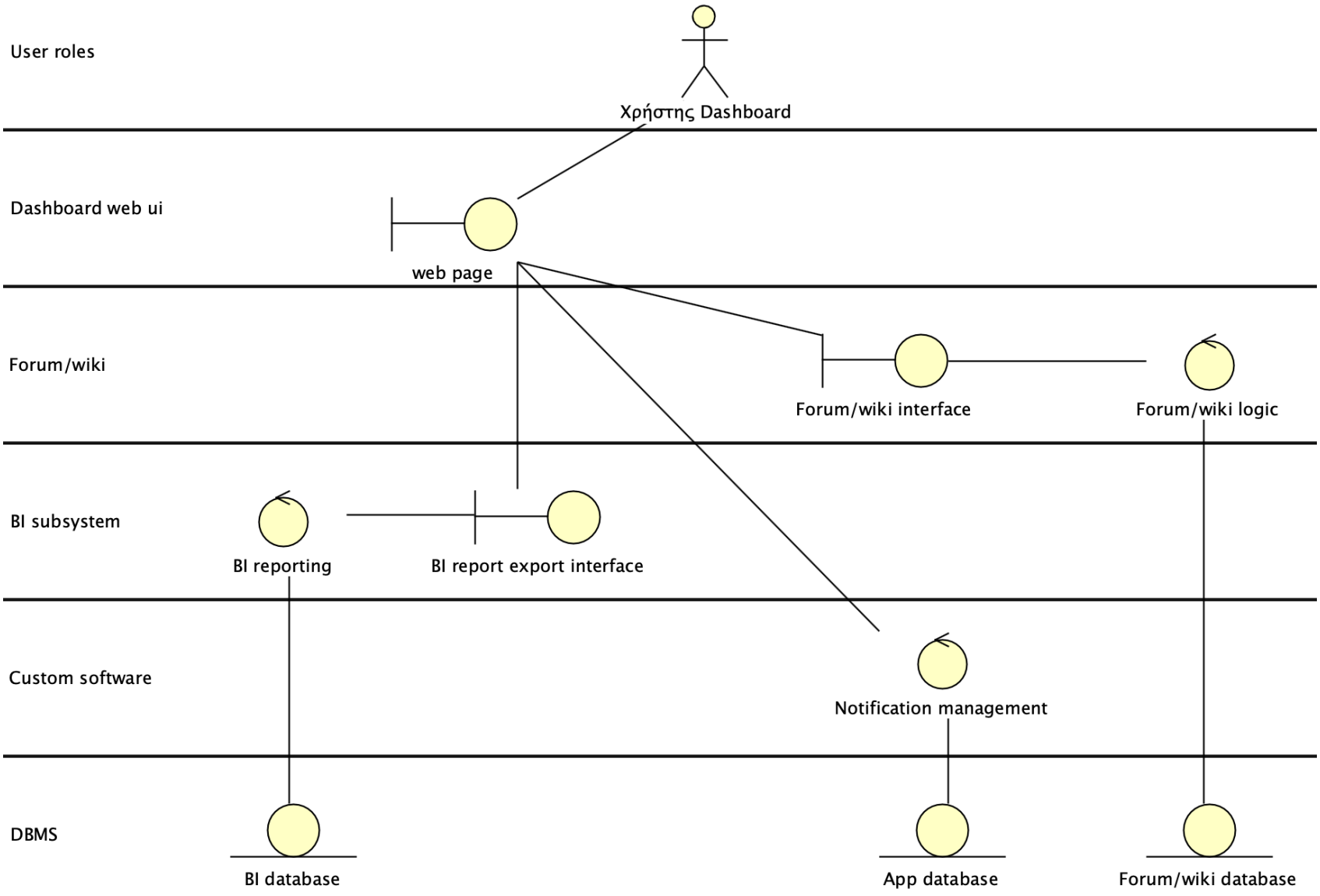
Collaboration diagram example



Collaboration diagram example



Collaboration diagram example



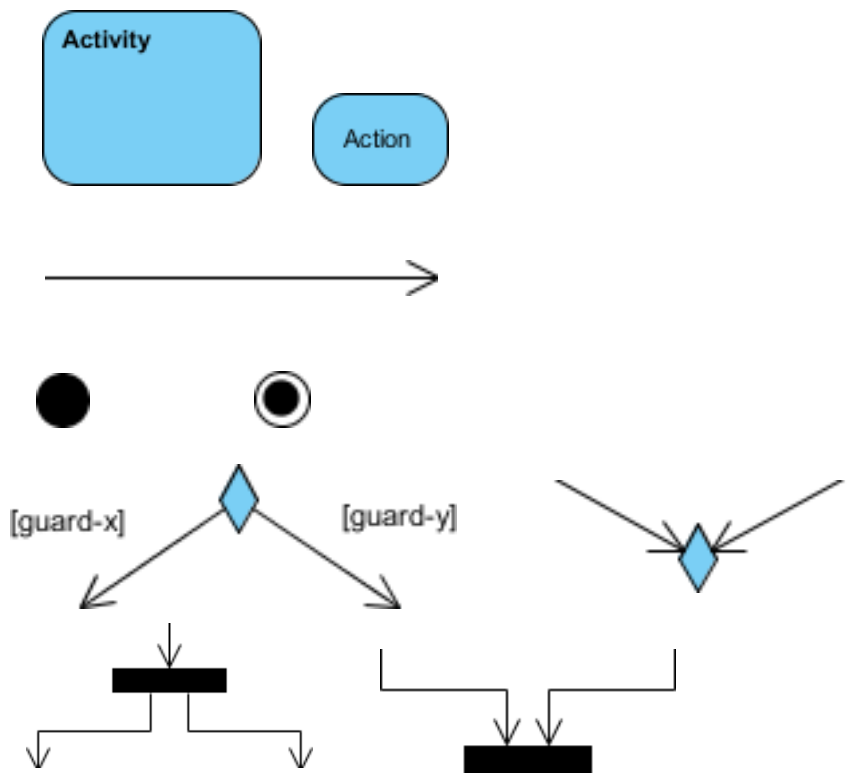
Interactions modeling: Activity diagrams

Modeling of the dynamic aspects of a system, component, etc.

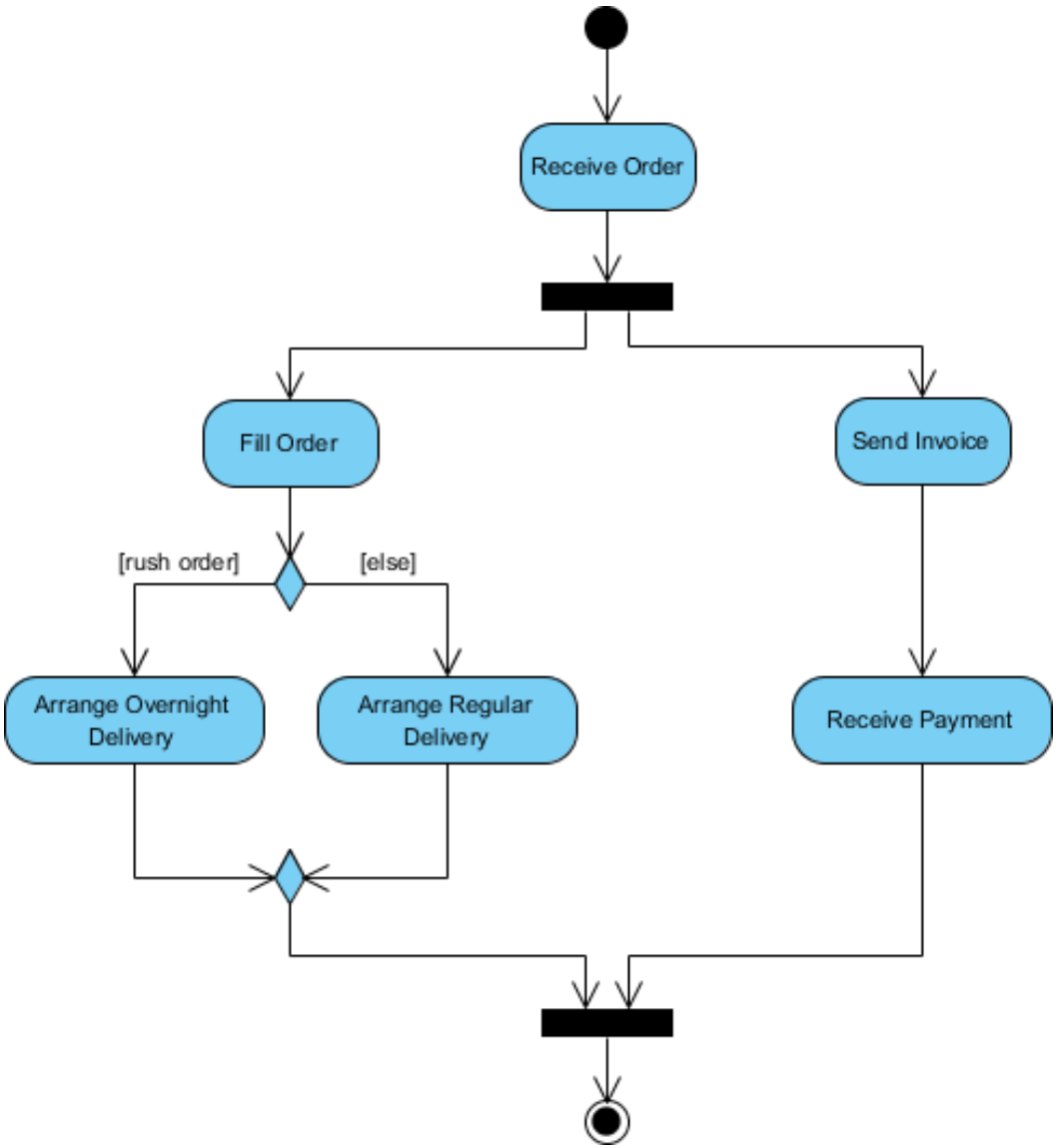
Similar to "old school" graphical representation of algorithms

Concepts

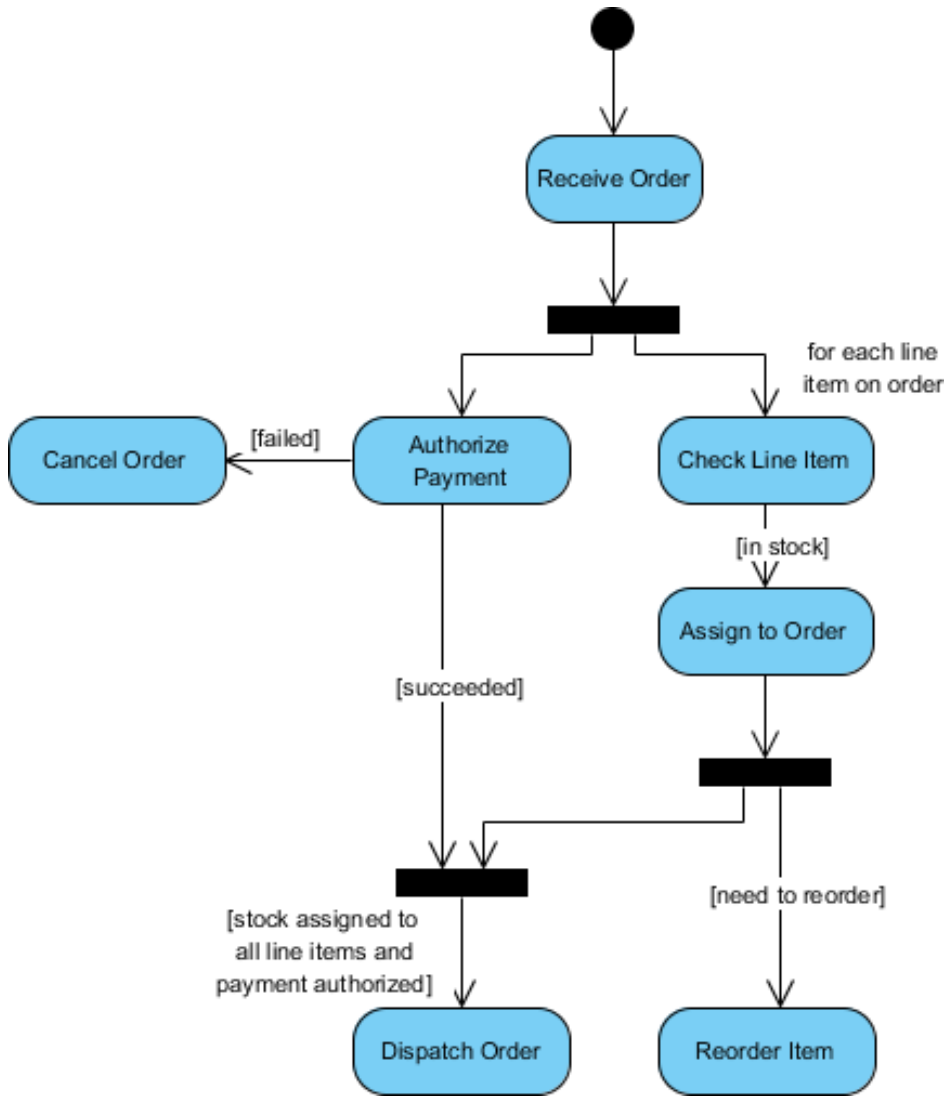
- Activity / action
- Flow (control / object)
- Initial / final node
- Decision / merge
- Fork / join



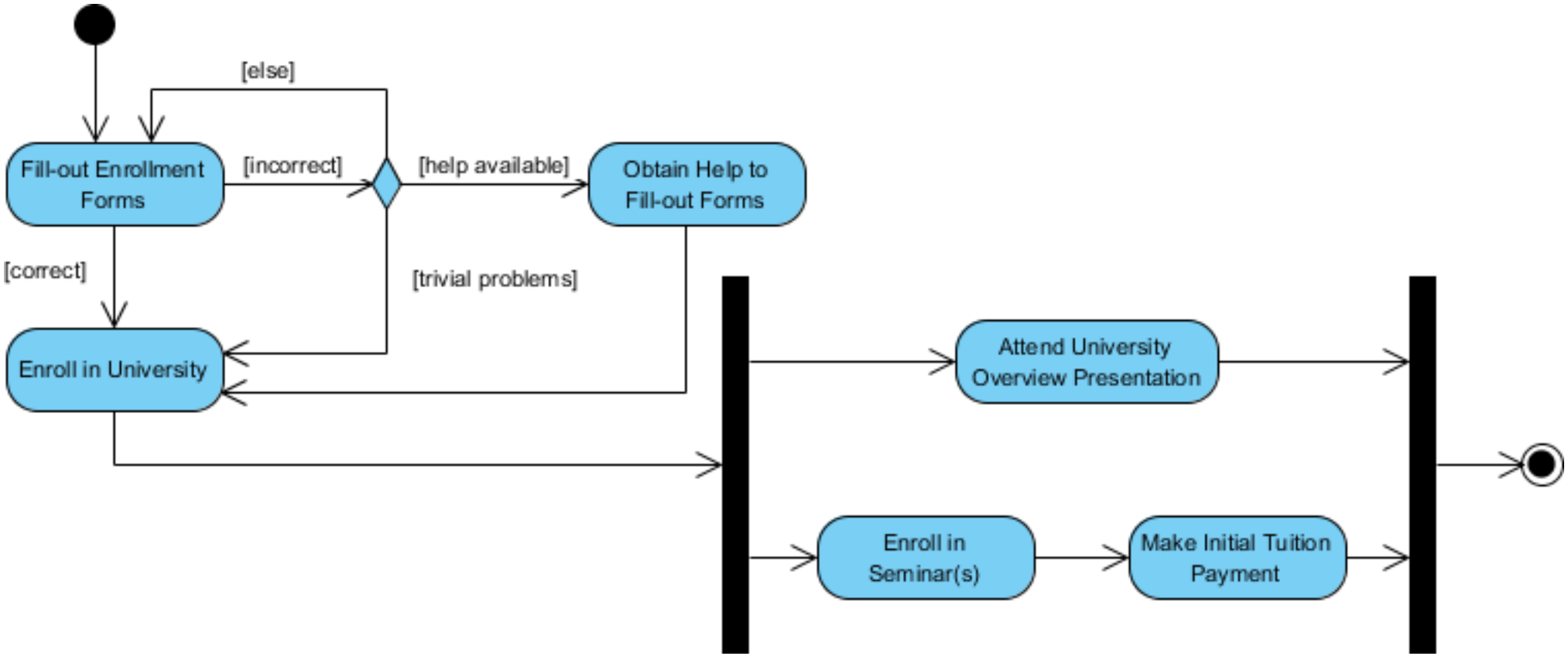
Activity diagram example



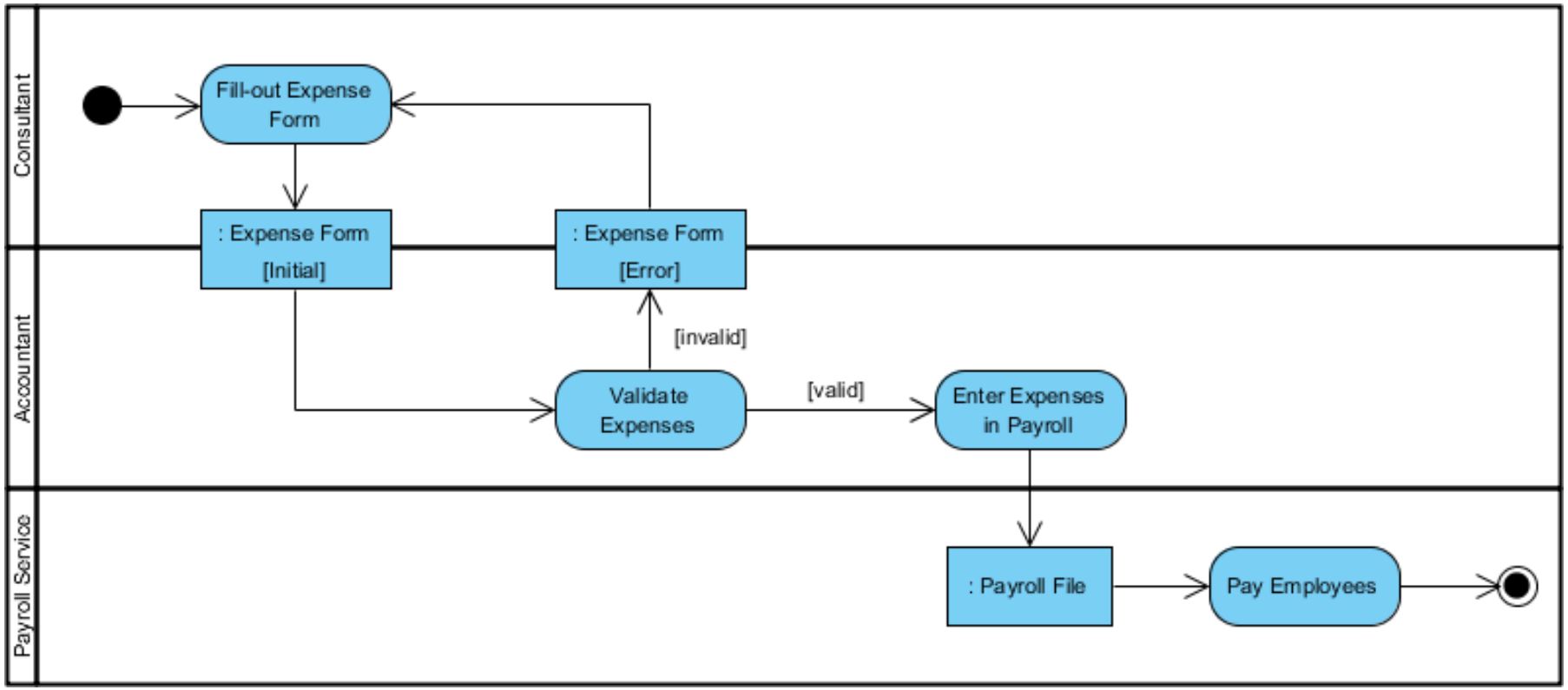
Activity diagram example



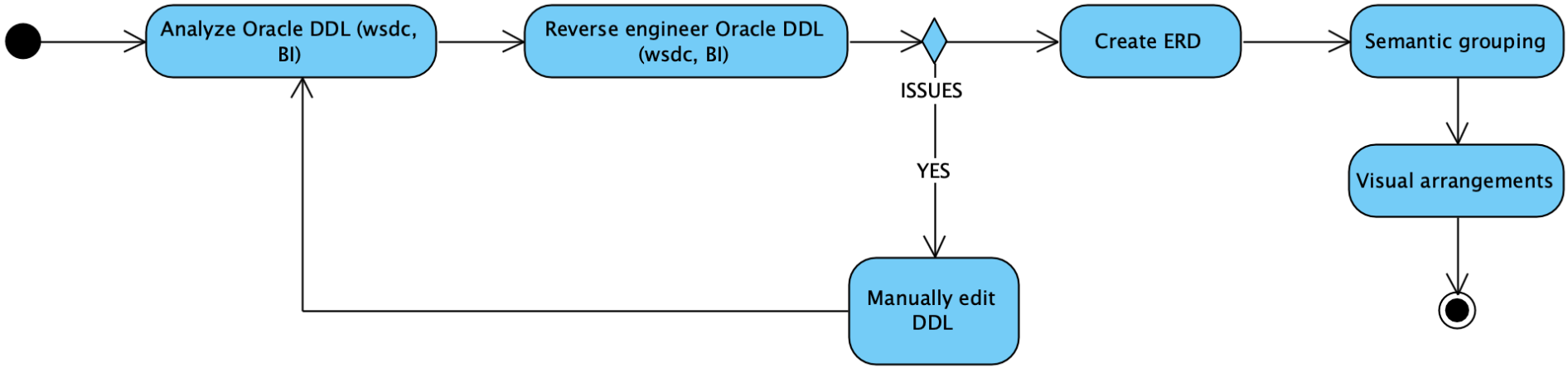
Activity diagram example



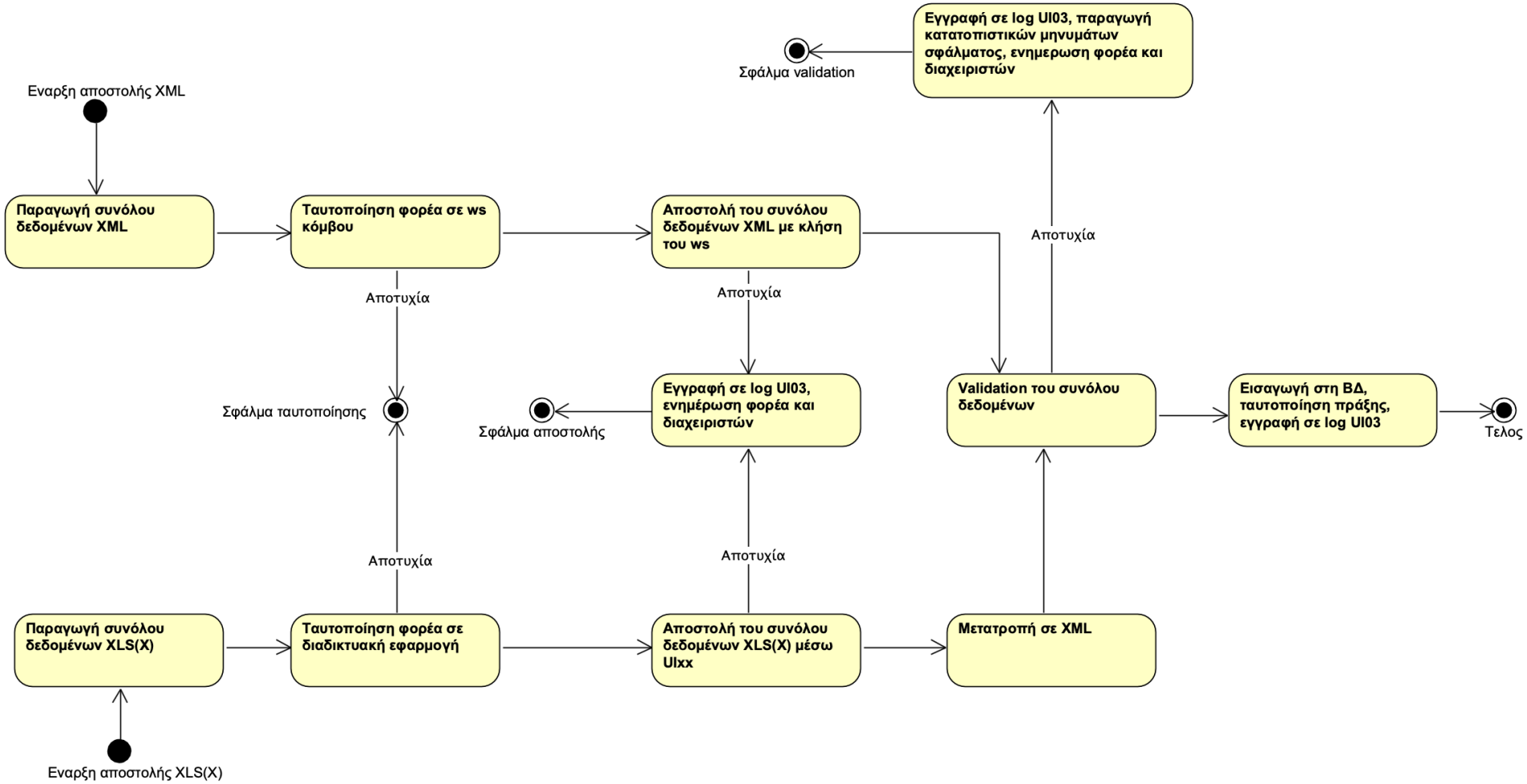
Activity diagram example



Activity diagram example



Activity diagram example

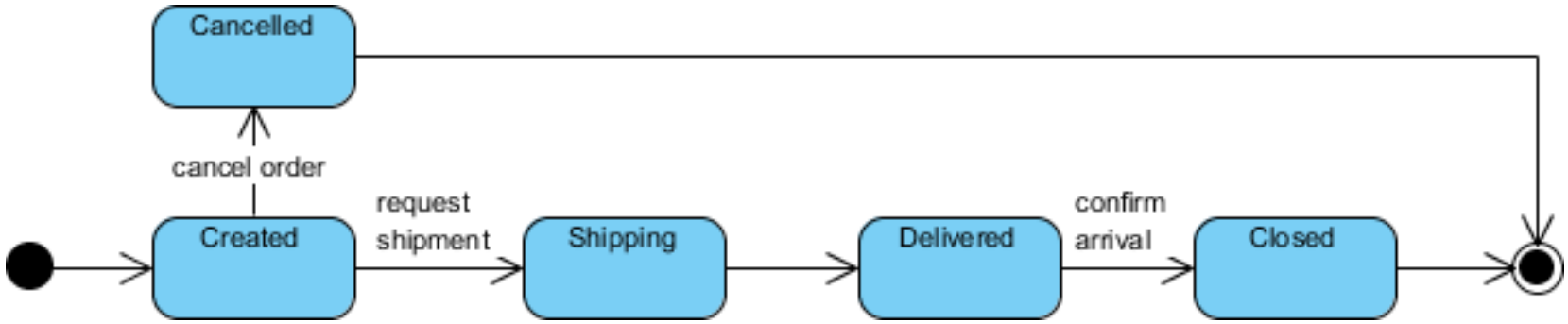


UML State diagrams

Used to model behavior diagram in UML, usually refer to a single class

- to show the lifetime behavior of a single object
- to describe all of the possible states of a particular object or the system

A state is like a 'mode of operation' for objects of a class, which behave differently depending on their state (if applicable)



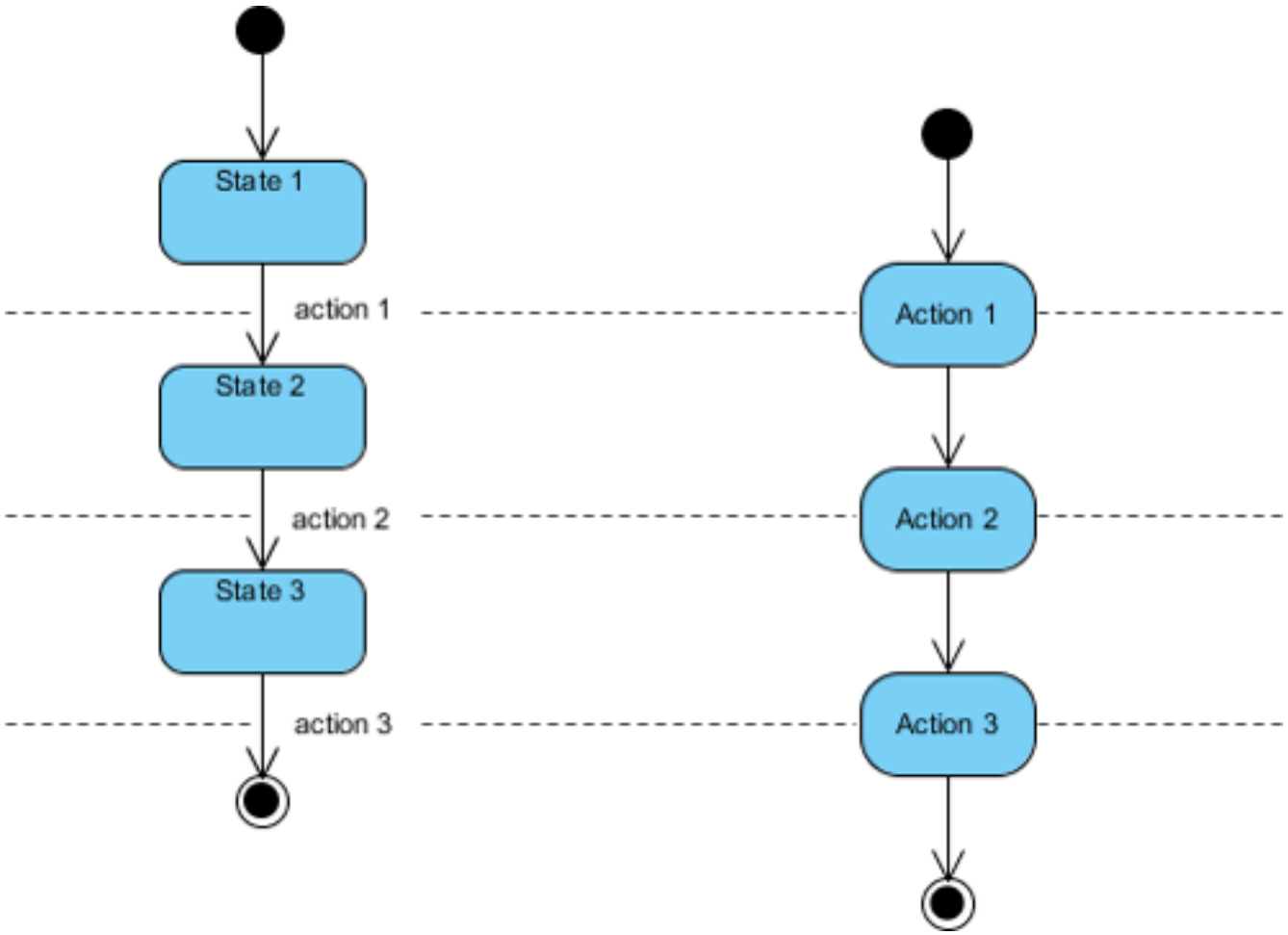
UML State diagrams

Events trigger transitions between states

- Signal event: the arrival of an asynchronous message or signal
- Call event: the arrival of a procedural call to an operation
- Time event: occurs after a specified time has elapsed
- Change event: occurs whenever a specified condition is met

Transitions between states occur as follows:

- An element is in a source state
- An event occurs
- An action is performed
- The element enters a target state


State 1
action 1
State 2
action 2
State 3
action 3
Action 1
Action 2
Action 3

Component diagrams / Package diagrams

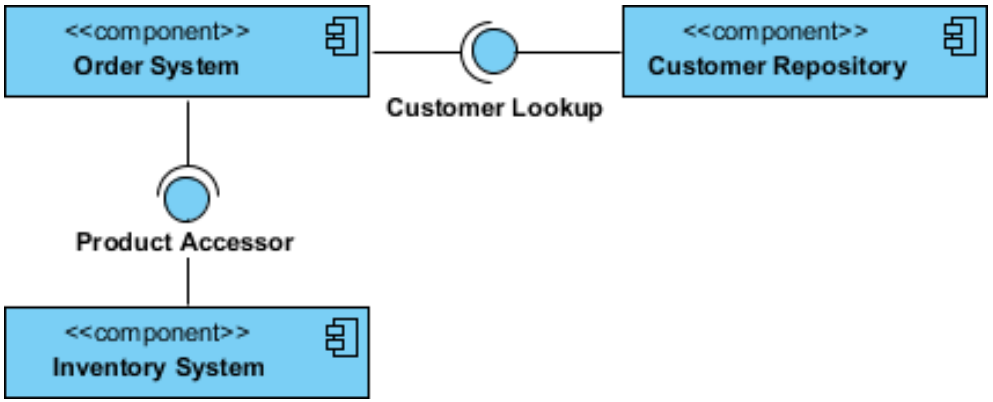
Display components in a system and their dependencies + interfaces

- Explain the structure of a system
- Usually a physical collection of classes

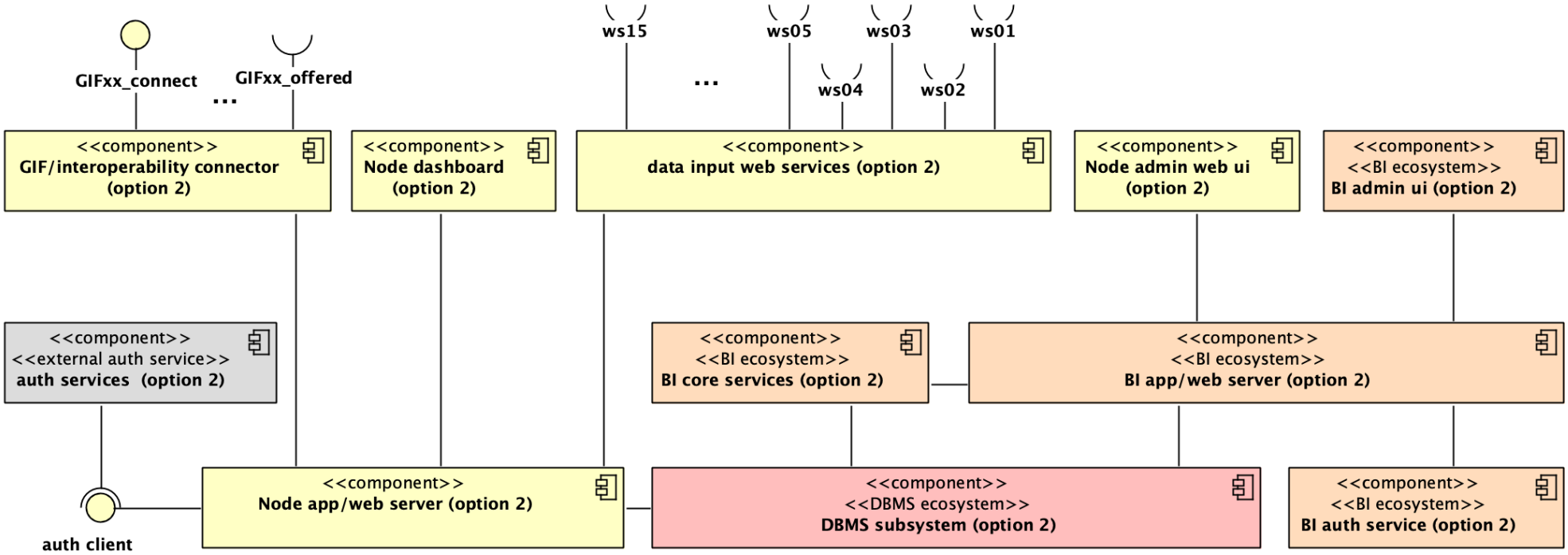
Component vs package Diagrams:

- Component: all of the model elements are private with a public interface
- Package: only display public items

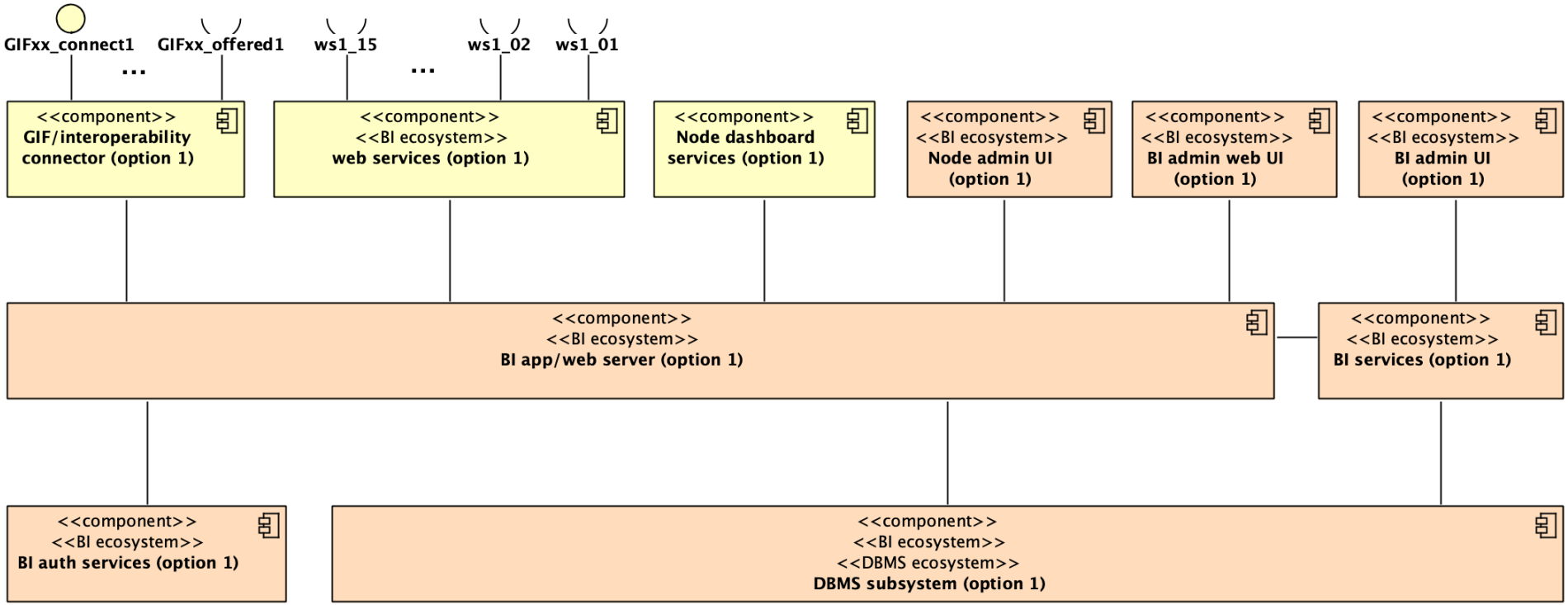
Both are used to group elements into logical structures



Component diagram examples



Component diagram examples



Deployment diagrams

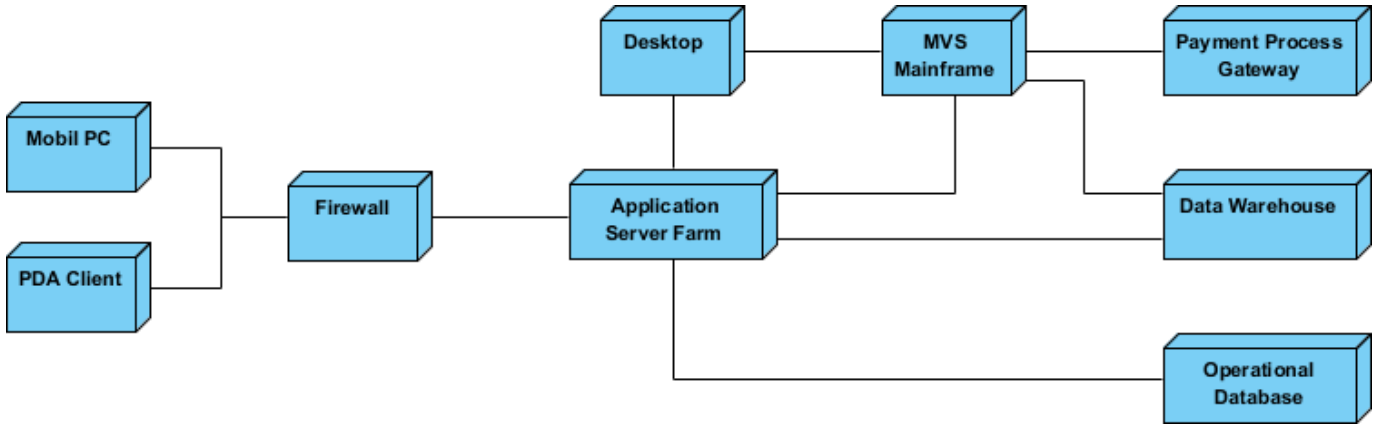
Show the physical architecture of the hardware and software of the deployed system

Nodes

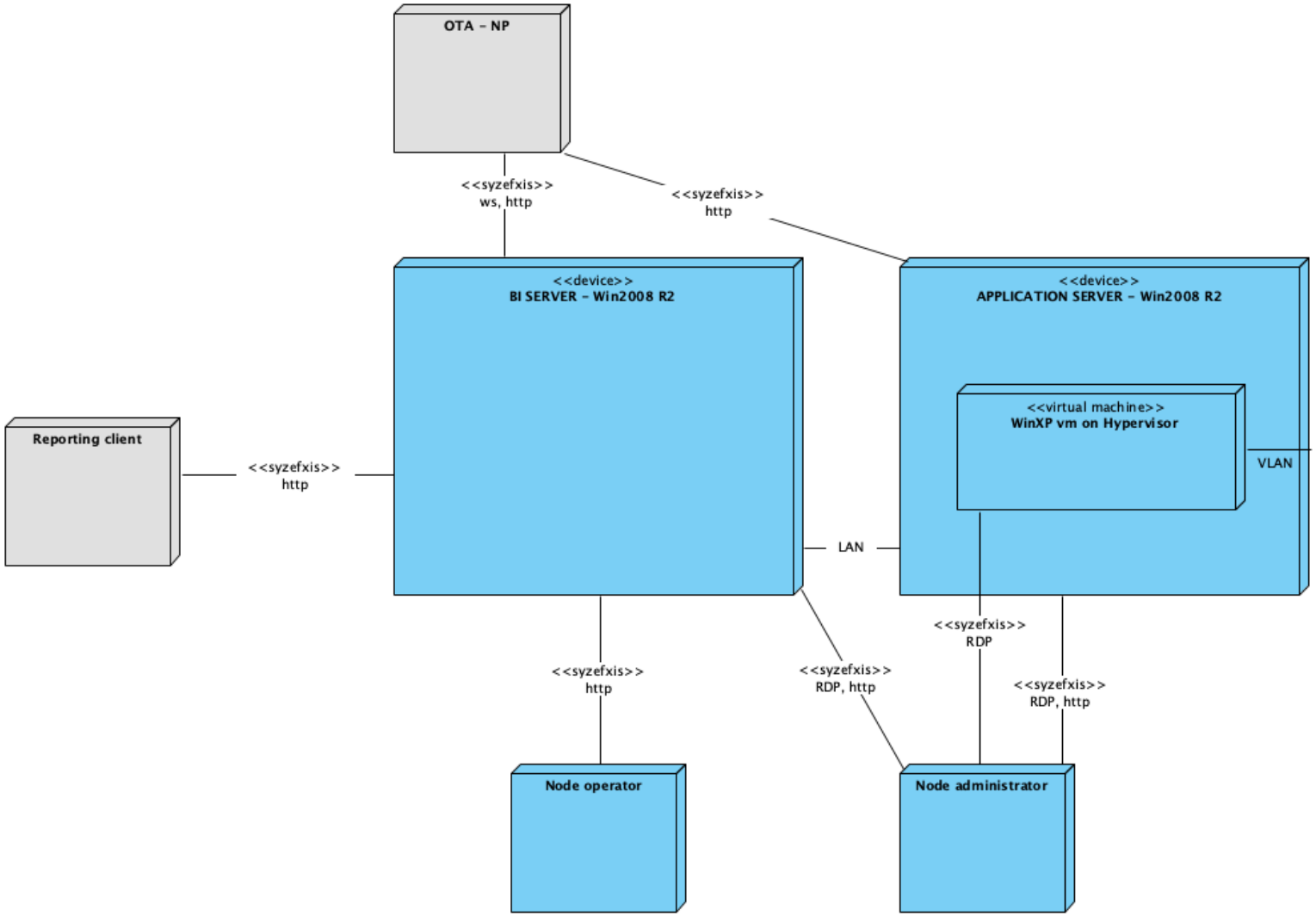
- Typically contain components or packages
- Usually some kind of computational unit; e.g. machine or device (physical or logical)

Physical relationships among software and hardware

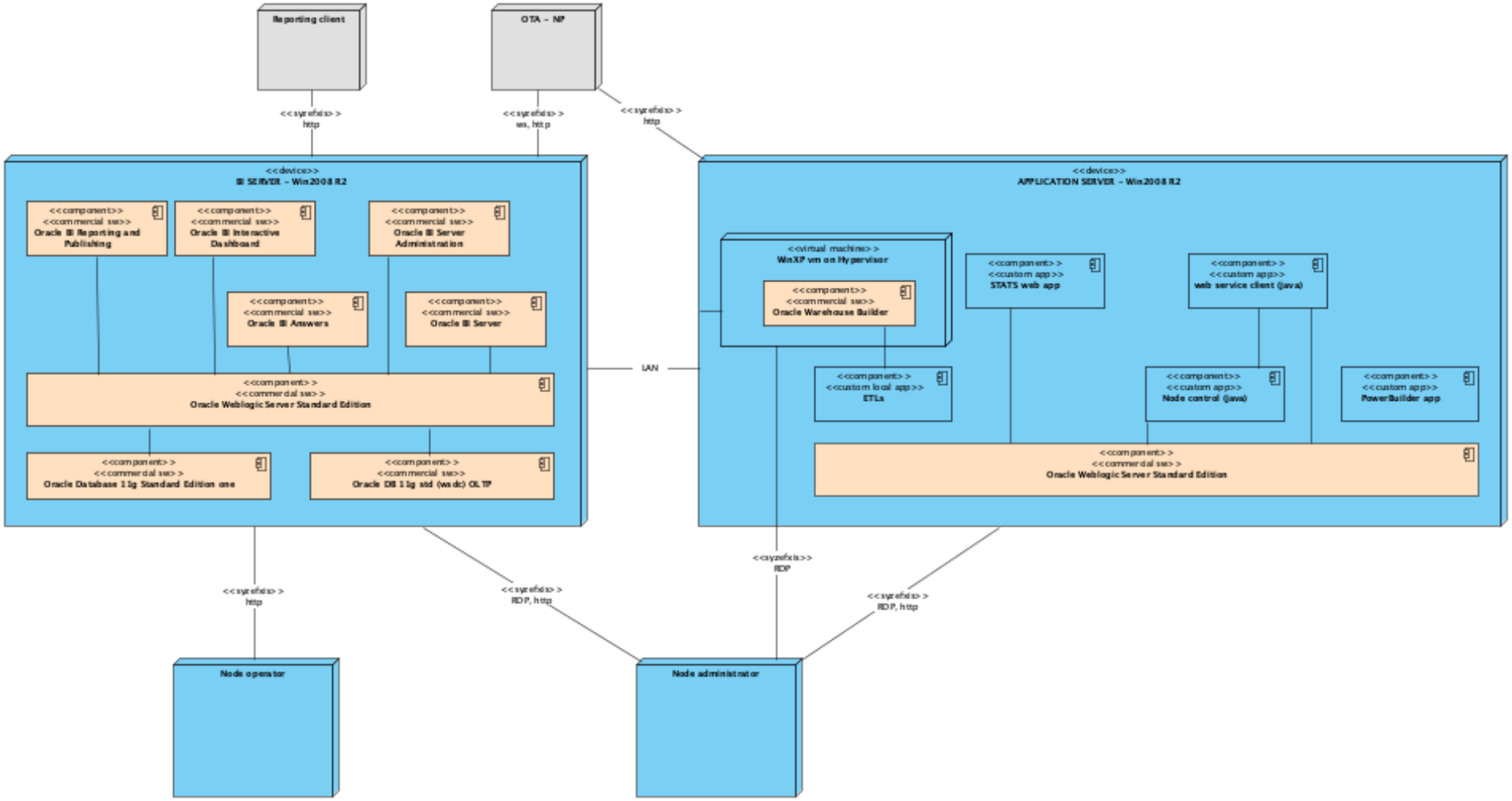
- Explain how a system interacts with the external environment



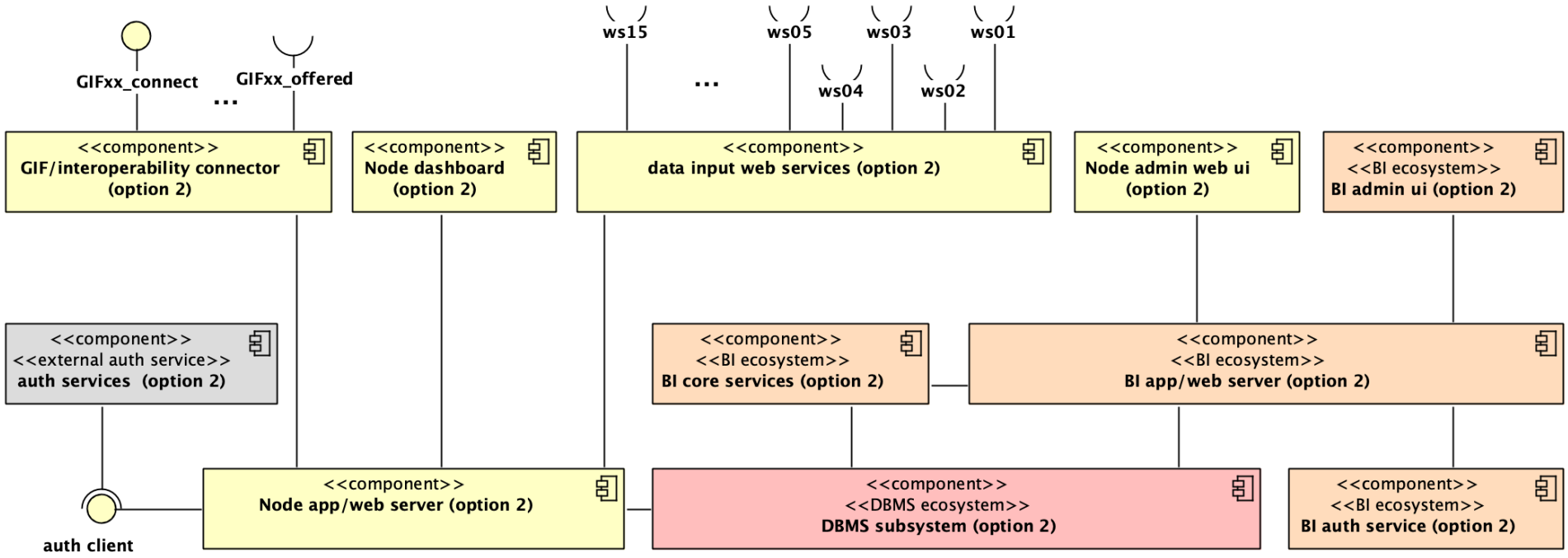
Deployment diagram example 1a



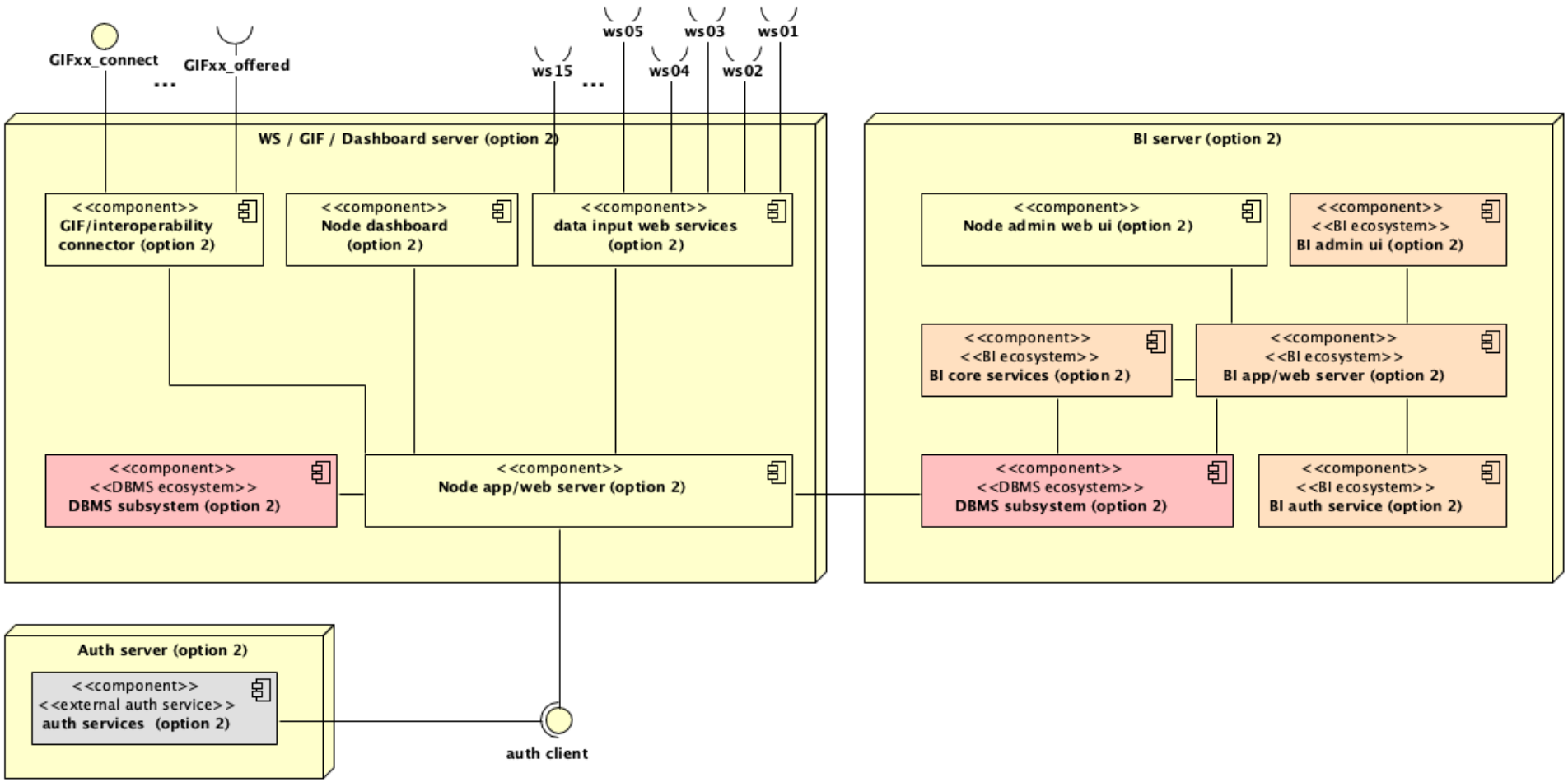
Deployment diagram example 1b



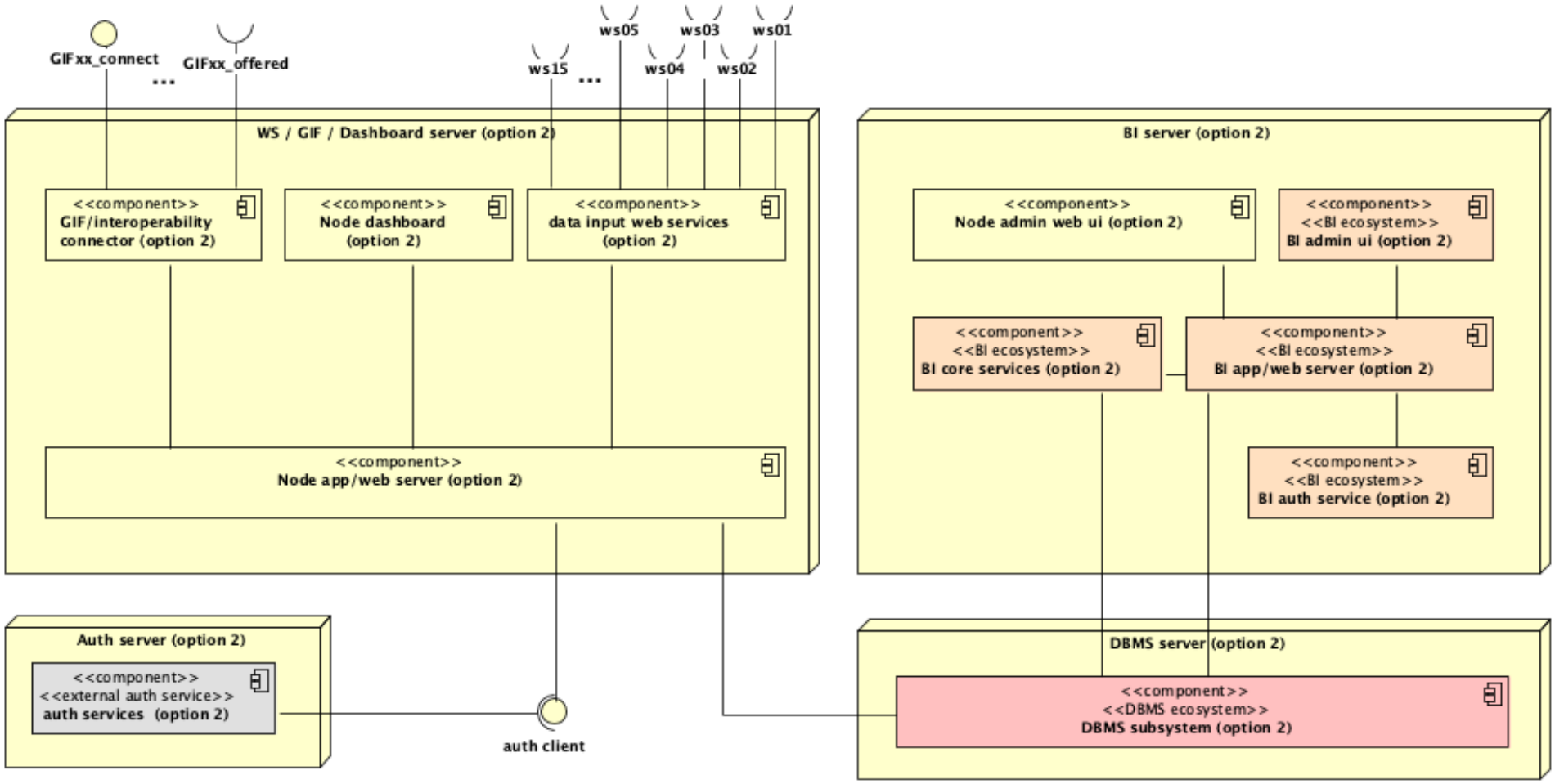
Deployment diagram example 2a

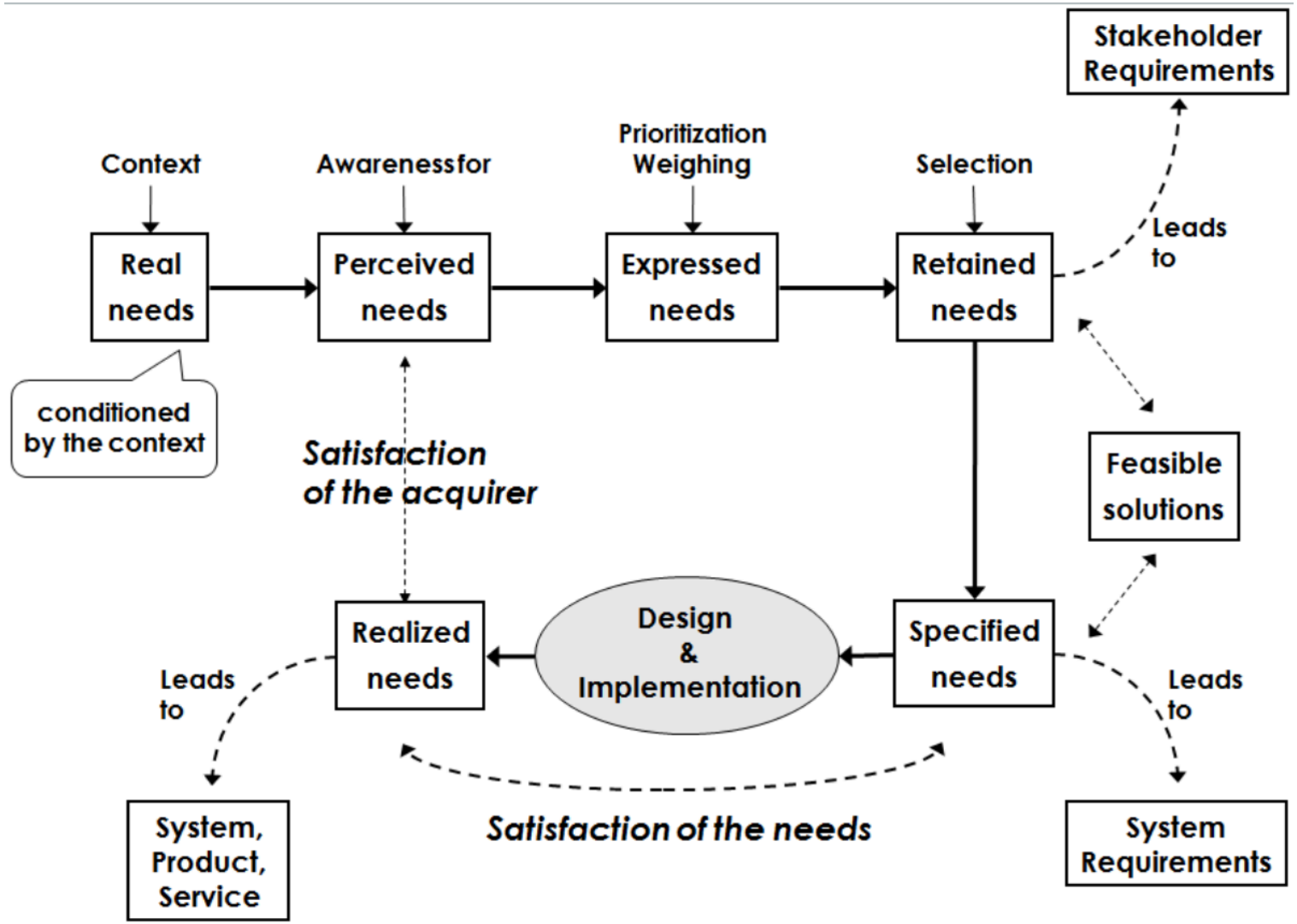


Deployment diagram example 2b



Deployment diagram example 2c




Stakeholder Requirements
Context
Awarenessfor
Prioritization Weighing
Selection
Real needs
Perceived needs
Expressed needs
Retained needs
Leads to
conditioned by the context
Satisfaction of the acquirer
Feasible solutions
Realized needs
Design & Implementation
Specified needs
Leads to
Leads to
System, Product, Service
Satisfaction of the needs
System Requirements