Seminar 1
Design of Information Systems

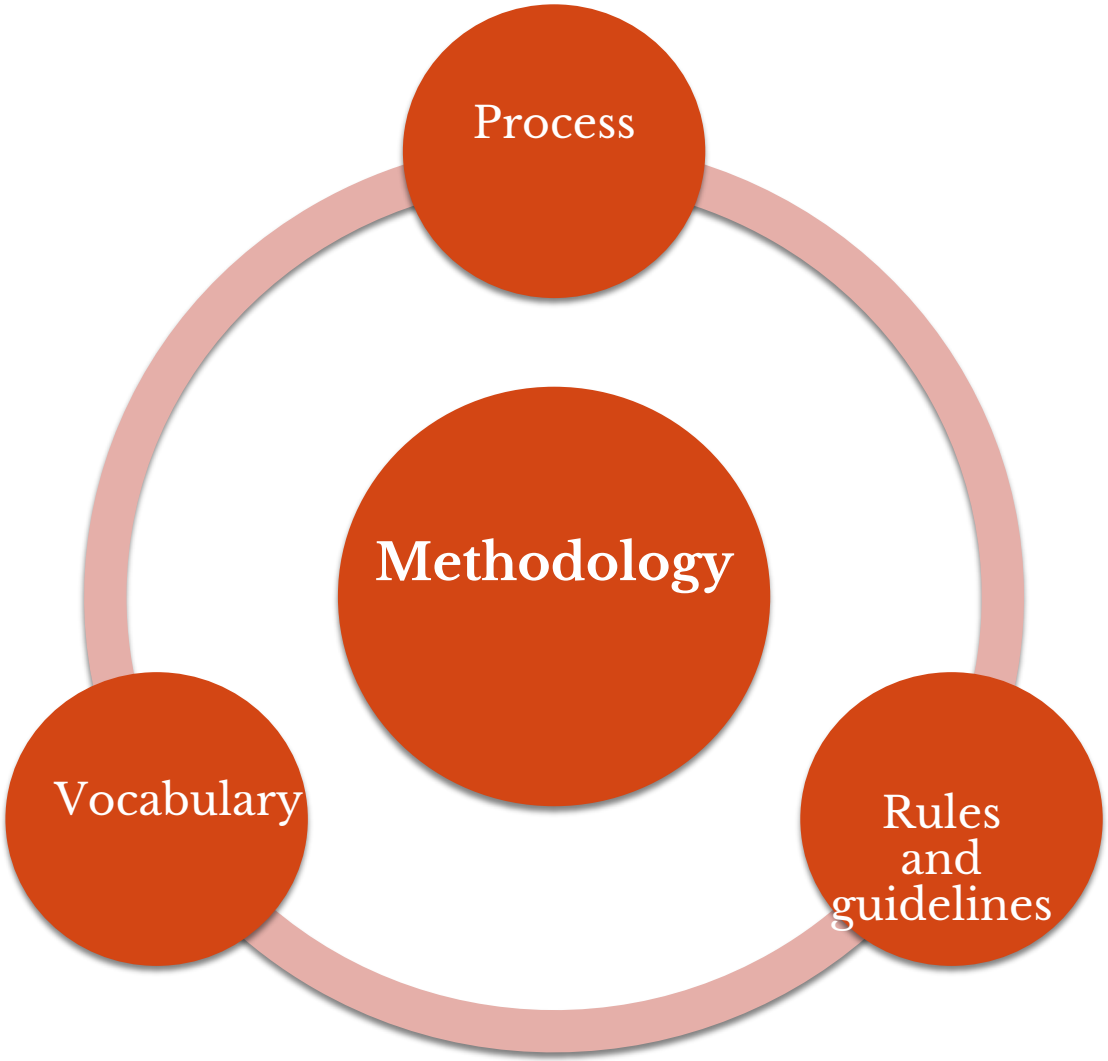
What is a model?

- Simplifications or abstractions of some real elements or elements that are being designed.
- Highlights only those elements that are important for the analyst.
- They are specified using precise graphical or textual notation, using a given symbols language.
- A collection of images and text that has a meaning and is intended to represent something.

What is a model?

- In software development models can be of several types:
  - Environmental models
  - Domain models
  - System specification models
  - System design models
- The models are valuable because:
  - they are fast;
  - it is easier to change a model than source code.

Software development methodologies



Software development methodologies

- *Process: a set of activities that help to achieve the desired objectives;*
- *Vocabulary: describes the process and the results obtained during its implementation;*
- *Rules and guidelines: they define the quality of the process and of the outcomes.*

Software development methodologies

- The part of a methodology that can be standardized: vocabulary (notation).
- UML (Unified Modeling Language) - common notation that can be applied to several methodologies.
- It is very difficult to define a single process suitable for all types of projects

Exemples of standard notations

Methodology examples - 1

- RUP (Rational Unified Process):
  - Iterative and incremental process
  - Partial delivery of product releases at each iteration
  - Short delivery periods and frequent checks
  - Customer verifiable results
  - Exhaustive process; it can become unmanageable
  - RUP customization requires a significant effort
- OMT (Object Management Technique)
  - UML precursor
  - Use the concepts of object orientation

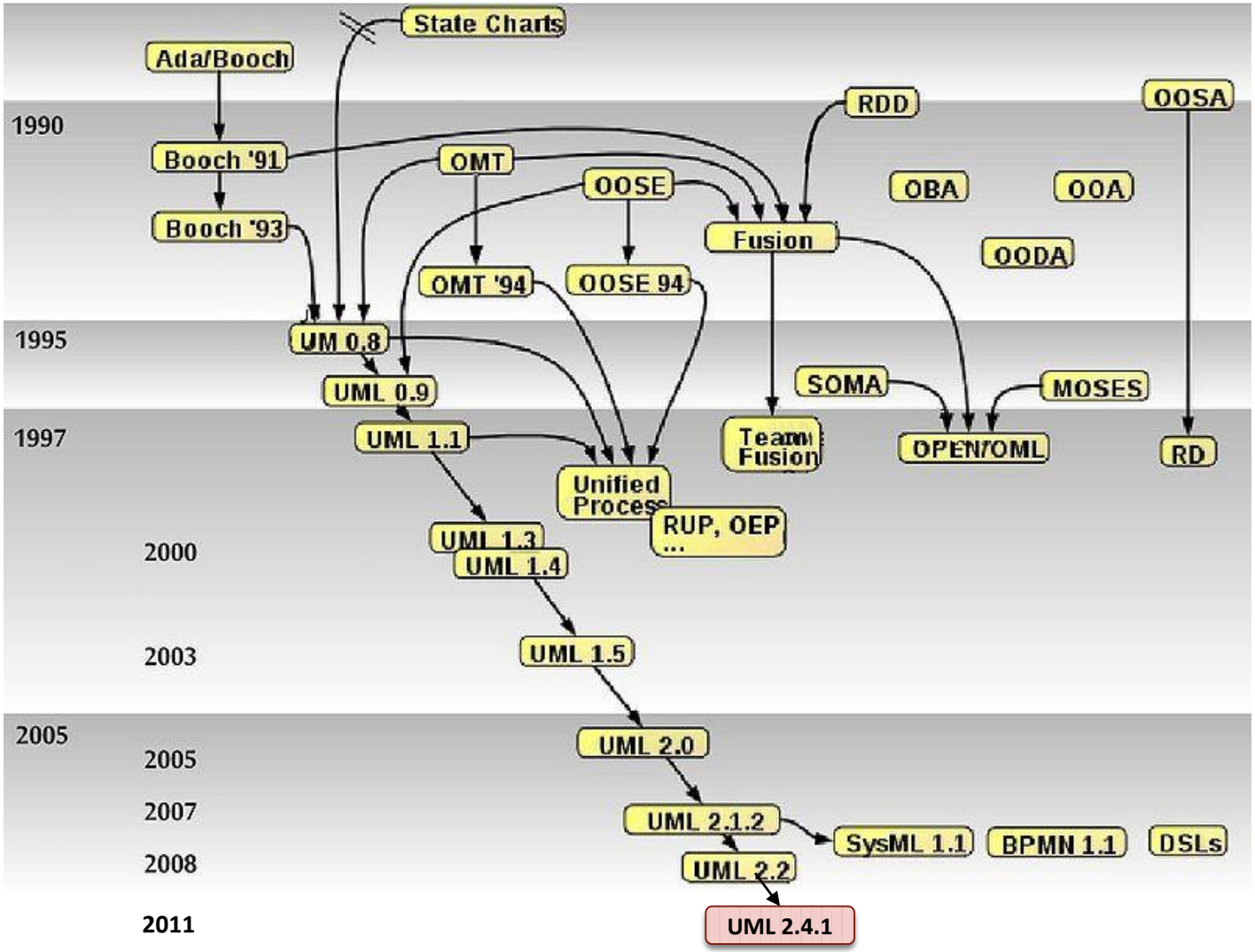
Methodology examples -2

- XP (Extreme Programming)
  - agile development methodology
  - focuses on coding (standards, principles)
  - it argues that programmers work in pairs ("pair programming")
  - numerous discussion sessions during development
  - each iteration (1-4 weeks) has a functional outcome
  - reduced support for modeling
  - close relationship between customers and developers
  - lack of documentation

What is UML?

- **UML = Unified Modeling Language**
- Notation language for specifying, building, visualizing and documenting software systems.
- Combines the best practices in diagram development in the last 50 years.
- Standardizes notation, but does not specify how they are used.
- It is not a methodology, it can be used as a vocabulary for methodologies.
- It offers developers flexibility, while ensuring consistency.
- Is a standard developed and maintained by the Object Management Group.

UML history


State Charts
Ada/Booch
1990
Booch '91
Booch '93
OMT
OOSE
RDD
OOSA
OBA
OOA
Fusion
OODA
OMT '94
OOSE 94
1995
UM 0.8
UML 0.9
SOMA
MOSES
1997
UML 1.1
Team Fusion
OPEN/OML
RD
Unified Process
RUP, OEP ...
2000
UML 1.3
UML 1.4
2003
UML 1.5
2005
2005
UML 2.0
2007
UML 2.1.2
SysML 1.1
BPMN 1.1
DSLs
2008
UML 2.2
2011
UML 2.4.1

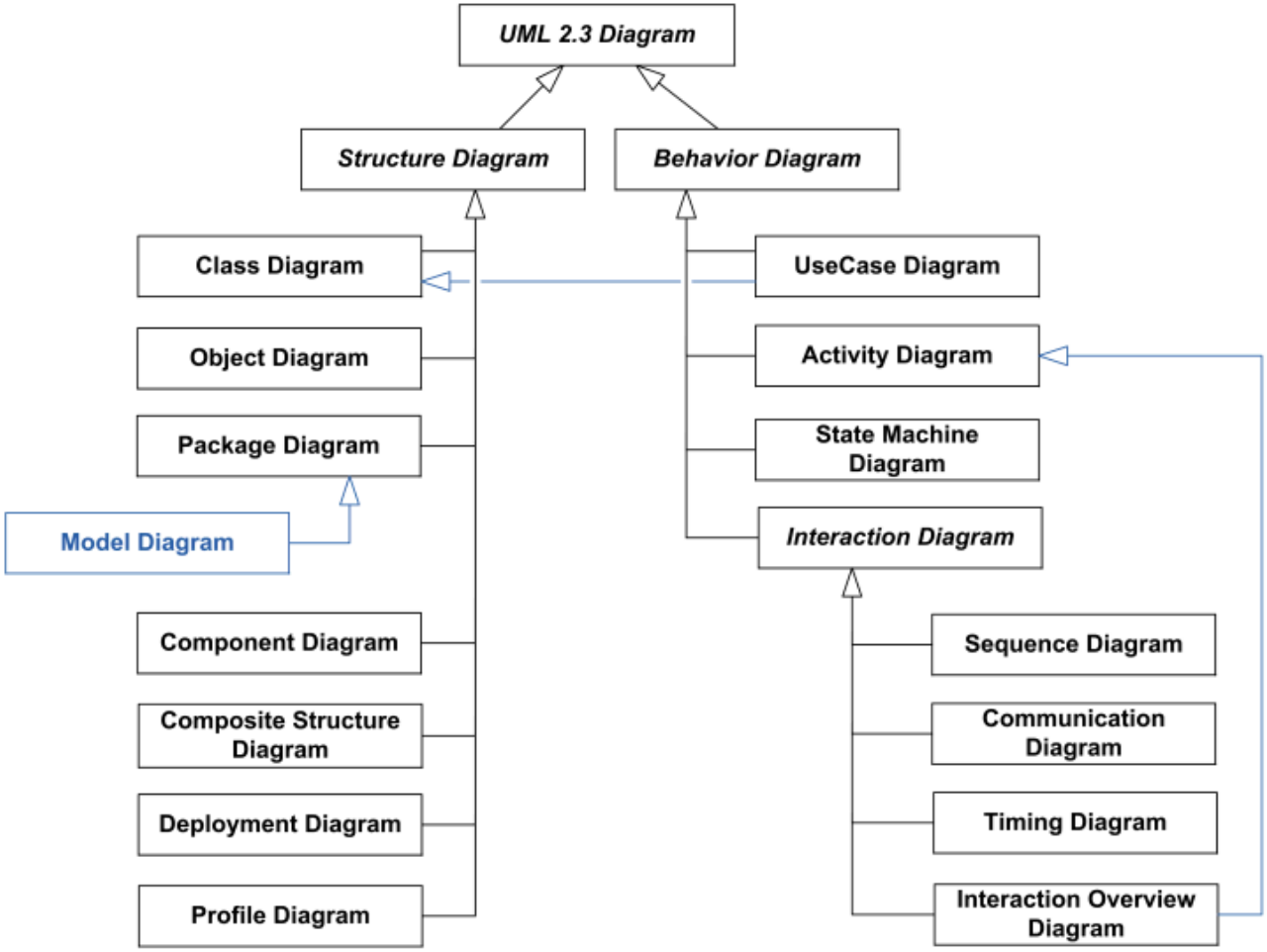
UML basic elements

1. Metamodel for object oriented modeling
   - Consistent set of definitions of **concepts** and the **relationships** between them;
   - Each element used in modeling is defined using a **precise syntax** (eg defining a class);
   - Support language for transmitting visual models between different tools;
   - It has a four-tier architecture.

Layer	Description	Example
meta-metamodel	Defines the language for metamodel specification.	Abstract concepts from which the metamodel is derived.
metamodel	Defines the language for metamodel specification.	Concepts: Class, Atribute, Operation, Component
model	Defines the language used for describing the analysed field.	Concepts: Student, Topic, Client, Product, Order
User objects	Define information about the object of the analysed field.	Exemple: Student #3456, Topic #0512

UML basic elements

2. Diagram types

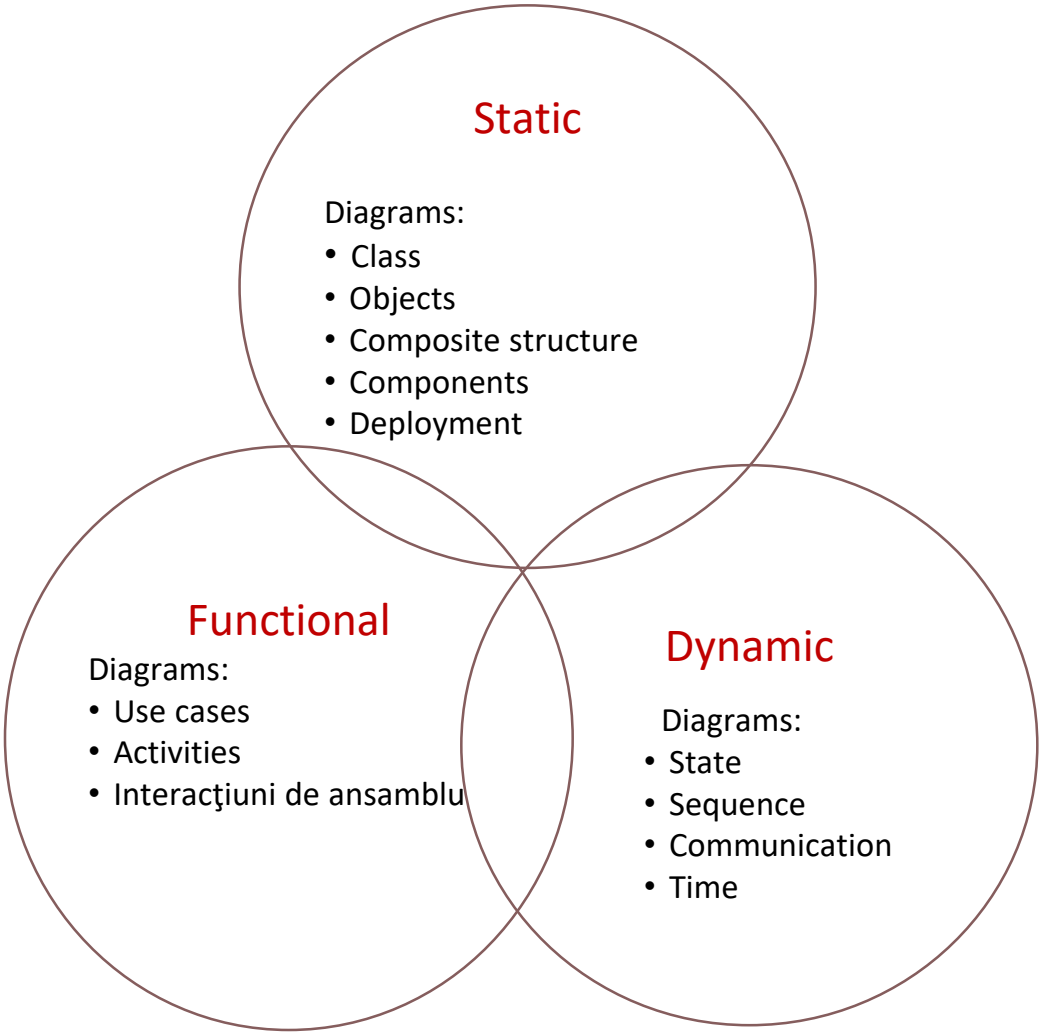


UML basic elements

3. Extension mecanisms

- *Stereotypes* characterizes an element of the model or a relationship between elements (there are predefined stereotypes).
- *Comments* (notes) offer additional descriptions of an element in the model.
- *Constraints* limit the use of an element in the model.
- *Tagged values* represent attributes defined for a stereotype.
- *Profiles* customize the metamodel through special constructions that are specific to a particular field, platform or development method.

Perspectives on the system



Package diagram – structuring / modularization
Profile diagram - language extension

CASE tools - 1

- CASE = Computer Aided Software Engineering

- Necessity:
  - working with visual models can be tedious and time consuming
  - need for IT support when we want to maintain the integrity of the models
  - the ability to generate code

CASE tools -2

Basic functions:

- Creating diagrams
- Managing information regarding diagrams
- Checking the consistency of the models
- Creating links between models
- Version tracking models
- Code generation
- Engineering and reverse engineering

Proiectarea sistemelor informatice

Seminar 2

Agenda

01

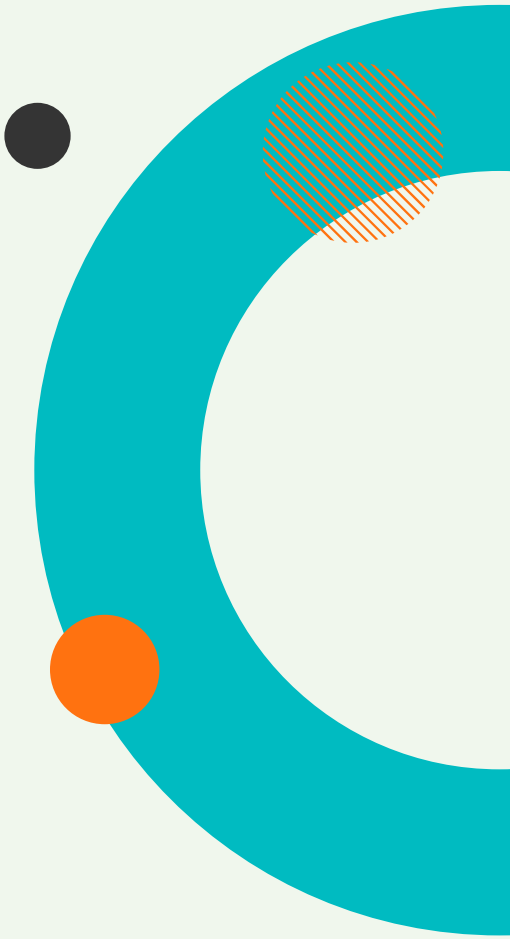
Requirement identification

02

Analysis of textual requirements

03

Structuring requirements - decision tables



Analysis of textual requirements -1

With demos in Visual Paradigm

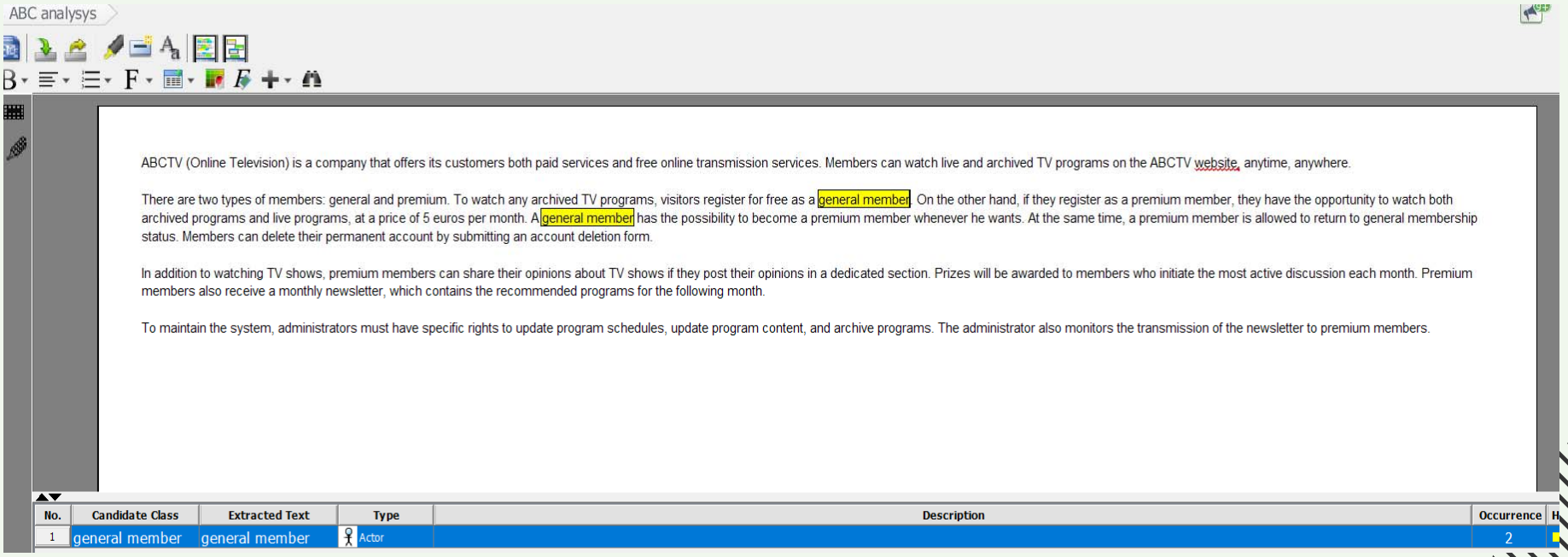
ABCTV is a company that offers online TV broadcasting services. It intends to improve its services by restructuring its IT system.

We will use **textual analysis** to identify the main use cases of the system and to create a use case diagram.

1. Download the **ABCTV-EN.txt** file from the seminar materials. It contains an overview of the main requirements of the system.
2. Open **Visual Paradigm**
3. Create a new project by selecting **Project> New** from the application menu bar. In the New Project window, enter ABCTV as the project name, and then select Create Blank Project.
4. Select **Diagram> New** from the menu bar.
5. In the New Diagram window, select **Textual Analysis** and click **Next**.
6. Press **OK** to create a description of the system requirements.
7. Requirements can be entered directly through an editor or imported from an existing file. In this case we will import the requirements by pressing the **Import File ()** button, selecting the **ABCTV-EN.txt** file and then **Open**.
8. The text has been imported. Study the description of the requirements.

Analysis of textual requirements - 2

9. Identify all the actors involved in using the system. Actors (concept of the UML language use case diagram) represent different categories of system users or other systems with which it interacts to implement functionalities. Going through the description of the requirements, we can identify the first candidate actor, **general member**, in the second paragraph. Select the text "***general member***", right-click and select from the **Add text** as list the ***Actor*** option from the menu that appears.



Analysis of textual requirements - 3

10. Check the grid at the bottom of the window. It lists the properties of the extracted candidate object: **Candidate Class** (name of the candidate object), **Extracted Text** (word / phrase from which it was extracted), **Type** (candidate type), **Description**, **Occurrence** (how many times the word / phrase appears in the text requirements), and **Highlight** (highlight color of the candidate object).

11. Similarly, identify the other two candidate actors, “***premium member***” and “***administrators***”. Change the name of the last candidate actor to "administrator".

12. Identify candidate **use cases**. Use cases describe how users use the system. Usually, candidate use cases are named by means of a verb or a verbal expression (for example, verb + noun). Select the text "***register for free as a general member***", right-click and select the **Use Case** option. Change the name (Candidate Class field) to "***register as a general member***".

Analysis of textual requirements - 4

13. Similarly, highlight the following text sequences to identify them as candidate use cases:

- watch live and archived TV programs
- watch any archived TV programs
- register as a premium member
- watch both archived programs and live programs
- return to general membership status
- delete their permanent account
- share their opinions
- receive a monthly newsletter
- update program grid
- update program content
- archive programs
- monitor the transmission of the newsletter

Analysis of textual requirements - 5

14. To distinguish between candidate objects in the requirements description, the background color can be changed. We will change the background color for the actors to orange.

No.	Candidate Class	Extracted Text	Type	Description	Occurrence	Highlig...
9	Administrator	administrators	Actor	The administrator has the role to: update the p	1	
10	Premium member	premium members	Actor	An administrative fee is charged for deleting th	3	
11	Request account removal	delete their permanent account	Use Case	An administrative fee is charged for deleting th	0	
12	Post an opinion	share their opinions	Use Case	Premium members can post their opinions abo	1	
13	Receive newsletter	receive a monthly newsletter	Use Case	Premium members can opt to receive a monthl	1	
14	Update program schedule	update program schedules	Use Case	The administrator updates the program schedu	1	
15	Update program content	update program content	Use Case	The administrator updates the content of a pro	0	

15. If necessary, we change the names of the actors and use cases so that they are short, informative and intuitive. The table below specifies the names of the candidate classes, after refining the extracted text, together with a brief description of them. Rename the candidate classes and complete the descriptions according to the following table.

Candidate classes table- 1

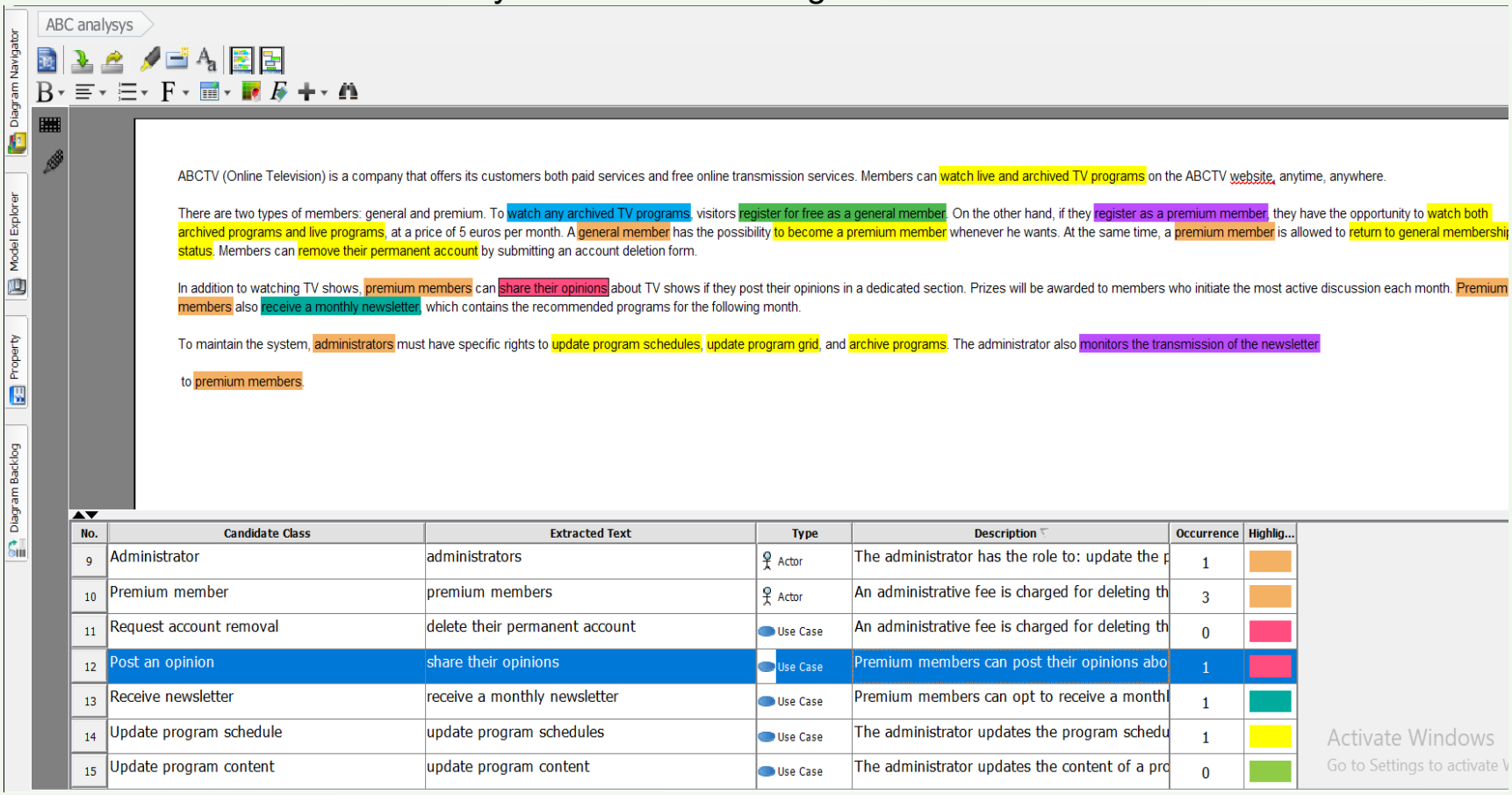
Candidate class	Extracted text	Description
General member	general member	A general member can only view archived TV shows. In order to watch live TV programs, he must change his status to become a premium member.
Premium member	Premium member	A premium member can watch both archived programs and live programs, at a price of 5 euros per month.
Administrator	administrator	The administrator has the role to: update the program grid, update the content of a program and archive the programs. It also monitors the delivery of the newsletter to premium members.
Watch archived TV programs	watch any archived TV programs	Members can always watch archived TV programs.
Register as a general member	register as a general member	Registration as a general member is free.
Register as a premium member	register as a premium member	Premium members pay a fee of 5 Euro / month.
Watch live programs	watch both archived programs and live programs	Only premium members are allowed to watch live programs.
Upgrade membership	to become a premium member	A general member can upgrade their premium membership status.

Candidate classes table- - 2

Clasă candidată	Text extras	Descriere
Return to general membership status	return to general membership status	A premium member can return to general membership status.
Request account removal	remove their permanent account	An administrative fee is charged for deleting the premium members' account.
Post an opinion	share their opinions	Premium members can post their opinions about TV programs on the official ABCTV website.
Receive newsletter	receive a monthly newsletter	Premium members can choose to receive a monthly newsletter.
Update program schedule	update program schedule	The administrator updates the program schedules every two weeks.
Update program content	update program content	The administrator updates the content of a program weekly.
Archive programs	archive programs	The administrator archives TV programs weekly.
Monitors the transmission of the newsletter	monitors the transmission of the newsletter	The administrator is responsible for monitoring the delivery of the newsletter.

Analysis of the textual requirements - 6

16. The results of the textual analysis in Visual Paradigm:



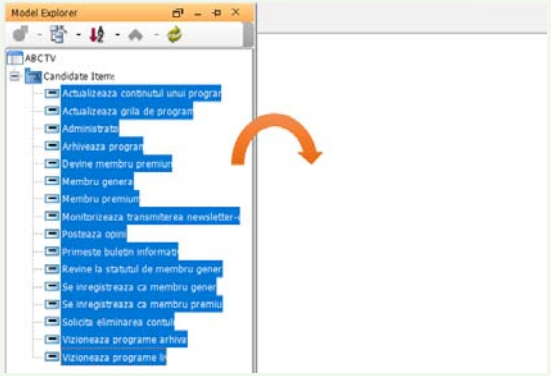
Analysis of the textual requirements - 8

17. Change the view pane by pressing the **Candidate Pane View** button. Candidate objects can be viewed in this panel for easy arrangement.

18. We will create a use case diagram based on these candidate objects. Select **Diagram**> **New** from the menu bar. In the **New Diagram** window, select **Use Case Diagram** and then **Next**. Click **OK** to create the chart.

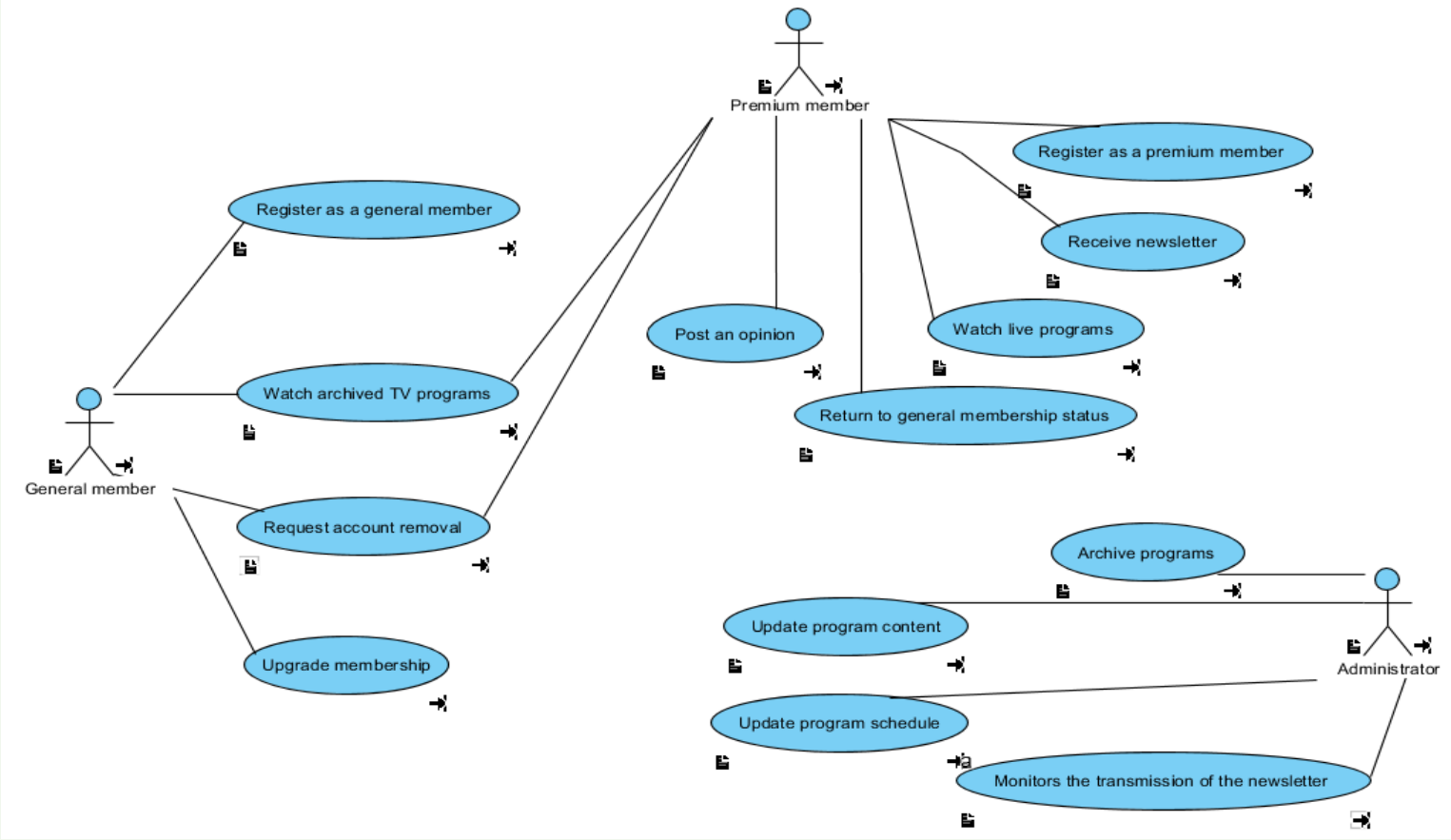
19. Open the **Model Explorer** tab on the right side of the window or choose **View**> **Panes**> **Model Explorer** in the menu bar.

20. Select all ***actors*** and ***use cases*** and, with drag & drop, place them on the empty surface on the right.



Analysis of the textual requirements - 9

21. Rearrange the objects on the chart and associate the actors with the use cases with simple lines (Association on the toolbar on the left).



Seminar 3
Design of Information Systems

UML Use case diagram

UML diagram types

Diagram name	Description	Specific stages
Structural diagrams		
Class diagram	Illustrates the relationships between classes modeled within the system.	Analysis, Design
Object diagram	Illustrates the relationships between objects modeled in the system, when instances of identified classes better communicate the constructed model.	Analysis, Design
Package diagram	Group together other elements of UML to form high-level constructs.	Analysis, Design, Implementation
Deployment diagram	Describes the physical architecture of the system. It can also be used to represent software components that overlap with physical architecture.	Physical design, Implementation
Component diagram	Illustrates the physical relationships between software components.	Physical design, Implementation
Composite structure diagram	Describes the internal structure of a class, such as the relationships between parts of a class.	Analysis, Design
Profile diagram	It is used to build UML language extensions.	None
Behavioral diagrams		
Use case diagram	Identifies system requirements to illustrate the interactions between the system and its environment.	Analysis
Activity diagram	Illustrates the independent workflows of a class, the workflow in a use case, or the detailed design of a method.	Analysis, Design
Sequence diagram	Model the behavior of objects in a use case with an emphasis on the temporal order of activities.	Analysis, Design
Communication diagram	Model the behavior of objects in a use case with an emphasis on communication between objects collaborating in an activity.	Analysis, Design
Interaction overview diagram	Illustrates the order of control flows of a process.	Analysis, Design
Timing diagram	Shows the interaction within a set of objects and the state changes they go through over time.	Analysis, Design
State machine diagram	Examines the behavior of a class.	Analysis, Design

UML diagrams in the development cycle

- Certain diagrams may play a more important role depending on the stage of system development
- The same type of diagram can be used during the development cycle
  - It starts initially from conceptual and abstract aspects
  - Diagrams evolve and include details that will ultimately be the basis for generating or writing code
- The diagrams will make the transition from documenting the requirements to establishing the design details
- The relationships between UML diagrams allow the creation of consistent models
- They are cover for most aspects that describe a system

Use case diagram

It graphically represents the functionalities that the computer system must fulfill

Together with the textual description of each use case it is called the **requirements model**

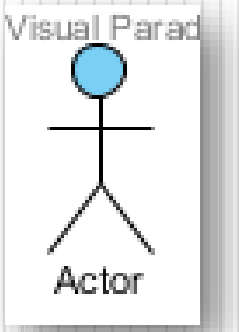
They consist of actors and use cases, along with the relationships between them

Use case

- Specifies a set of actions executed by a system or a subject that lead to a certain result.
- The result, normally, is important for an actor or a beneficiary.

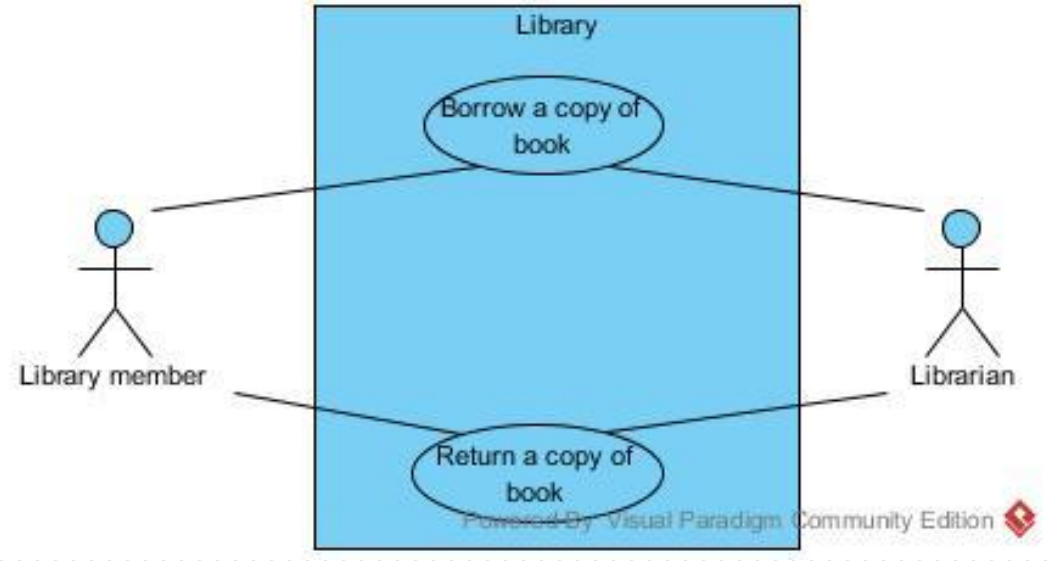
Actors

- An actor interacts with the system in the context of a use case.
- Actors are roles that can include human factors, external hardware or other systems.
- Answer to questions like:
  - WHO requests information from the system
  - WHO modifies information in the system
  - WHO interacts with the system
- It is represented in a diagram with the specific symbol of a little man.



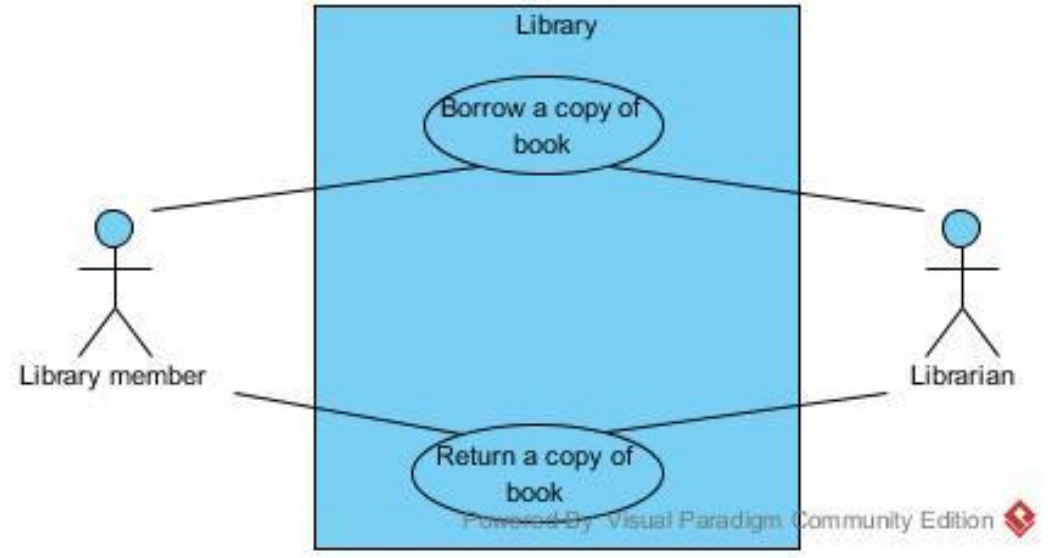
Use Case Diagram

- It describes the relationship between a set of use cases and actors who participated in these UC.
- UC diagrams do not describe processes or flows.
- It can define the boundaries of the analyzed system by incorporating the use cases within a rectangle



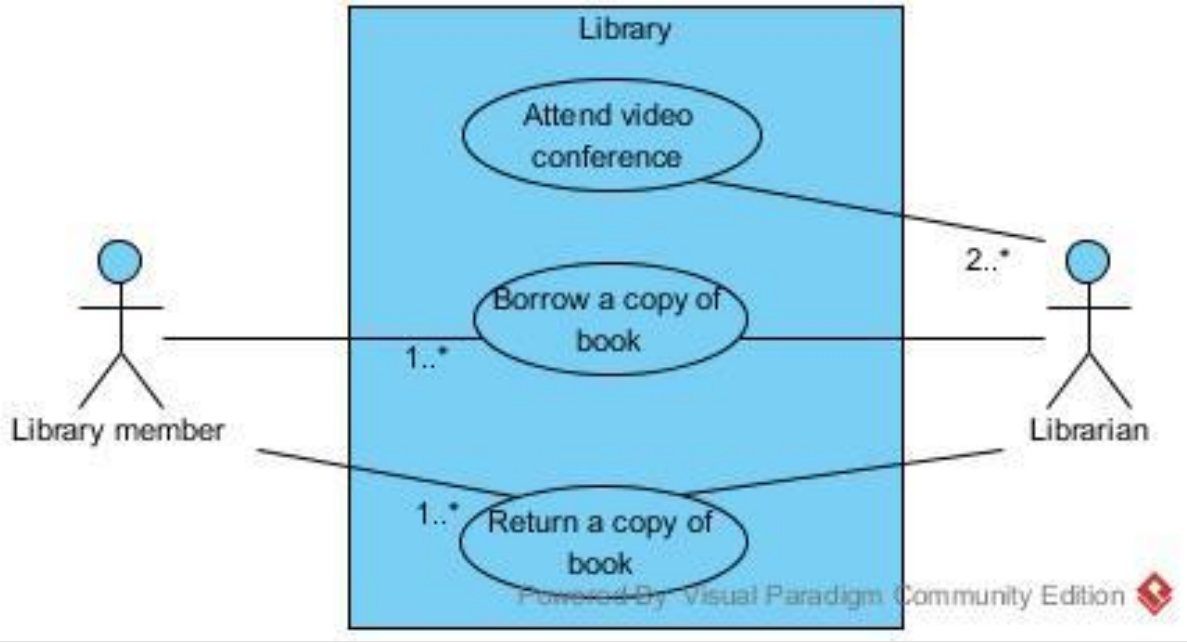
Relationships between actors and use cases 1

- Simple associations are used to connect an actor with a use case.
- It is a way of communication between the two.
- The communication can be unidirectional as well.



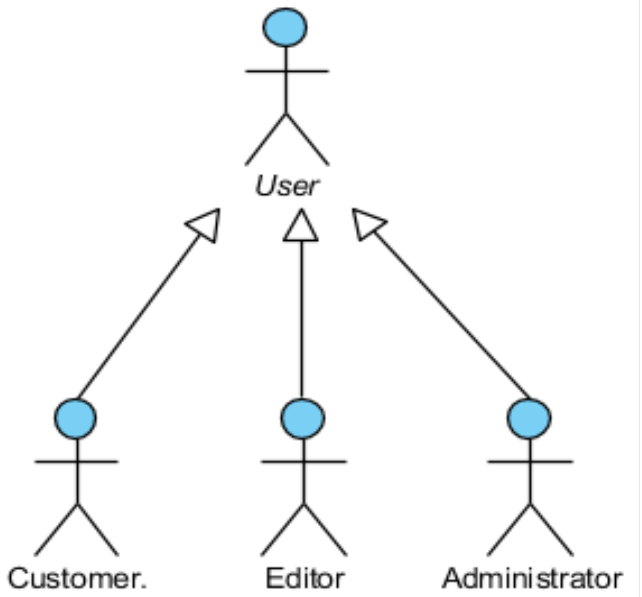
Relationships between actors and use cases 2

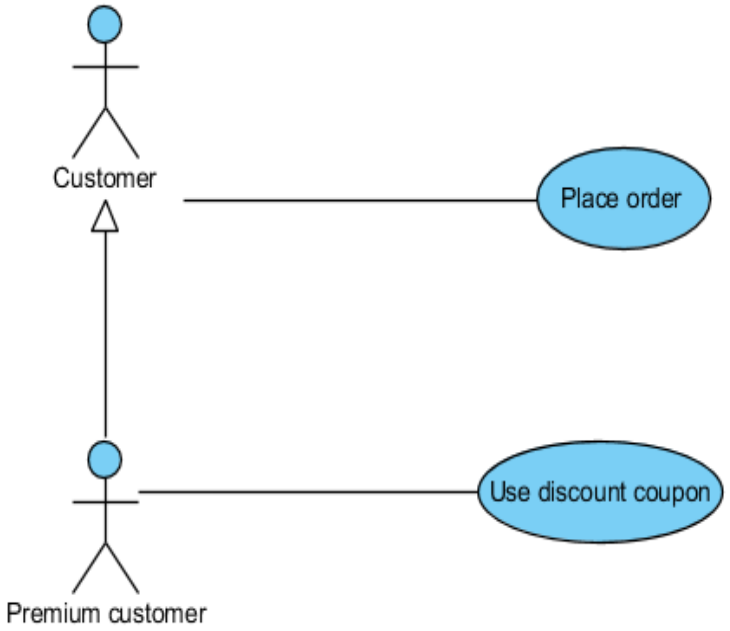
- At this level multiplicities are allowed
  - multiplicity greater than one at the end:
    - corresponding to the Use case ⇨ the actor is involved in several use cases of that type and can initiate use cases: in parallel (concurrently) at different points in time or mutually exclusive in time.
    - corresponding to the Actor ⇨ multiple instances of the actor are involved in initiating the use case and can perform simultaneous and successive actions.
  - UML does not have standard notations for the above situations.



Relationships between actors

- Are of the type generalization / specialization between an abstract actor and one or more concrete actors

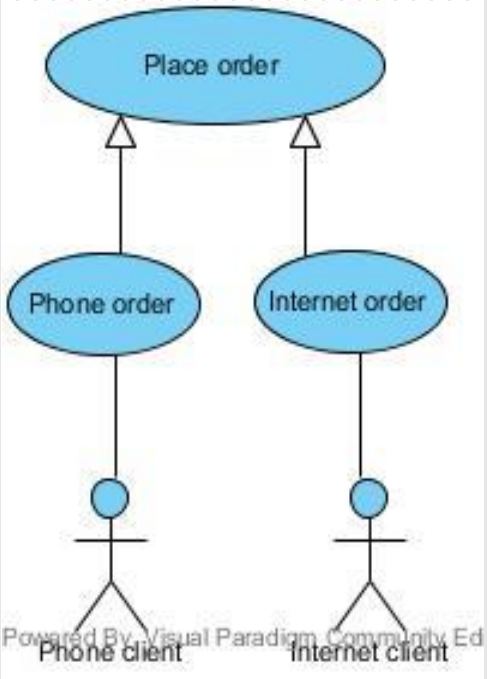




Relationships between use cases - 1

1. Generalization

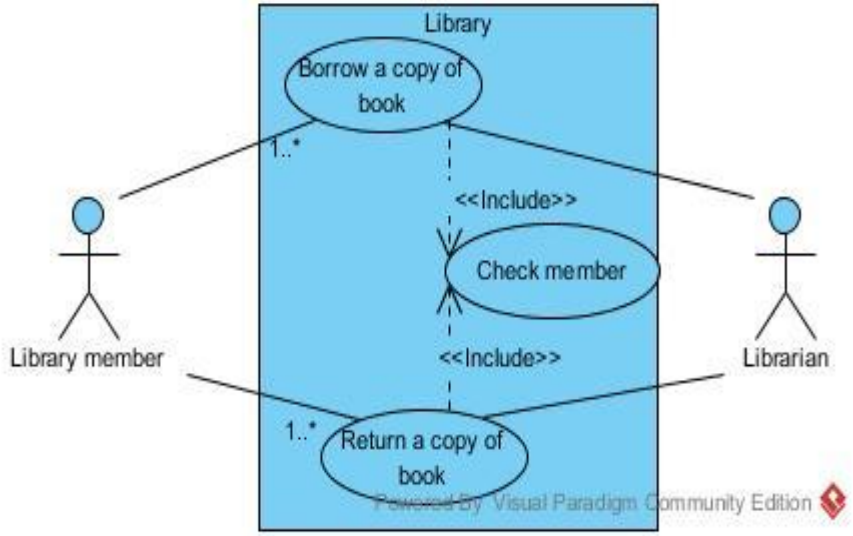
- It is used when there are two or more UC who have in common behavior, structure and purpose.
- The behavior of the parent UC can be overridden.
- Only the differences between the two cases are specified in the specialized use case.

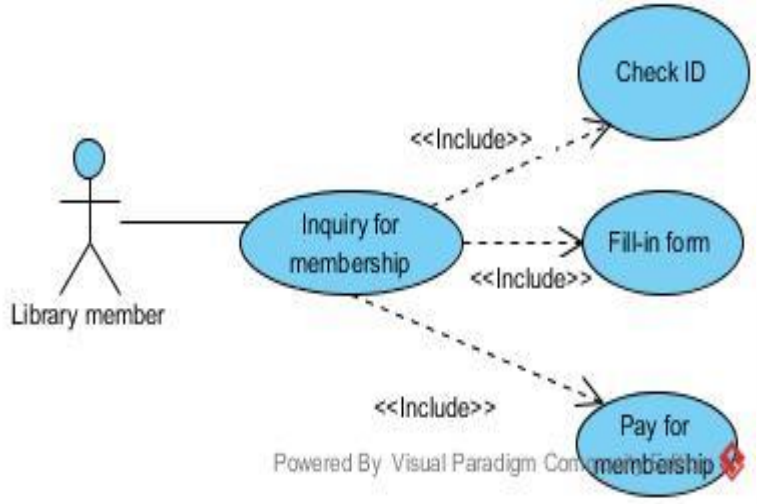


Relationships between use cases – 2

2. Include

- It aims to integrate an use case in another use case, the first becoming a logical part of that UC. The UC that includes another is not complete.
- It is used when:
  - there are parts of behavior common to several use cases
  - to simplify large UC.
- It is equivalent to call a subroutine in programming.
- Denotes a **mandatory** behavior, not an optional one.
- There are no properties inherited from a UC to another.
- It avoids redundancy of parties with identical behavior

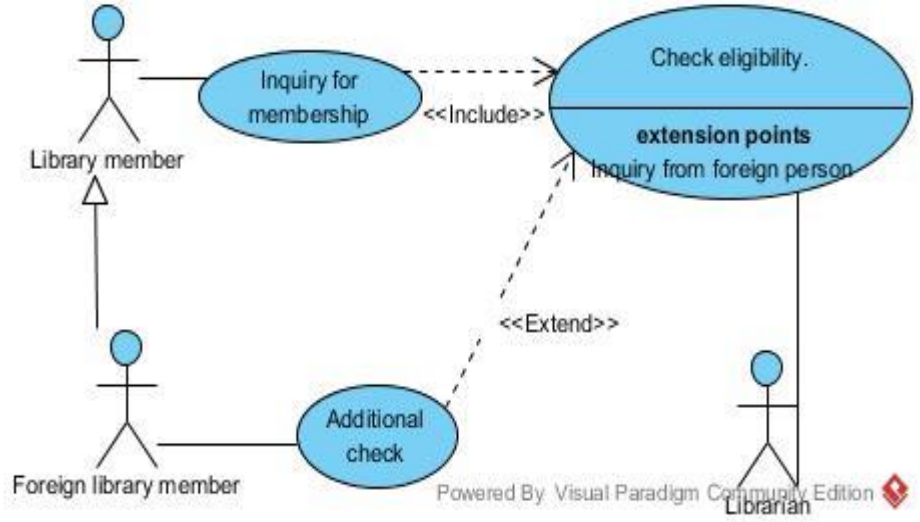


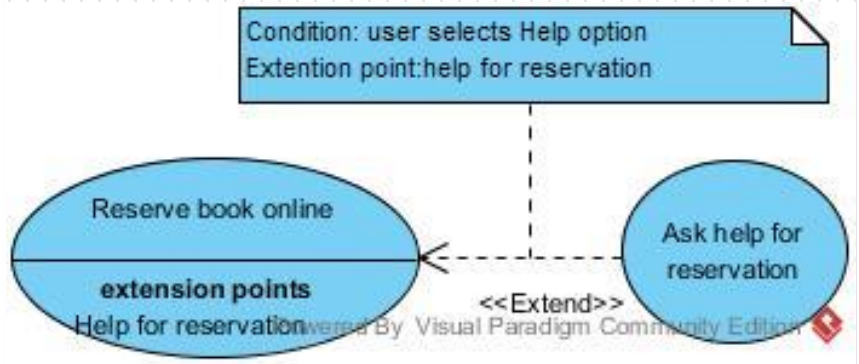


Relationships between use cases – 3

3. Extend

- It is used when a CU occurs only under certain conditions or it is optional.
- The extended UC is complete and independent from the one it expands.
- Extension occurs in one or more **extension points**
- You can associate **notes or constraints** to that relationship to illustrate the conditions in which the extended behavior must be executed.



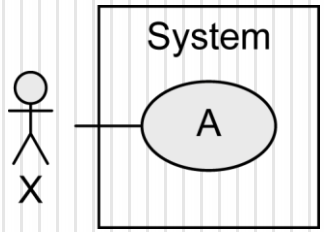
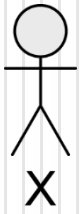


Textual description of a UC

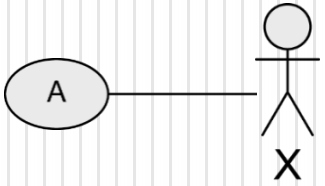
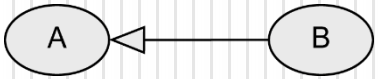
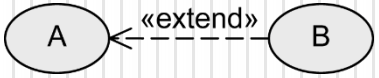
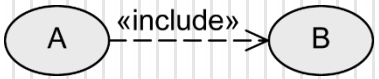
- All the processes that must be executed by the system are found in a use case.
- The use cases are then described textually or in a sequence of steps.
- The activity diagram can be used for graphical modeling of scenarios.
- Templates can be used to structure the description of a use case.

Element of the use case	Description
Code	Unique identifier associated to the use case
State	The stage of completion it is in, for example, outline, completed or approved
Purpose	The system (or part of the system) or application to which it belongs
Name	Name of the use case,as short and representative as possible
Main actor	The beneficiary who initiates the use case and pursues a particular purpose
Description	Short presentation, in free text, of the use case
Preconditions	What conditions must be satisfied for the script can begin
Postconditions	What conditions must be met to ensure a successful end of the scenario
Trigger	An event or sequence of events that initiate the use case
Basic flow	The basic flow describes the basic flow of events when everything is going according to a predetermined script; there are **no exceptions or errors**
Alternate flows	The most **significant exceptions** and **alternatives** to the baseline scenario
Relashionships	What relationships it has with other use cases (include or extend)
Frequency of use	How often it is expected to use this system functionality
Business rules	What rules govern the use case

Notation Elements (1/2)

Name	Notation	Description
System		Boundaries between the system and the users of the system
Use case		Unit of functionality of the system
Actor		Role of the users of the system

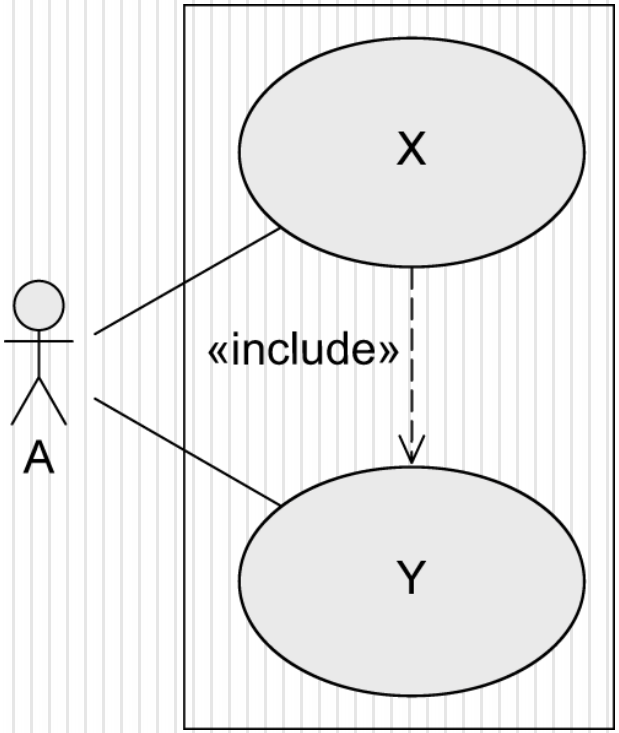
Notation Elements (2/2)

Name	Notation	Description
Association		Relationship between use cases and actors
Generalization		Inheritance relationship between actors or use cases
Extend relationship		B extends A: optional use of use case B by use case A
Include relationship		A includes B: required use of use case B by use case A

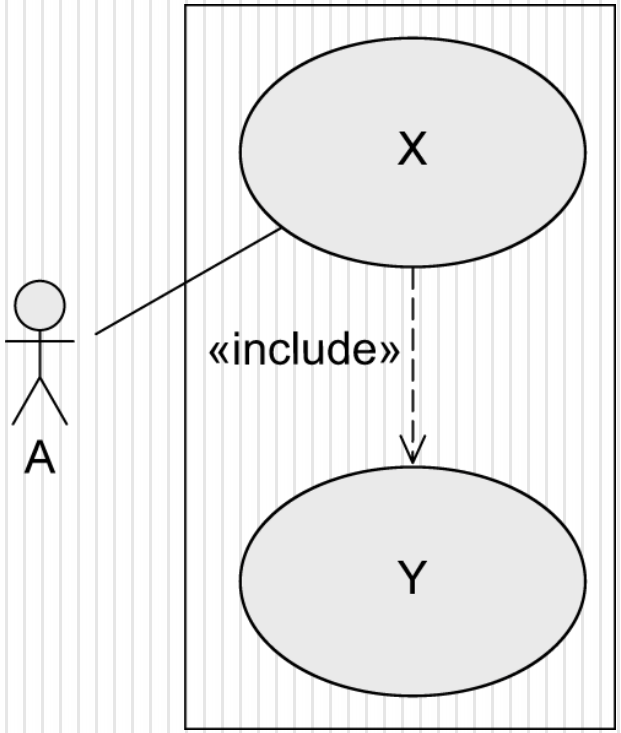
Best Practices

«include»

UML standard



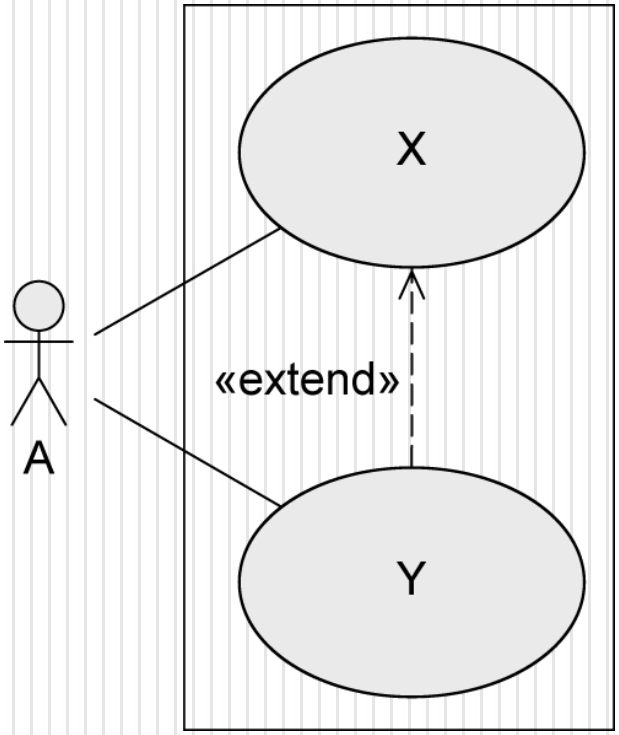
Best practice



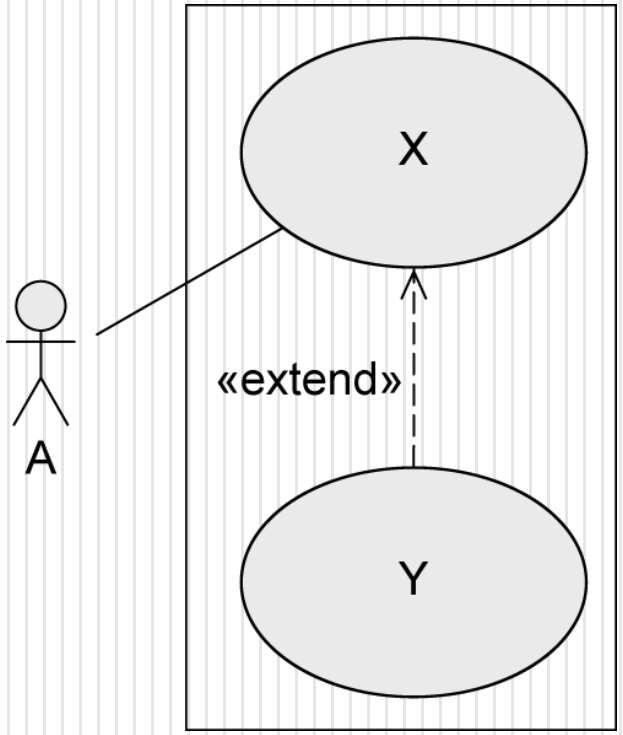
Best Practices

«extend»

UML standard



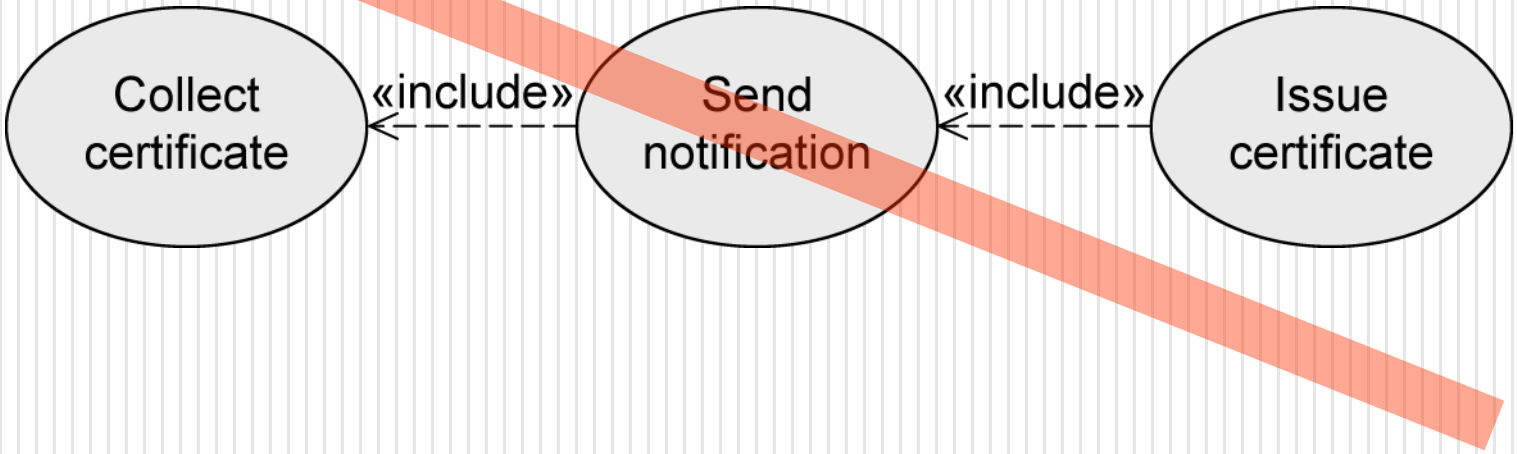
Best practice



Best Practices

Typical Errors To Avoid (1/5)

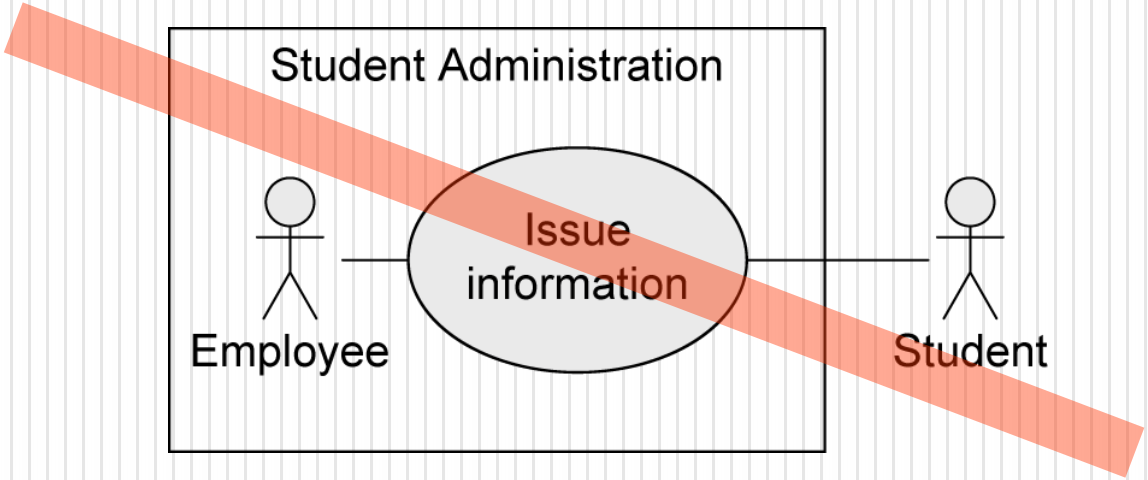
- Use case diagrams do not model processes/workflows!



Best Practices

Typical Errors To Avoid (2/5)

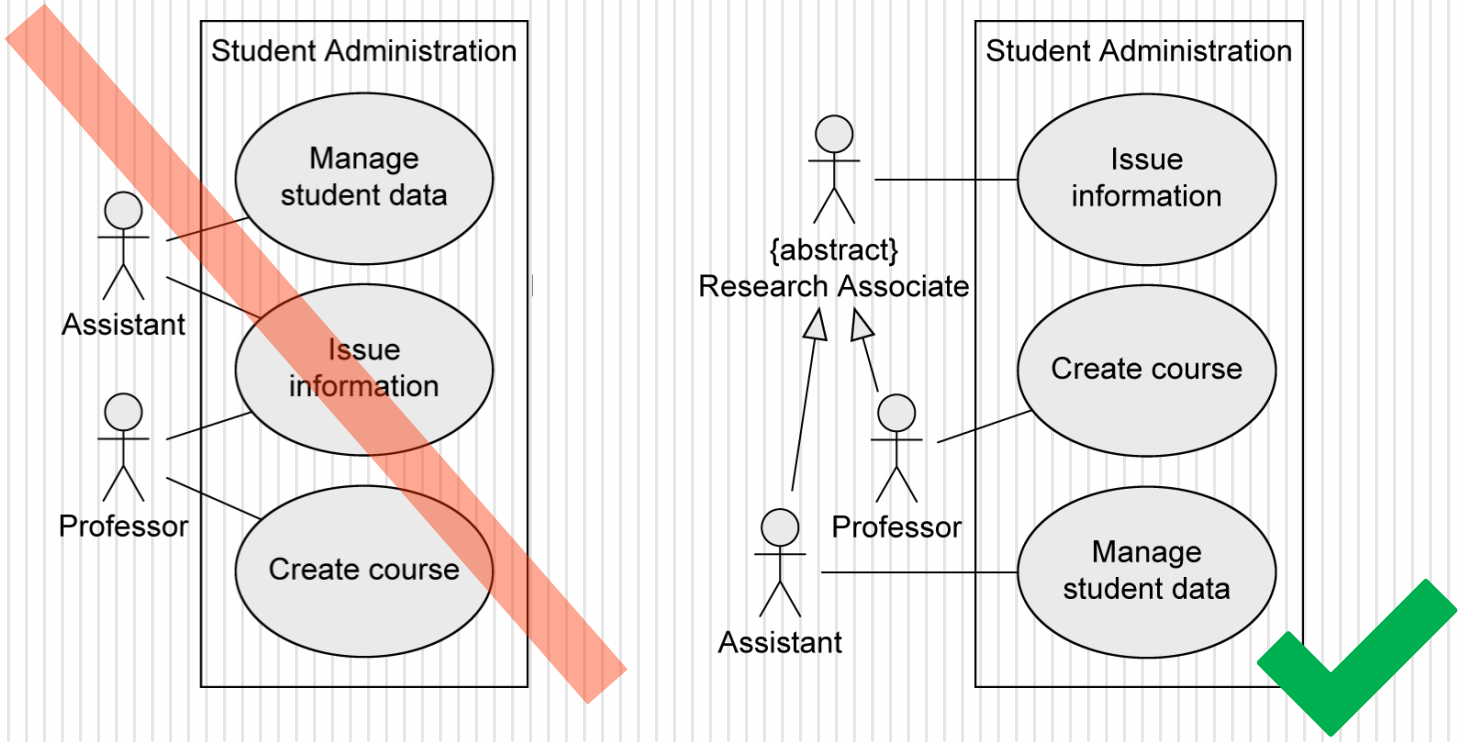
- Actors are not part of the system, hence, they are positioned outside the system boundaries!



Best Practices

Typical Errors To Avoid (3/5)

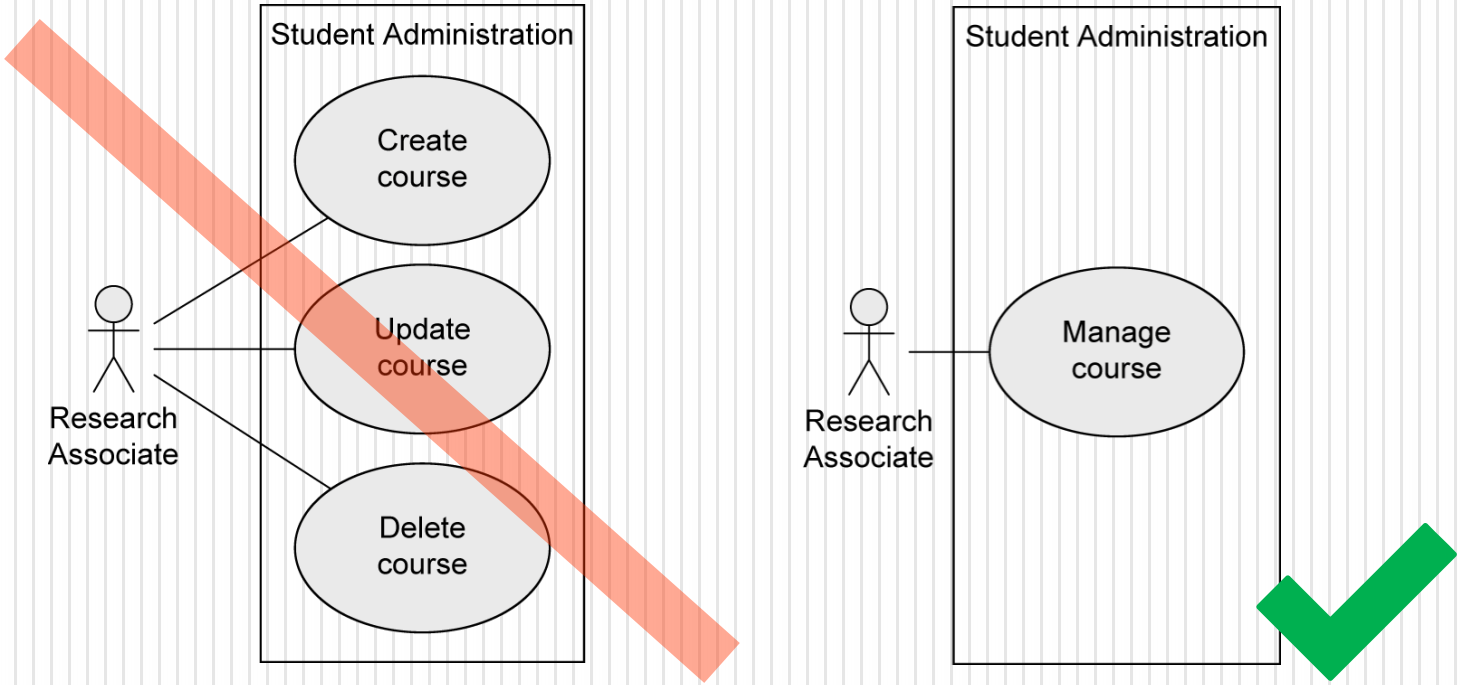
- Use case **`Issue information`** needs **EITHER** one actor **`Assistant`** **OR** one actor **`Professor`** for execution



Best Practices

Typical Errors To Avoid (4/5)

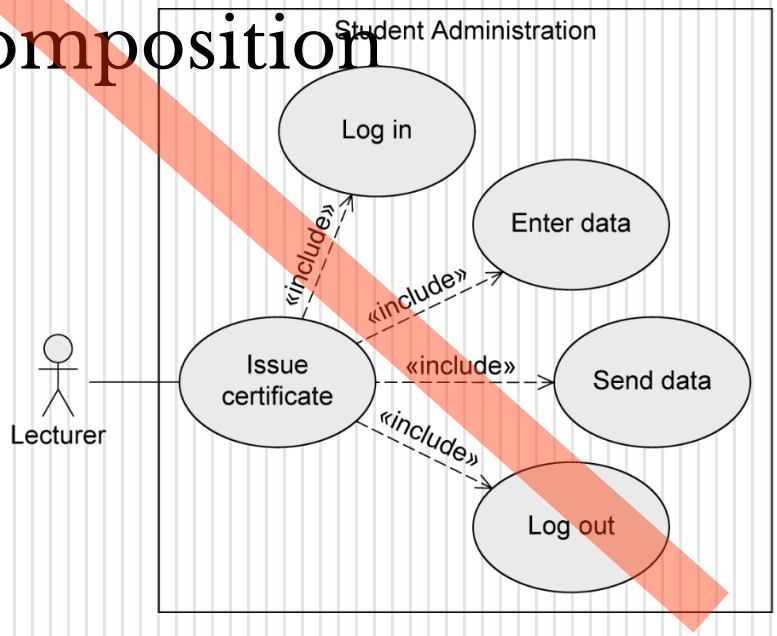
- Many small use cases that have the same objective may be grouped to form one use case

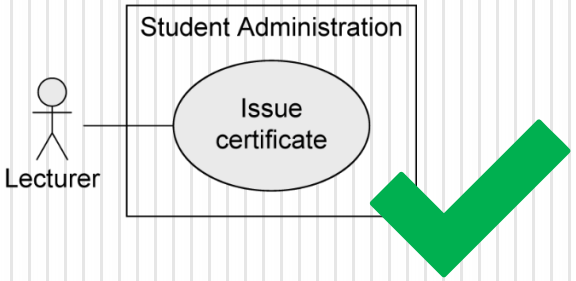


Best Practices

Typical Errors To Avoid (5/5)

- The various steps are part of the use cases, not separate use cases themselves! -> NO functional decomposition





Seminar work

Create the general use class diagram and the textual description for a use case,for the scenario below

The project goal is to develop a software application for the management of a hotel business unit. In order to check in, a customer can request to reserve one or more rooms by e-mail or telephone. For this, he provides the receptionist with information on the period of accommodation and type of rooms required. Customers will get discounts if they reserve at least 3 rooms or if the period of accommodation exceeds 5 days. The receptionist checks availability and notifies the client of this and the estimated cost of accommodation. If there are no rooms available as requested, the receptionist can provide alternatives to the customer. The client may request a discount (additional or not) and the receptionist will decide the feasibility discount, assisted mandatory by the hotel manager. If the client agrees with the proposed price, they proceed to the reservation. For new customers, the receptionist asks identification data, which he introduces in the application.

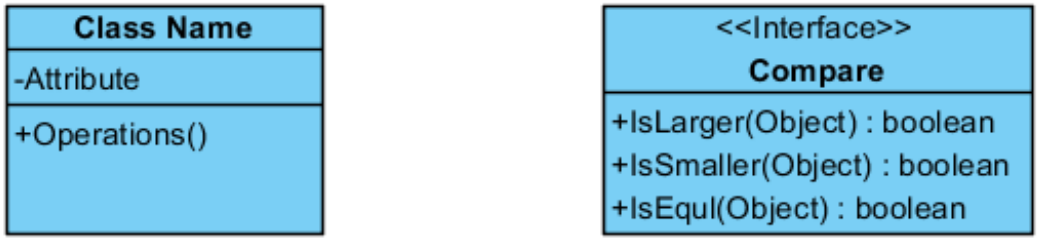
Once at the hotel and if it has made a prior booking, the customer will provide his identification and / or booking number and the check in is finalized. If there is no reservation, the availability for the required period will be checked. When there is such a room, accommodation is made. At the end of the stay, the receptionist prepares a list of all the services used by the customer and their price. The list must be validated by the customer, then the final invoice is drawn up. The invoice can be paid partially or fully by bank transfer, cash or using a credit card. Also, before leaving the hotel, the customer is asked to complete a form to evaluate the services provided by the hotel premises

Information system design
Seminar 4

UML Class Diagram

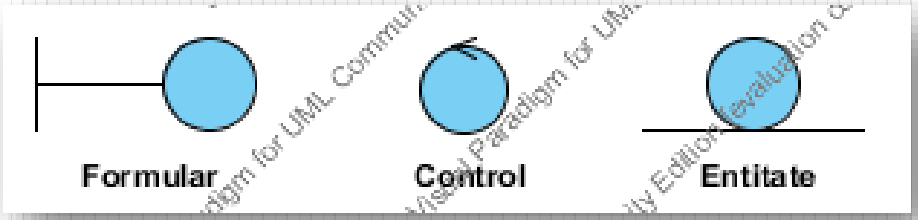
Defining a class

- Set of objects that have the same characteristics and constraints.
- The characteristics of a class are attributes and operations.
- Abstract classes can not be instantiated. Their role is to enable other classes to inherit them, for reuse of characteristics.
- An interface describes a set of characteristics and public obligations. Actually, it specifies a contract. Any instance that implements the interface must provide the services provided by the contract.



Examples of common stereotype classes

- <<entity>> – a passive class, which does not initiate interactions;
- <<control>> – initiates interactions, contains a transactional components and acts as separator between the entities and limits;
- <<boundary>> – it is located an the periphery of the system, but inside. It's the contact element to the actor or to other systems.
- <<enumeration>> - it is used to define data types whose values are listed.
- <<primitive>> - a form of class that represents predefined data types, such as Boolean.



Attributes -1

- Each attribute is described at least by its name.
- Additional information can be added, and the general structure of an attribute is:
- [visibility][/]name[:type][multiplicity][=default value] [{property}]
- Visibility may be:
  - + Public: can be viewed and used by anyone
  - - Private: only the class itself has access
  - # Protected: the class and subclasses have access
  - ~ Package: only classes in the same package have access
- / symbolizes a derived attribute

Attributes -2

- UML allows specification of multiplicity for attributes ,when you want to define more than one value for an attribute. They have the following meanings

Multiplicity	Meaning
1	Exactly 1 (default)
2	Exactly 2
1..4	From 1 to 4 (inclusive)
3, 5	3 or 5
1..*	At least one or more
*	Unlimited (including 0)
0..1	0 or 1

- Property indicates an additional property that applies to the attribute:
  - {readonly}: the attribute can be read but not modified
  - {ordered}, {unordered}: an ordered or unordered set
  - {unique}, {nonunique}: the set of values may or may not contain identical items

Operations

- The general form of an operation is:
- [visibility] name ([direction] parameter list) [:returned type] [{property}]
- Visibility - the same as in class
- Direction - 'in' | 'out' | 'inout' | 'return'
- Return type - whether they return something, if it's a function
- An example of an operation's property: {query} – it does not change the stat of an object or other objects

Attribute examples:

- - age: Integer {age>18}
- # name:String[1..2]="Ioana"
- ~ Id:String {unique}
- / TotalValues:Real=0

Operations examples :

- + setAge (out Age: Integer)
- + getAge(in Id:String): Integer {query}
- - changeName(inout Name:String)

Constraints

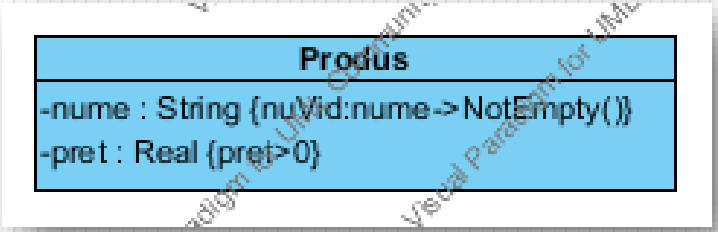
- A constraint is an expression that restricts a certain element of a class diagrams.
- This may be a formal expression (written in Object Constraint Language - OCL) or it can have a semi-formal or informal format.
- They are represented in braces.
- They can be written immediately after defining an element or as a comment.
- A constraint can have a name, as follows: :
- **{name : Boolean expression}**

Examples of OCL constraints:

context Organisation

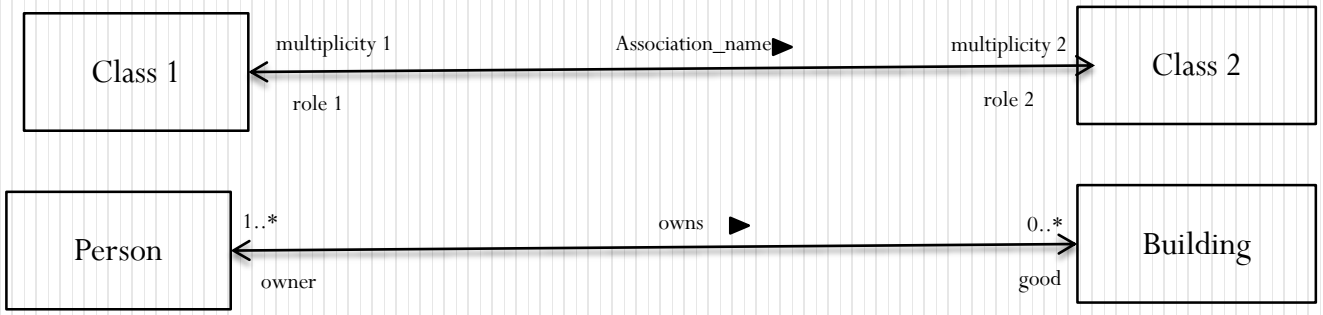
inv: self. departaments→isUnique (name)

inv: departaments.employees→isUnique (code)



Relationships between classes - 1

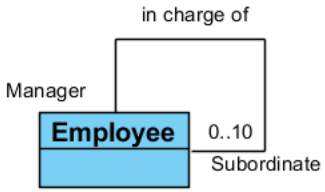
1. *An association* implies establishing a relationship between classes.It is characterized by:
   - name (optional)
   - multiplicities – specified at both ends of the association
   - roles of the association: specified at each end of the association and contain a brief and representative description (1-2 nouns)
   - navigation direction



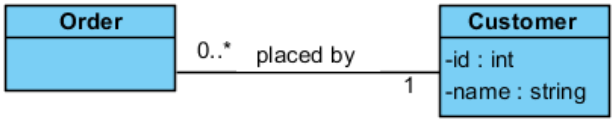
Relationships between classes -2

Types of associations :

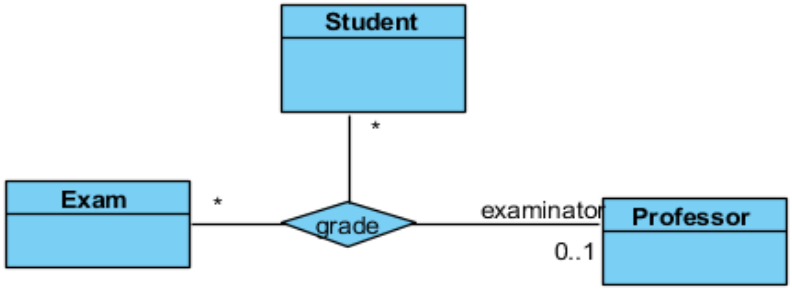
- Unary: connects a class to itself.



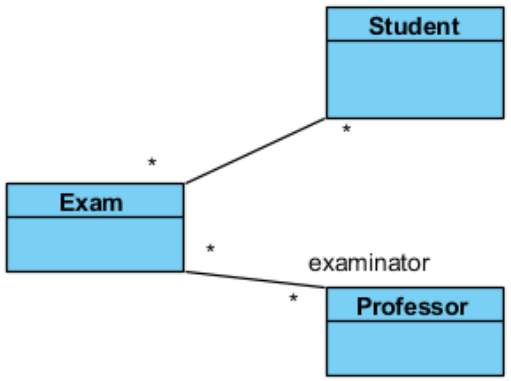
- Binary: between two classes.



- Ternary: they are usually transformed in binary association.

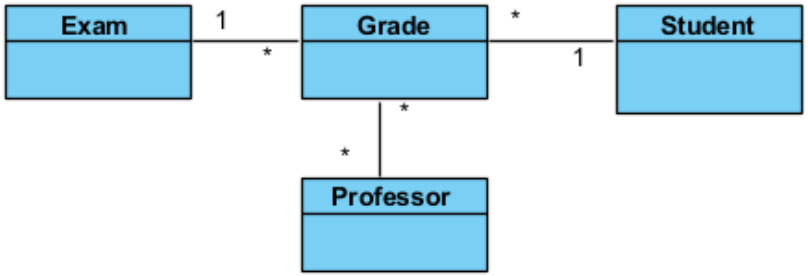


N-ary association



Varianta 2: This transformation created a model with a different meaning.

- ✓ First, in this representation, an exam can be scored by several teachers.
- ✓ Secondly, it is not clear which teacher evaluated a particular student in an exam.

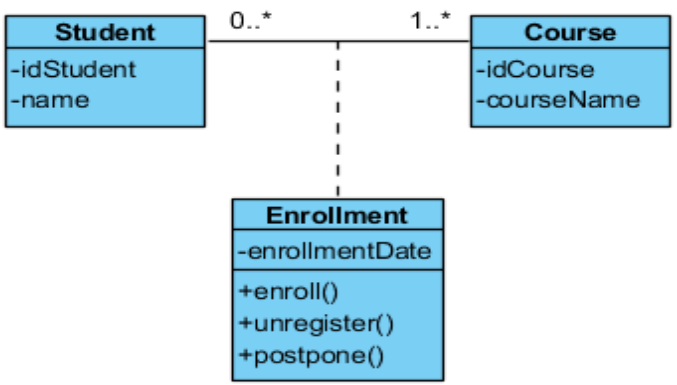


Varianta 3: An additional class was added, Grade.

- ✓ this model allows a student to be graded several times for the same exam, which is not possible with the ternary association model

Relationships between classes -3

- The association modeled as a class allows the relationship to have attributes and operations.

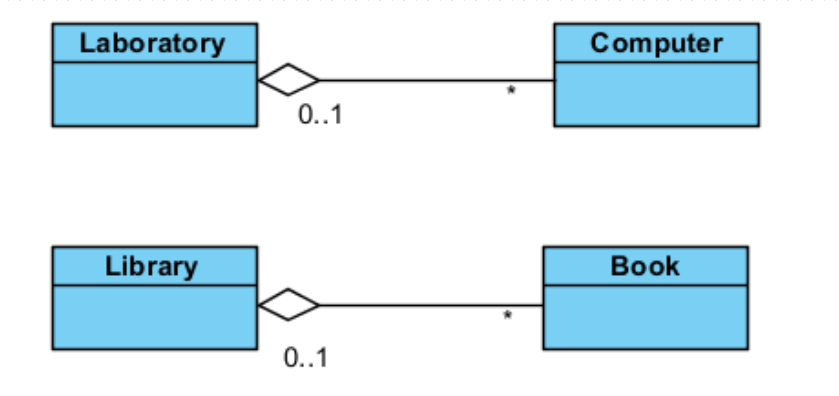


2. *The Aggregation relationship* is a binary form of association representing a 'part – of ' type relationship .

- It can be of two types:
  - Shared aggregation (aggregation)
  - Composed aggregation (composition / notn-shared association)

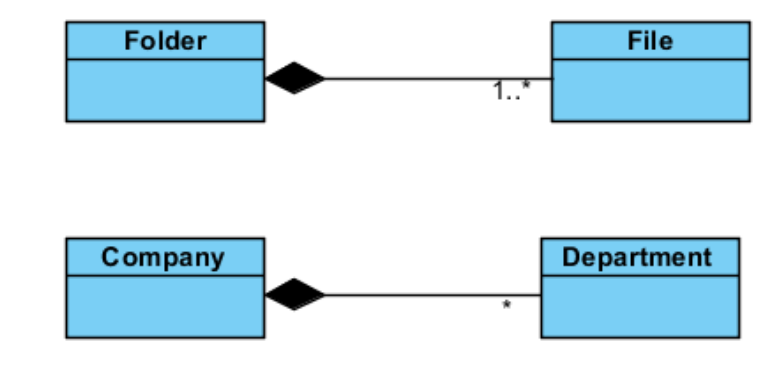
Relationships between classes -4

- *Shared aggregation* is a weak form of aggregation in which the instances of the parts are independent from the whole, as follows:
  - The same elements can be aggregated by several other 'whole' classes
  - If you delete the class that aggregates, the aggregated classes will continue to exist.
  - It is represented using the shape of a diamond at the end of the association corresponding to the aggregating class.



Relationships between classes -5

- *Composed aggregation* is a powerful of aggregation form in which the instances of the parts are independent from the whole, as follows: :
  - If the aggregating class is deleted, the aggregated classes will also be deleted.
  - It is represented using the shape of a full diamond at the end of the association corresponding to the aggregating class.
  - When used for modeling objects in a certain domain, the deletion may be figuratively interpreted as an "end" and not as a physical destruction.



Relationships between classes -6

Association

Objects know of each other and can work together

Aggregation

1. It protects the integrity of the configuration.
2. It works as a whole.
3. Control through a single object.

Composition

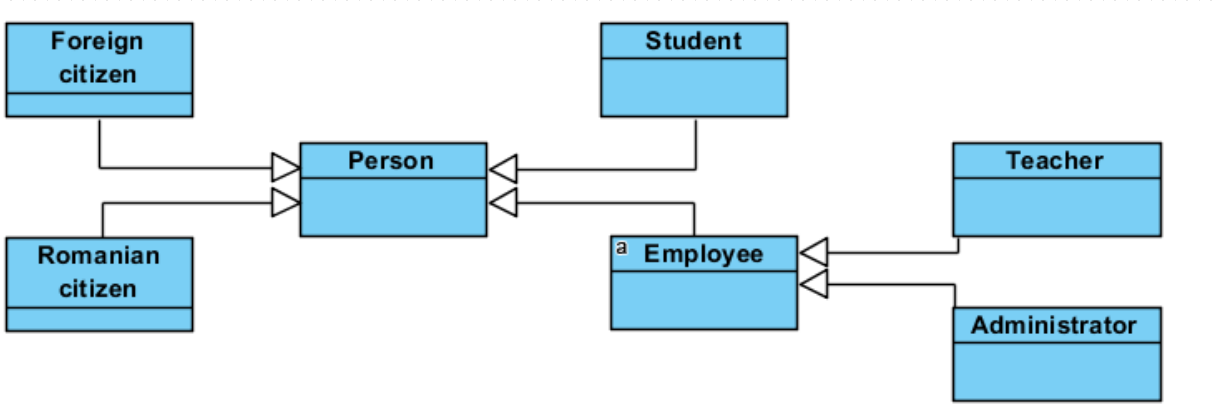
Each part can be member of a single aggregated object.

Relationsips between association, aggregation and composition

Relationships between classes -7

3 The generalization relationship is used to indicate inheritance between a general class (superclass) and a specific class (subclass)

- Also called informally ‘is a type of’ relationship.
- It is represented as an empty triangle placed at the end of the superclass.
- Subclasses inherit the characteristics and constraints of the superclass.
- Multiple inheritance is allowed.



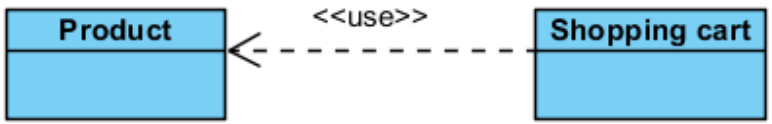
Relationships between classes -8

4. The dependency relationship is used to show a wide range of dependencies between elements of a model

- In the analysis stage, the dependency type can be unspecified.
- At the design stage, dependencies will be personalized with stereotypes or will be replaced with connectors specific to the technology used.
- It is represented as a dotted line from the dependent class "client" to the "supplier" class, with an arrow at the end of "supplier" class .
- In class diagrams, the most important dependencies are ‘use’ and ‘abstraction’

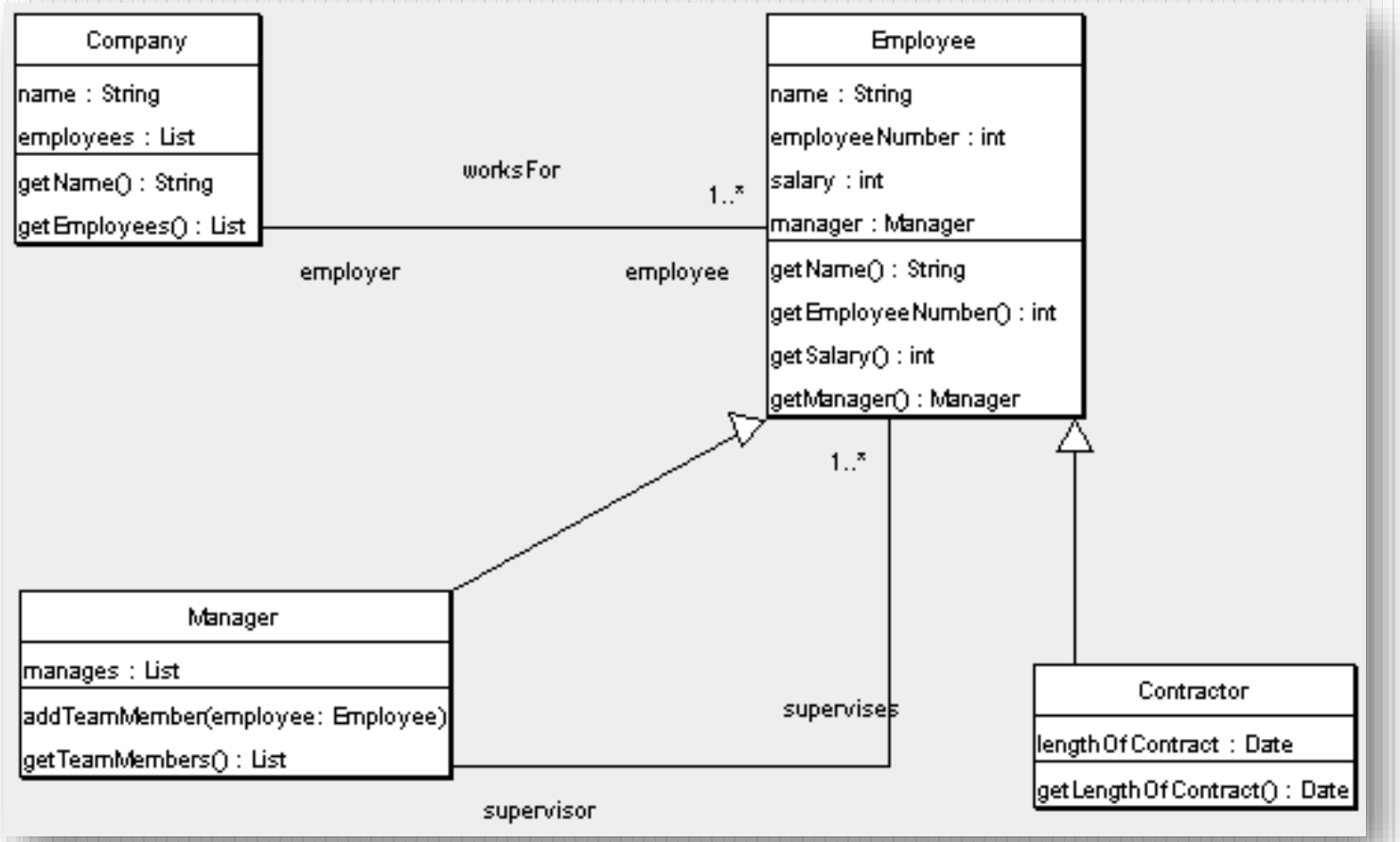
Relationships between classes -9

- Use dependency (<<use>>, <<create>>, <<call>> etc.) is a relationship in which a ‘client’ class needs another class or group of classes (supplier) to operate
- An abstraction dependency relates two elements or set of elements (called customer and supplier), representing the same concept, but at different levels of abstraction or seen from different views



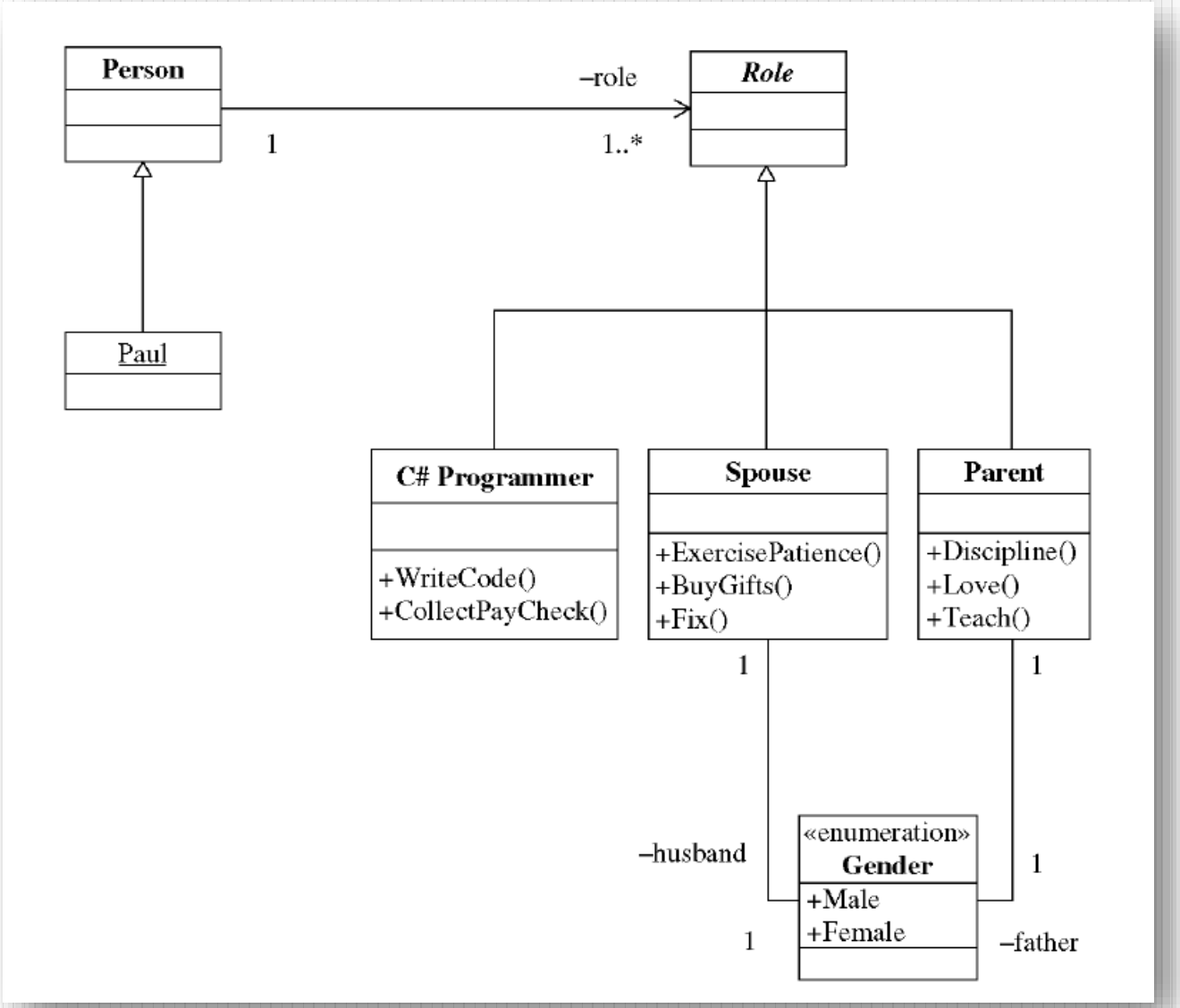
Exercise 1

Read the information represented in the diagram below



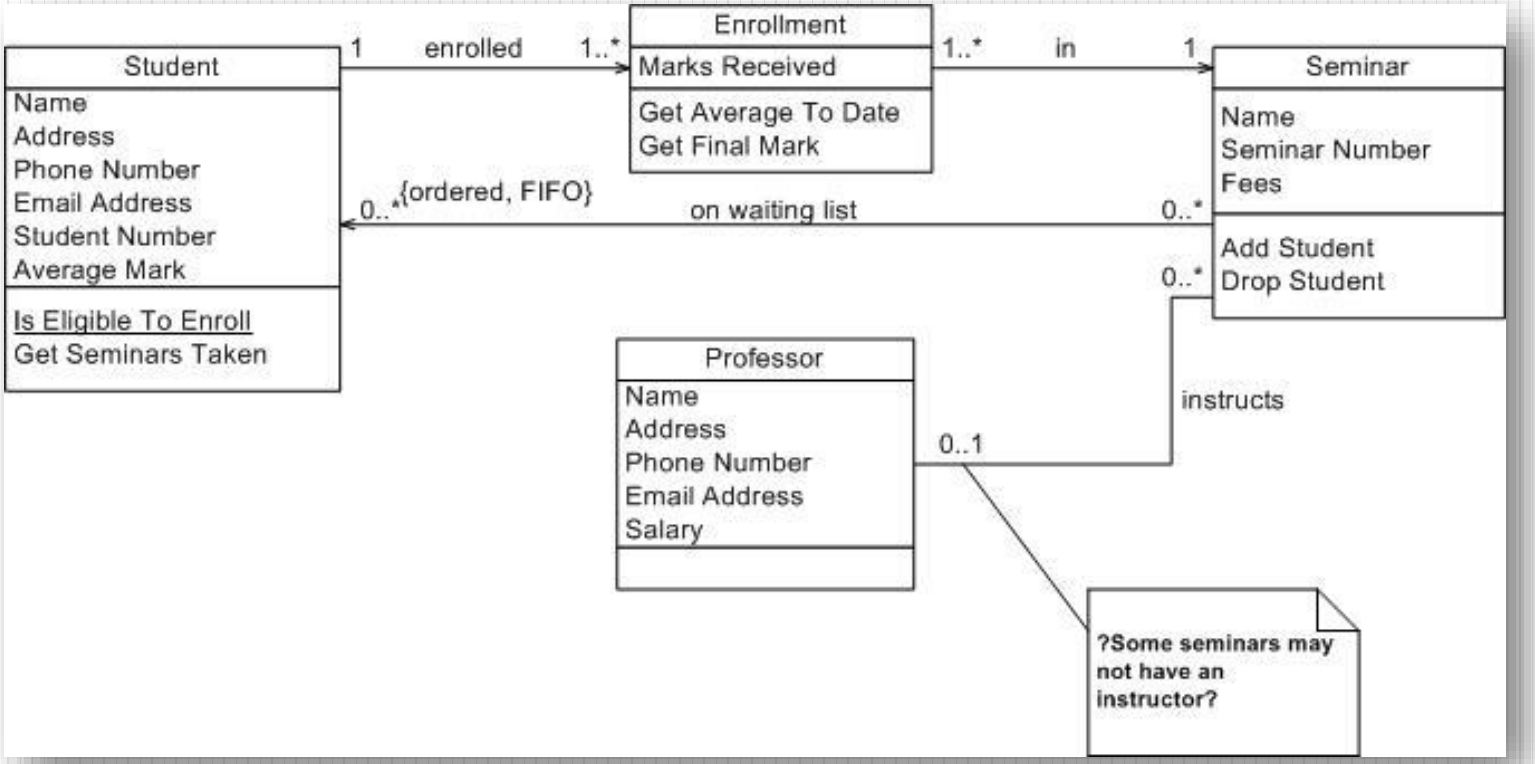
Exercise 2

Read the information represented in the diagram below



Seminar 3

Read the information represented in the diagram below



Seminar scenario

Create the class diagram for the scenario below

The project goal is to develope a software application for the management of a hotel business unit. In order to check in, a customer can request to reserve one or more rooms by e-mail or telephone. For this, he provides the receptionist with information on the period of accommodation and type of rooms required. Customers will get discounts if they reserve at least 3 rooms or if the period of accommodation exceeds 5 days. The receptionist checks availability and notifies the client of this and the estimated cost of accommodation. If there are no rooms available as requested, the receptionist can provide alternatives to the customer. The client may request a discount (additional or not) and the receptionist will decide the feasibility discount, assisted mandatory by the hotel manager. If the client agrees with the proposed price, they proceed to the reservation. For new customers, the receptionist asks identification data, which he introduces in the application.

Once at the hotel and if it has made a prior booking, the customer will provide his identification and / or booking number and the check in is finalized. If there is no reservation, the availability for the required period will be checked. When there is such a room, accommodation is made. At the end of the stay, the receptionist prepares a list of all the services used by the customer and their price. The list must be validated by the customer, then the final invoice is drawn up. The invoice can be paid partially or fully by bank transfer, cash or using a credit card. Also, before leaving the hotel, the customer is asked to complete a form to evaluate the services provided by the hotel premises

Information system design

Seminar 5 - UML Activity diagrams

Activity diagram

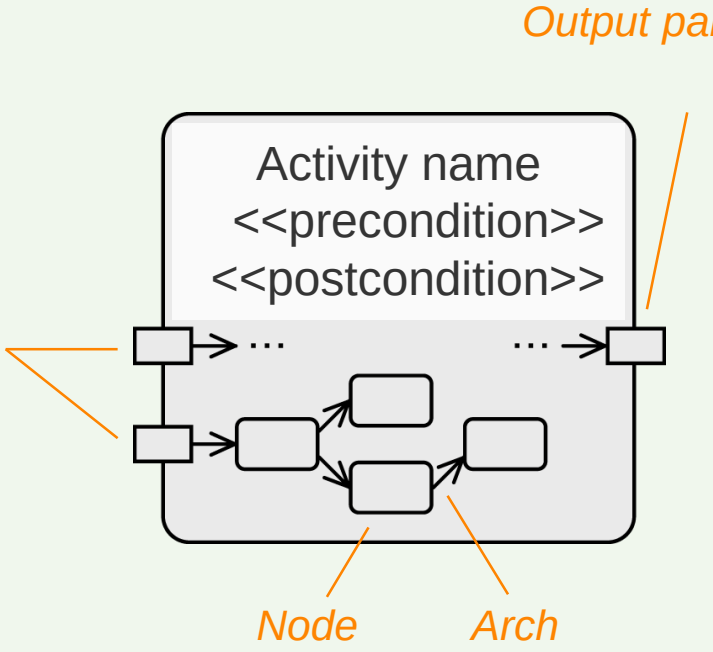
It supports visual representation of sequences of actions that target a certain result.

It may be built for **one or many use cases** or for **complex operation description**.

It describes the workflow from an entry node to a finish node, detailing the **decision branches** that may arise in an activity.

Activity

- Activity - it represents a parametrized behaviour in form of a coordinated flowchart.
- It specifies at different granularity levels the behavior defined by the user
- An activity is a directed graph:
  - Nodes: actions and activities
  - Arcs: for control flows and object flows



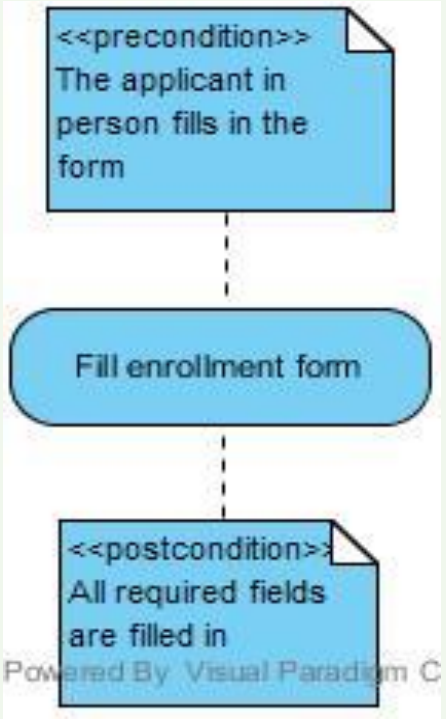
Action

- Action –it represents a single step of an activity.
- Actions are atomic, they cannot be decomposed
- It can be a:
  - physical action, accomplished by a person or
  - electronic action.
- There are special notations for some activity types, such as:
  - Accept event actions
  - Send signal actions

Constraints

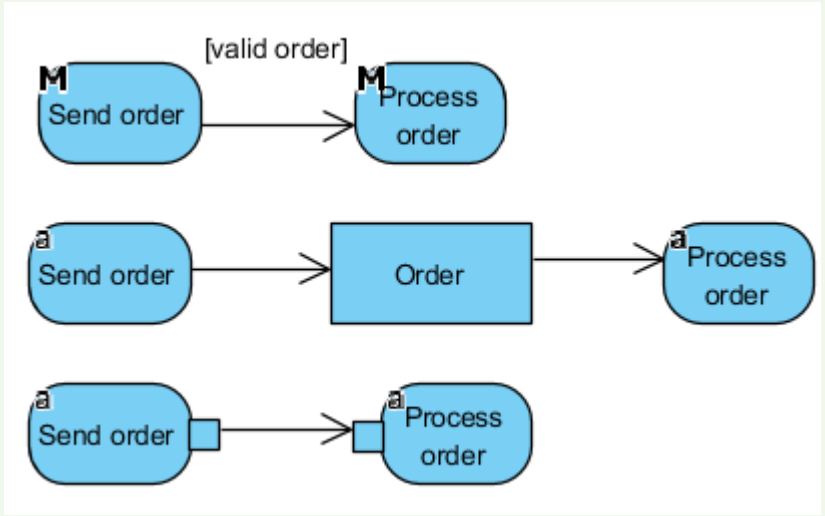
Constraints can be attached to an action, for example, as **preconditions** or **postconditions**.

Use **<<precondition>>** and **<<postcondition>>** keywords**.**



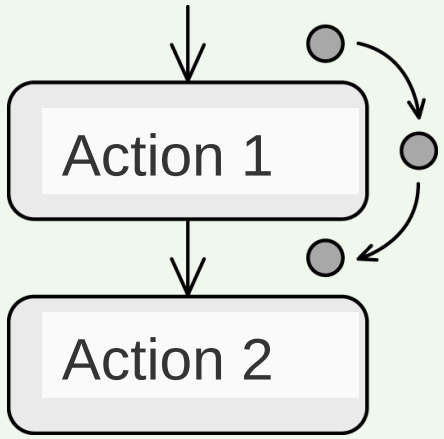
Flows and objects

- **Control flow**– it is an arch on the diagram that describes how control is transferred from an action to another
- **Object flow**- it is a flow along which the objects or data are transferred.
  - It should have an object at least at one end
  - There is also a short notation that uses input and output qualifiers/pins.
- **Transition condition** –it is a text on a flow defining a condition that should be met to activate transition to the next action.



Tokens

- A virtual mechanism that describes exactly the execution flow
  - it is not included in the diagram notations
  - It is a mechanism that grants the actions permission to execute
- If an action receives a token, then the action can be executed. When an action is completed, it sends the token of the next action, triggering its execution
- There are two types of tokens:
  - **Control token**: "execution permission" for a node
  - **Object token**: data transport + "execution permission"

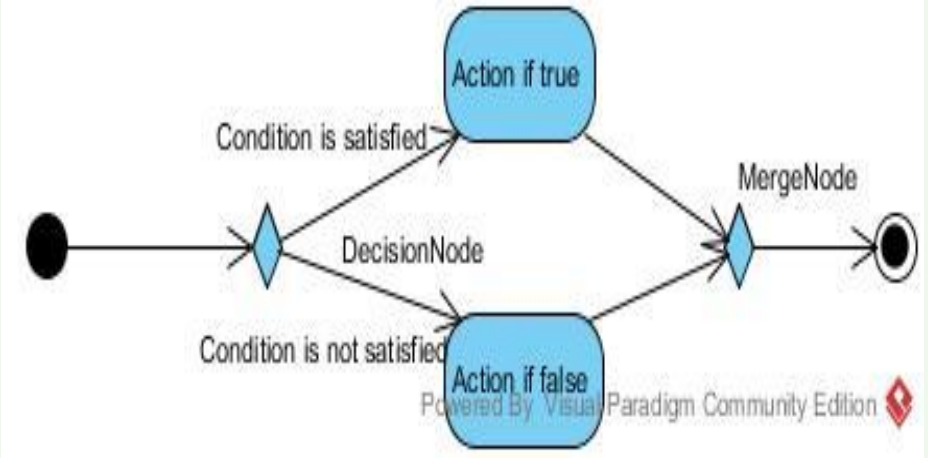


Nodes

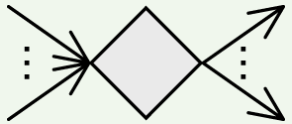
- **Initial node** ●
  - it represents the initial point of the diagram.
  - it sends tokens to all output flows
  - it keeps tokens until the successive nodes accept them
- **Final node**- there are two types of final nodes:
  - ***Activity final node***:◉ it represents the end of all control flows in a digram.
  - ***Flow final node***: ⊗ it shows the process ends at that point. It represents the end of a single control flow.

Decision and merge nodes

- Both are represented by a diamond and may have a name
- **Decision node** :
  - It has an input flow and several output flows
  - Output flows must be accompanied by **mutual exclusive conditions**
  - The token chooses **only one path**

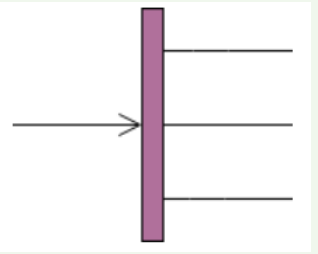


- **Merge node**
  - It has several input flows and a single output flow
  - It sends the token to the next node
- There is also the combined version of the decision and junction node

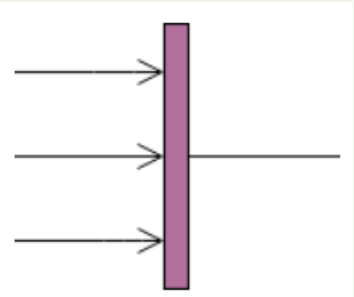


Fork and join nodes

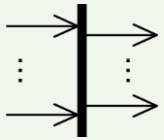
- They are both represented as a black thick line
- **Fork node**:
  - It has an input flow and several output flows
  - It shows the start of some parallel actions
  - Token copies are send to the all the output flows



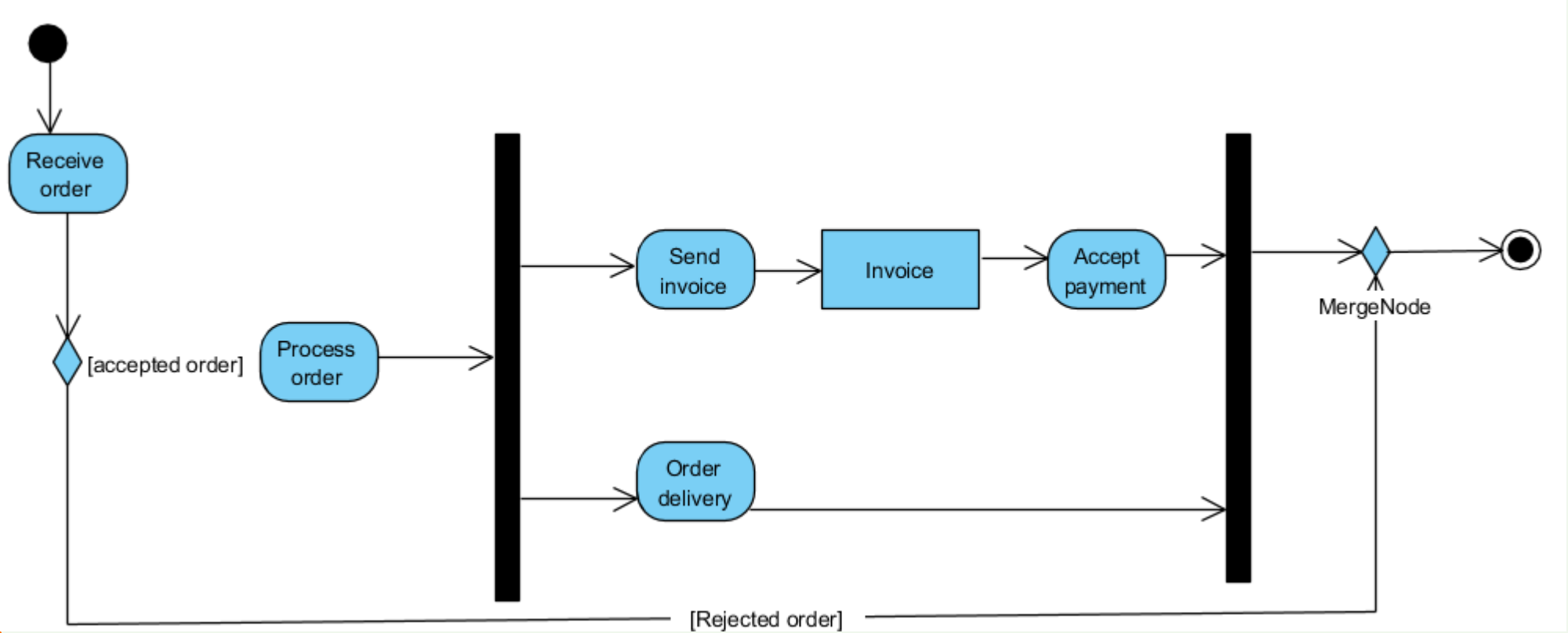
- **Join node**:
  - It has several input flows and a single output flow
  - All the flows entering the join node must reach the node before the processing can move on
  - It shows the end of some parallel processing actions



- A combination of fork and join nodes can be used



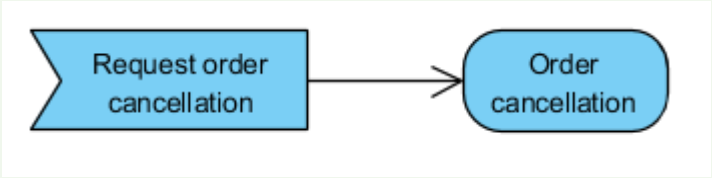
Nodes- exemple



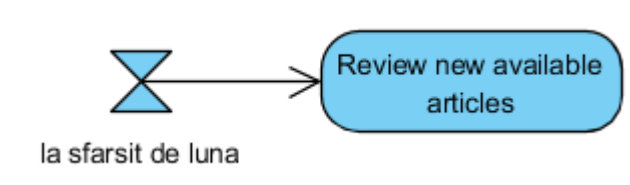
Event-based actions

- **Send Signal Action**

- **Accept Event Action**
  - An action that accepts an **event**

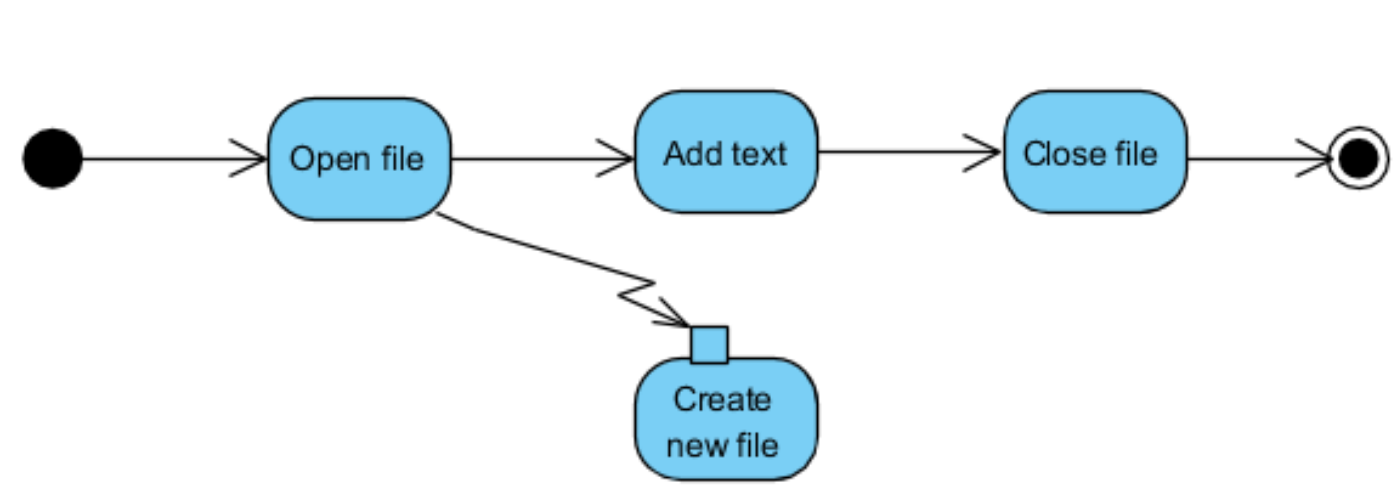


  - An action that accepts a **time event**



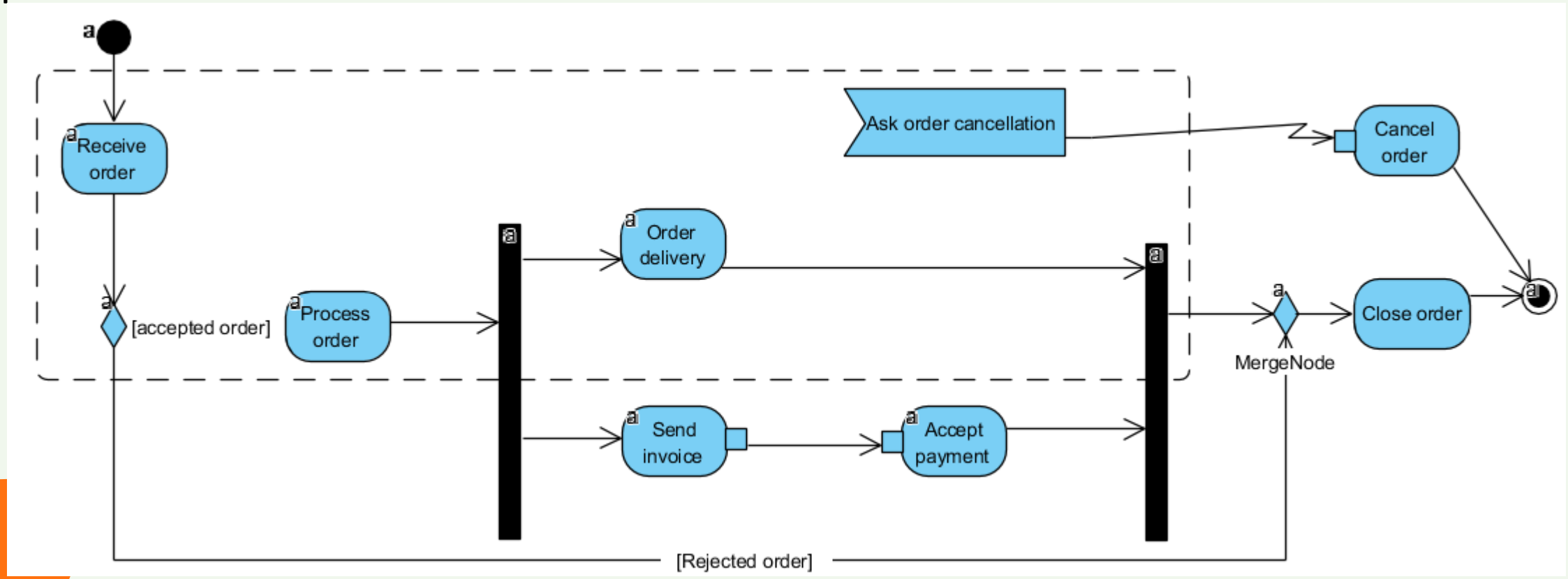
Handling exceptions

- Defines how the system should react in a given error situation
- The exception handler replaces the action where the error occurred



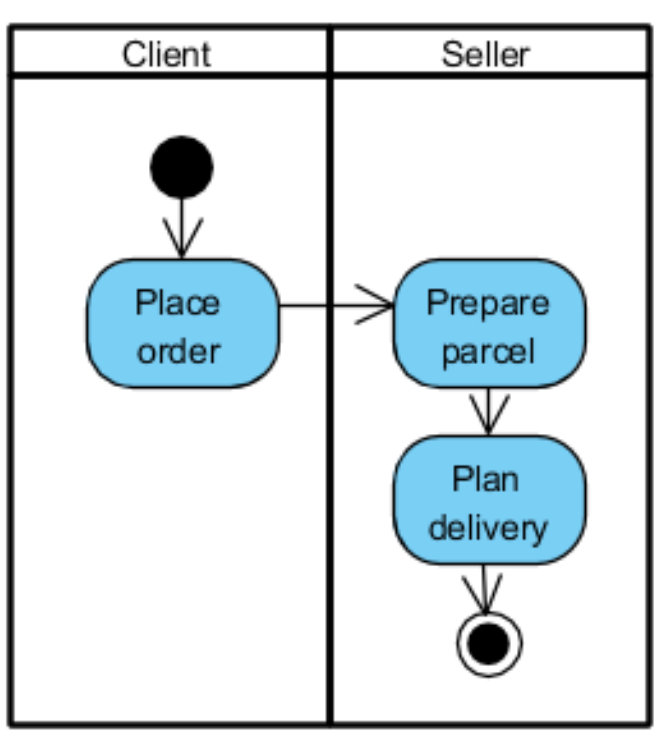
Handling exceptions- Interruptible activity region

- it defines a group of actions whose execution must be completed immediately if a certain event occurs. In this case, another behavior is performed



Partitions

- They are swim lanes that show who or what is responsible for the actions in the activity diagram
- They may be horizontal or vertical
- Partition separation can be made according to organizational units, responsibilities, etc.
- They achieve a better structuring of the diagram



Good practices – activity diagram

✓ Place the **initial node** in the upper left **corner** of the chart for better readability

✓ Every diagram must have **at least one initial node and one final node.**

✓ Check the correctness of ‘**black hole**’ activities, which have an input flow but no output flow. Most likely you missed a transition.

✓ Check the correctness of '**miracle' activities**, which have output flows but no input streams. Only the starting activity can have such a behavior.

Good practices – nodes

- ✓ Decision nodes must reflect previous activity.
- ✓ A fork node must normally also have a suitable join node.
- ✓ The decision node has a single input flow.
- ✓ The merge node has a single output flow.
- ✓ The fork node has a single input flow.
- ✓ The join node has a single output flow.

Good practices – conditions

✓ Each output flow from a decision node must have a condition.

✓ The conditions **must not overlap**. For example: x <= 0 and x> = 0. It is not clear which path will be followed if x is 0.

✓ The conditions must be complete. For example: x <0 and x> 0. It is not clear which path will be followed if x is 0.

✓ Use [**otherwise**] conditions to cover all conditions not explicitly specified.

Good practices – partitions

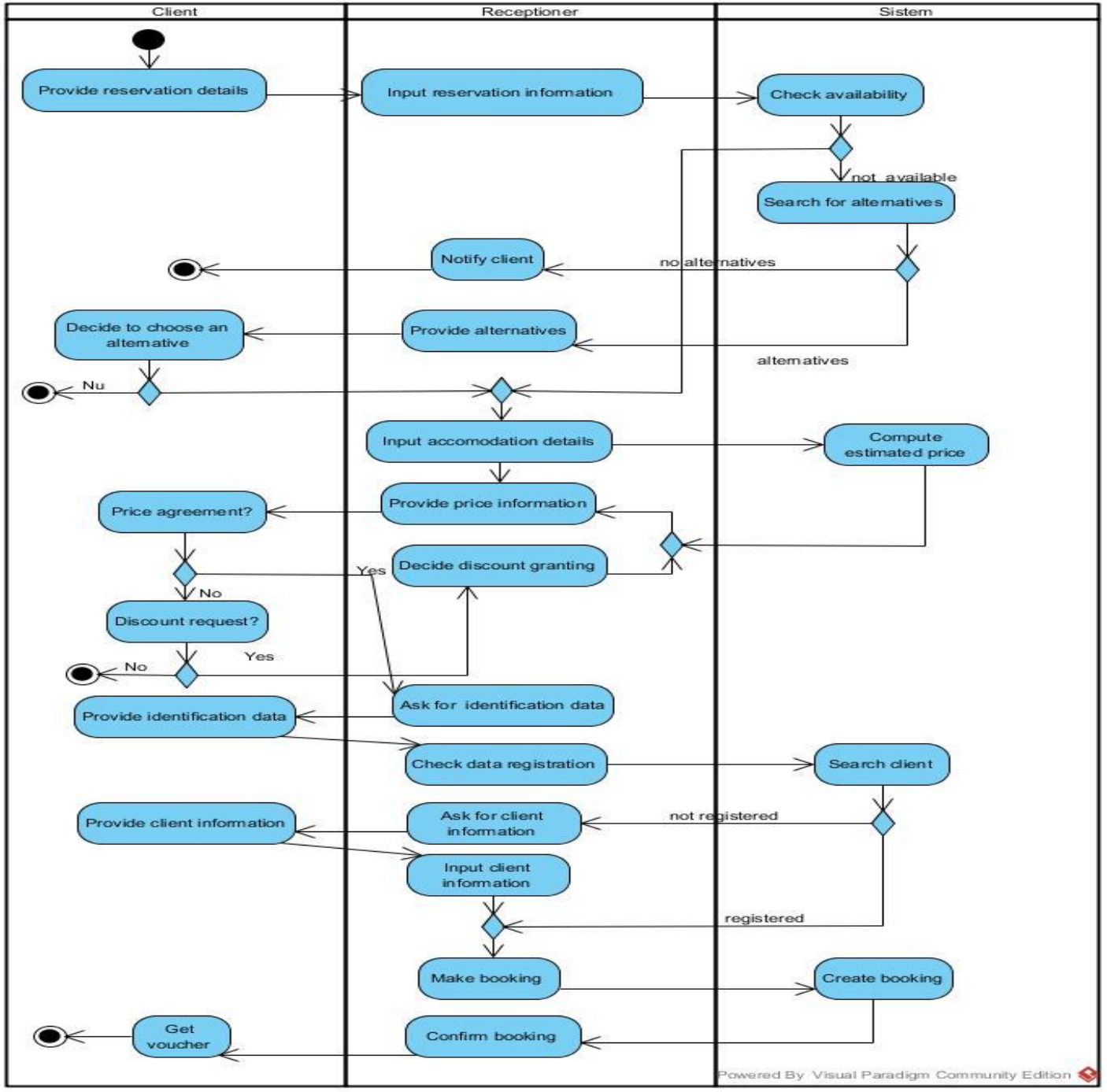
- ✓ Order the partitions in a **logical** way.
- ✓ Use partitions when modeling **linear processes**, not cyclic processes.
- ✓ Try to avoid including more than **five partitions** in a chart.
- ✓ Use **horizontal** partitions to model **business processes**.

Seminar work

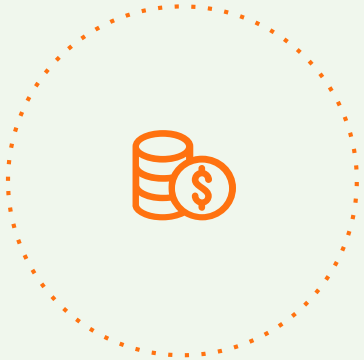
Create the general use class diagram and the textual description for a use case,for the scenario below

The project goal is to develope a software application for the management of a hotel business unit. In order to check in, a customer can request to reserve one or more rooms by e-mail or telephone. For this, he provides the receptionist with information on the period of accommodation and type of rooms required. Customers will get discounts if they reserve at least 3 rooms or if the period of accommodation exceeds 5 days. The receptionist checks availability and notifies the client of this and the estimated cost of accommodation. If there are no rooms available as requested, the receptionist can provide alternatives to the customer. The client may request a discount (additional or not) and the receptionist will decide the feasibility discount, assisted mandatory by the hotel manager. If the client agrees with the proposed price, they proceed to the reservation. For new customers, the receptionist asks identification data, which he introduces in the application.

Once at the hotel and if it has made a prior booking, the customer will provide his identification and / or booking number and the check in is finalized. If there is no reservation, the availability for the required period will be checked. When there is such a room, accommodation is made. At the end of the stay, the receptionist prepares a list of all the services used by the customer and their price. The list must be validated by the customer, then the final invoice is drawn up. The invoice can be paid partially or fully by bank transfer, cash or using a credit card. Also, before leaving the hotel, the customer is asked to complete a form to evaluate the services provided by the hotel premises

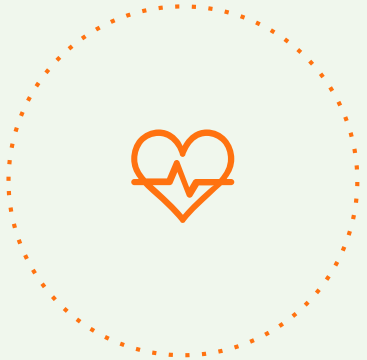

Client
Receptioner
Sistem
Provide reservation details
Input reservation information
Check availability
not available
Search for alternatives
Notify client
no alternatives
Decide to choose an alternative
Provide alternatives
alternatives
Nu
Input accomodation details
Compute estimated price
Price agreement?
Provide price information
Yes
Decide discount granting
No
Discount request?
Yes
No
Ask for identification data
Provide identification data
Check data registration
Search client
Provide client information
Ask for client information
not registered
Input client information
registered
Make booking
Create booking
Get voucher
Confirm booking
Powered By Visual Paradigm Community Edition

Studiu individual



Comerț electronic

Învățământ universitar



Servicii medicale

Realizați o diagramă de activitate pentru una dintre cele trei categorii de sisteme

Information system design

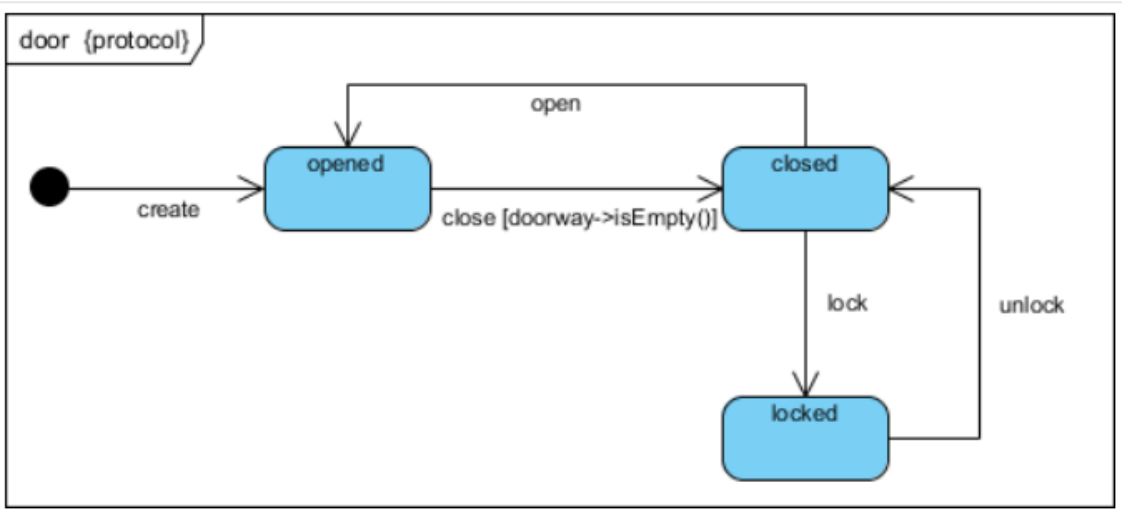
UML State machine diagram

UML Interaction diagrams

State-machine diagrams

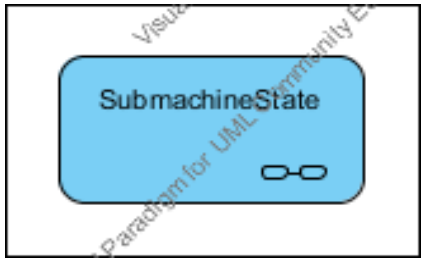
State machine diagram

- It models the dynamic state of a particular object (instance).
- According to UML, a state is „a condition during an object's life when it satisfies some criterion, performs some action, or waits for an event".
- It identifies the events that determine the transition of an object from one state to another state.
- Not all events are applicable in the context of all states. There are conditions that may cause the occurrence of a particular event.



States

- **State**- represented by a rectangle with rounded corners.
- **Composite state** – It is a state that contains sub-states (embedded states).
- **Sub-machine state** – it is used to reuse part of a state diagram into other state diagrams; the behavior is activated as a subroutine call in programming

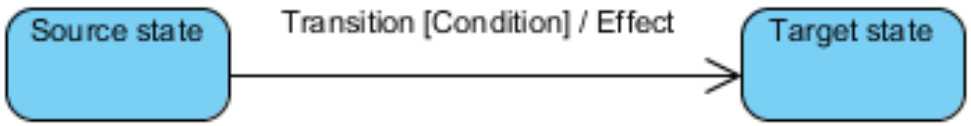


States

- **Initial state** 
  - Pseudostate (the object cannot remain in that state)
  - It signifies the beginning of an object's life
- **Final state** 
  - Real state
  - It signifies the end of an object's life
  - The object can remain in this state forever
- **Terminate node** 
  - Pseudostate
  - It finishes a state machine
  - The modelled object ceases to exist (= it is deleted)

Transitions

- **The object transits** from one state to another **when an event occurs** and **when certain conditions are met**.
- **Transition –** represented by an arrow from an source state to a target state.
- It may contain:
  - ***Trigger***: it is the cause of transitions, and it may be an event, a condition change or the time passing.
  - ***Condition***: A condition that must be true for the trigger to determine the transition.
  - ***Effect***: Action that will be invoked by object as a result of transition.

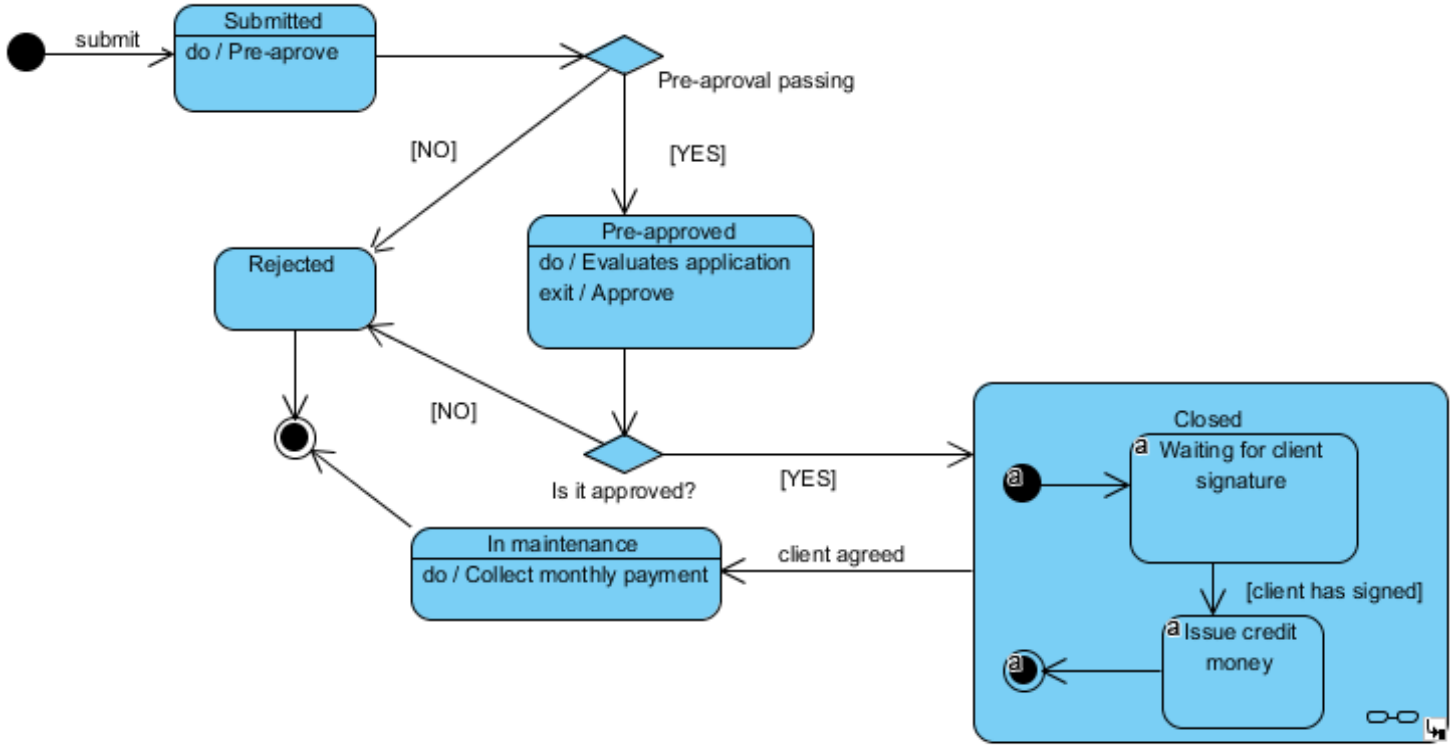


Activities

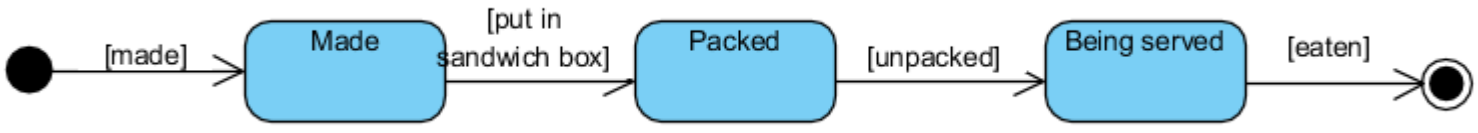
- Except for the initial and the final state, every state has a name, state attributes, performed actions and activities.
- Special activities are:
  - **Entry** – activity taken when entering into a state.
  - **Exit** – activity taken when leaving a state.
  - **Do** - activity taken during a state; external events can interrupt Do action.

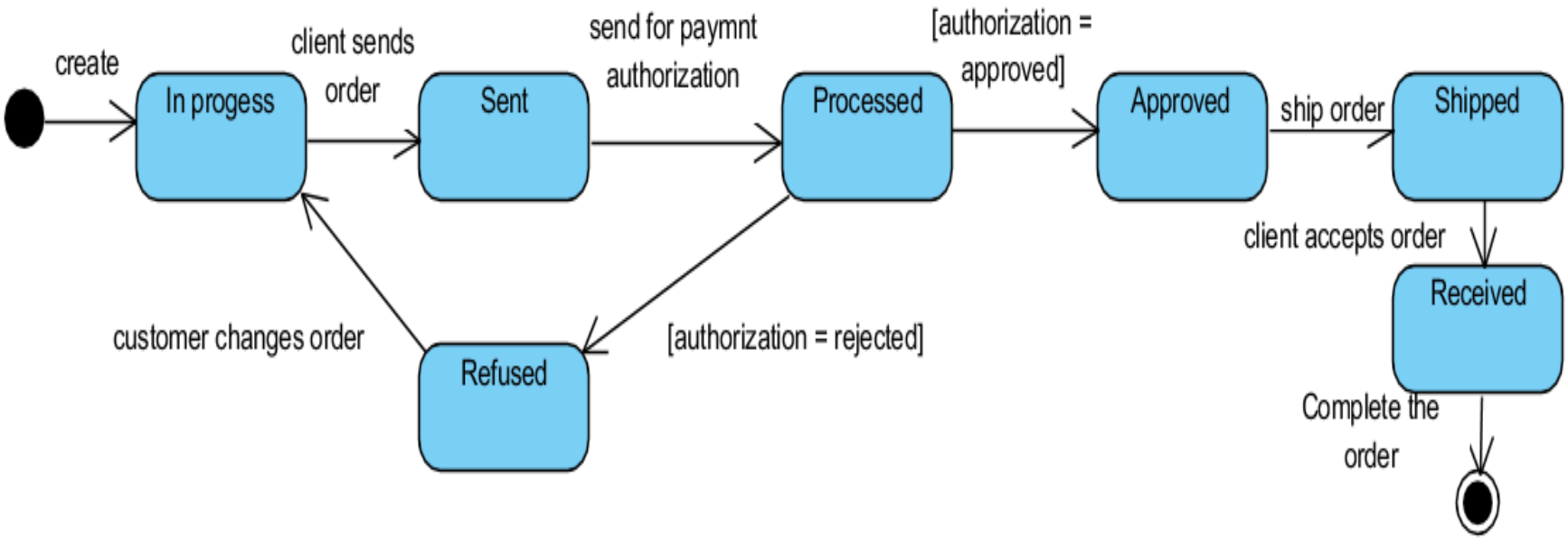
Decisions

- **Decision** (Choice) – it is a pseudo-state that make a conditional fork. It evaluates conditions for the triggers of output transitions in order to choose only one output transition.



Examples of state machine diagrams





Recommendations

- It is recommended to create a machine diagram with states for objects whose behavior changes depending on the state of the object. No such diagram should be created for an object whose behavior is always the same regardless of its state.
- To follow the left-to-right and top-down reading conventions, the initial state must be drawn in the upper left corner of the diagram and the final state must be drawn in the lower right part of the diagram.
- State names must be simple, intuitive, and descriptive. These will be called by an adjective or by an expression that designates a characteristic.
- All mutually exclusive conditions need to be mutually exclusive. It is necessary to check the coverage of all possible situations.
- All transitions should be associated with an event. Otherwise, the state of the object could never change.

Create the state diagram for the ROOM class.

The project goal is to develope a software application for the management of a hotel business unit. In order to check in, a customer can request to reserve one or more rooms by e-mail or telephone. For this, he provides the receptionist with information on the period of accommodation and type of rooms required. Customers will get discounts if they reserve at least 3 rooms or if the period of accommodation exceeds 5 days. The receptionist checks availability and notifies the client of this and the estimated cost of accommodation. If there are no rooms available as requested, the receptionist can provide alternatives to the customer. The client may request a discount (additional or not) and the receptionist will decide the feasibility discount, assisted mandatory by the hotel manager. If the client agrees with the proposed price, they proceed to the reservation. For new customers, the receptionist asks identification data, which he introduces in the application.

Once at the hotel and if it has made a prior booking, the customer will provide his identification and / or booking number and the check in is finalized. If there is no reservation, the availability for the required period will be checked. When there is such a room, accommodation is made. At the end of the stay, the receptionist prepares a list of all the services used by the customer and their price. The list must be validated by the customer, then the final invoice is drawn up. The invoice can be paid partially or fully by bank transfer, cash or using a credit card. Also, before leaving the hotel, the customer is asked to complete a form to evaluate the services provided by the hotel premises

UML Interaction diagrams

Sequence diagrams

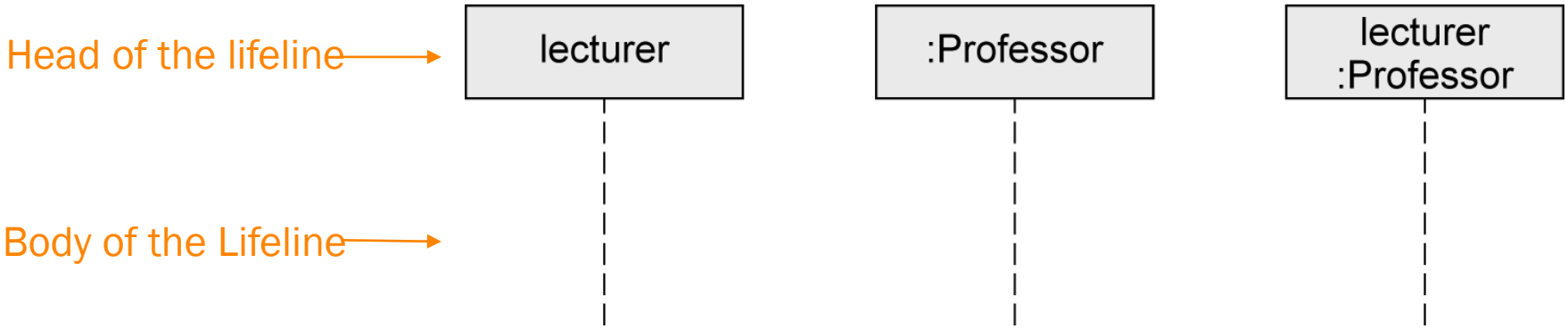
Communication diagrams

The role of interaction diagrams

- Model the dynamic aspects of the system.
- They consist of a set of objects and relationships between them, including messages that objects send to each other
- Two types of interaction diagrams are presented: sequence diagram and communication diagram (called collaboration diagram in UML 1.4).
- The two diagrams are equivalent in terms of semantics and can transform from one to the other.

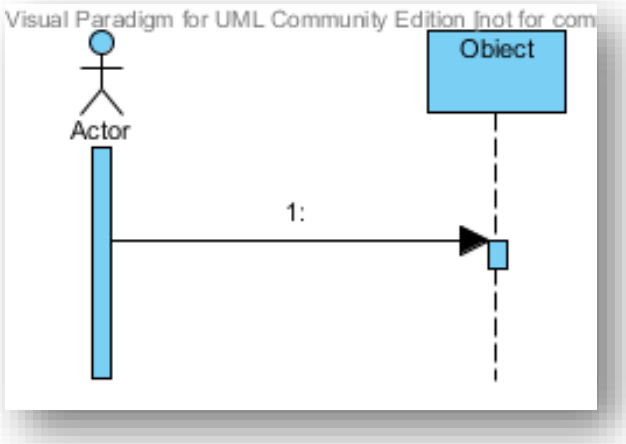
Sequence diagram

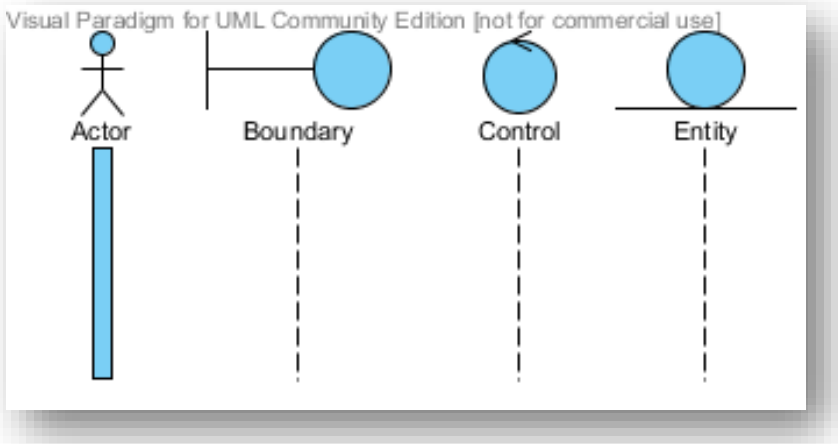
- It's an interaction diagram made up of objects, messages exchanged between them and the **temporal dimension** represented progressively vertically.
- It underlines the sequence of messages according to time
- **The objects** are placed in the upper part of the diagram, along the X-axis, from left to right
  - They are arranged in any order that allows the simplification of the diagram.
  - Typically, objects starting the interaction are placed to the left and objects that follow to the right
  - The existence of objects is illustrated by their lifelines



Sequence diagram - objects

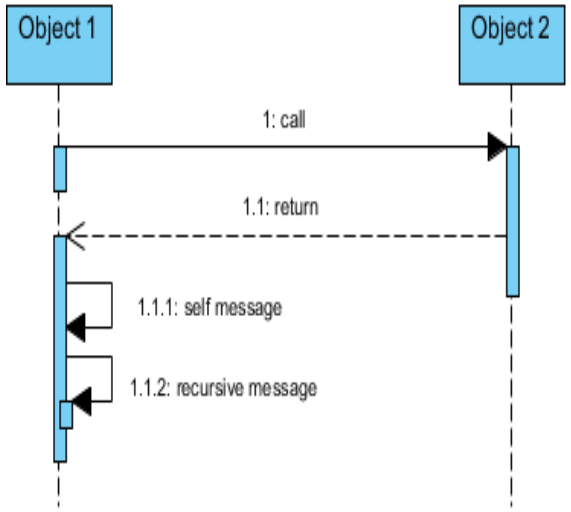
- *Lifeline:* vertical line that represents the existence of an object over a period of time. Most of the objects that appear in the diagram exist through the duration of the interaction, with the lifeline drawn from the top of the diagram to the bottom. Other objects can be created during the interaction.
- *Activation :* a tall, thin rectangle indicating the period of time the object performs an action. The upper end of the rectangle is aligned with the beginning of the action and the lower end to end of the action
- Objects can be represented using the following stereotypes: actor, boundary, entity and control.





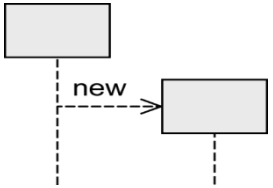
Sequence diagram - messages

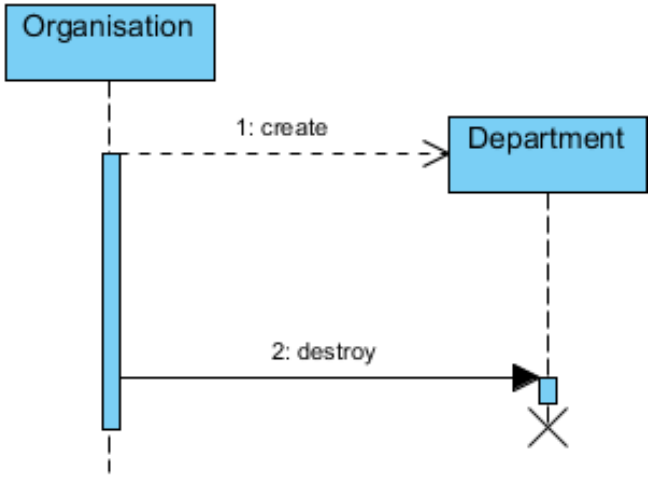
- *The messages* are represented as arrows that start from the lifeline of an object and stop at the lifeline of another object.
- Messages can be of several types and can include parameters:
  - A *call* message represents a request of the object that sends the message to the object that receives the message. The request implies that **the receiver will perform one of its operations**.
  - The sender waits for the receiver to execute the operation and to receive an answer from it. (**Reply**)
  - **Asynchronous message** - the sender continues its activity without waiting for an answer
  - An object can send messages to itself – **self call**. Such a message can mean a recursive call of an operation or method that calls another method of the same object.



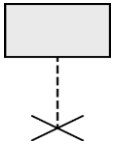
Messages -1

- The *create* and *destroy* messages of an object begin and end the lifeline of an object. These are optional and are used when you want to **explicitly specify these events**.



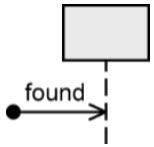


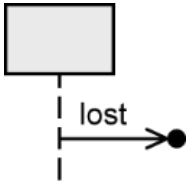
- A *destroy* message can generate subsequent destruction of objects that it contains through composition. After destruction, an object can not be created again on the same lifeline.

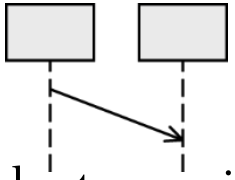


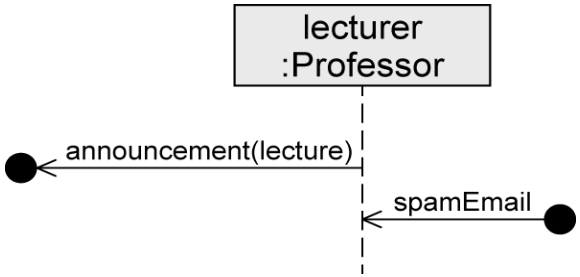
Messages -2

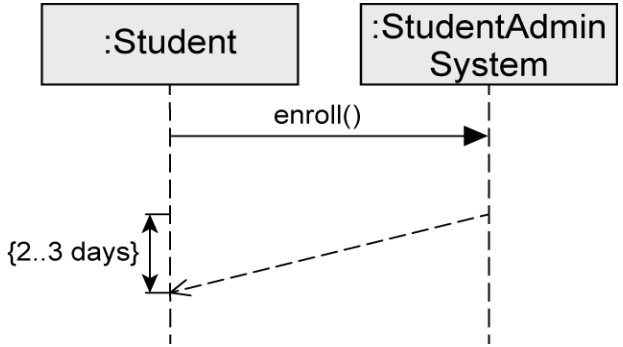
- **Found message**
  - Sender of a message is unknown or not relevant
- **Lost message**
  - Receiver of a message is unknown or not relevant
- **Time-consuming message**
  - "Message with duration"
  - Usually messages are assumed to be transmitted without any loss of time
  - Express that time elapses between the sending and the receipt of a message







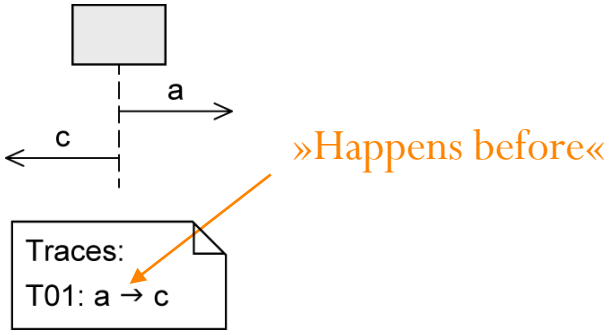




Messages - 3

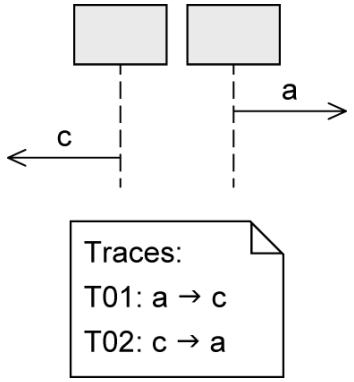
Order of messages:

… on one lifeline

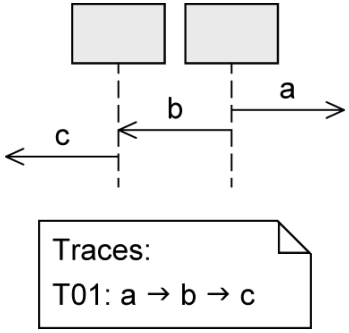


»Happens before«

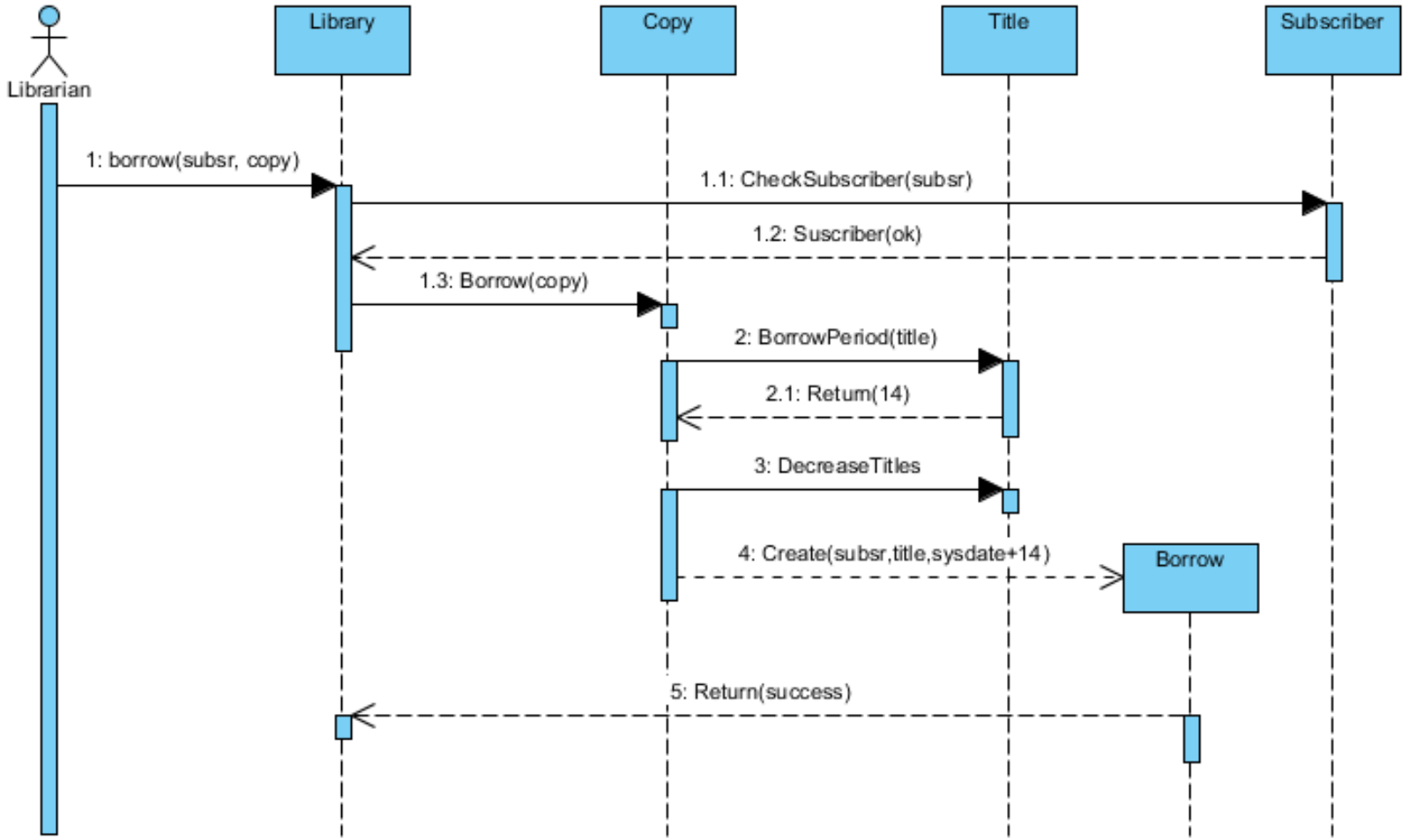
… on different lifelines



… on different lifelines which exchange messages

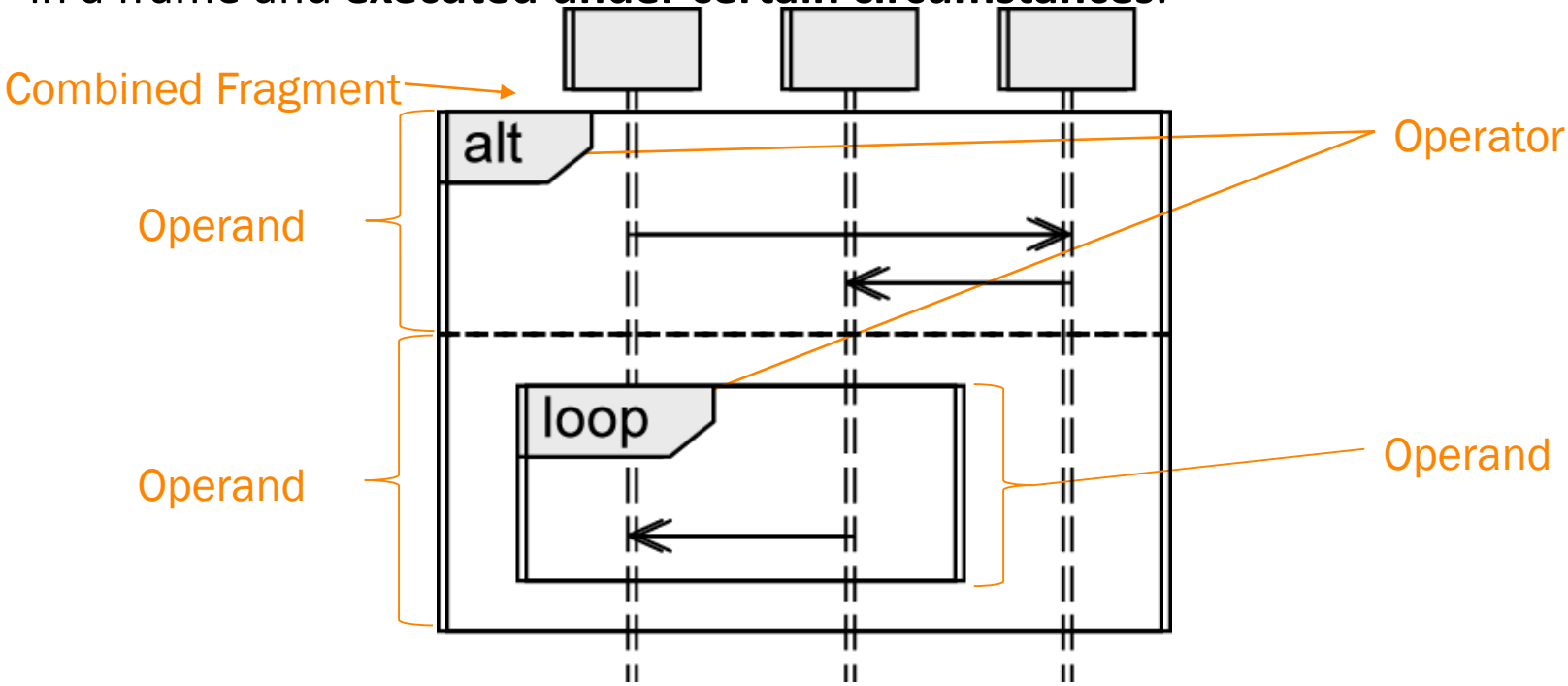


Sequence diagram - example



Combined Fragments -1

- There are mechanisms that allow the addition of a certain level of **procedural logic** in the diagrams through **combined fragments**.
- A combined fragment represents one or more processing sequences included in a frame and **executed under certain circumstances**.

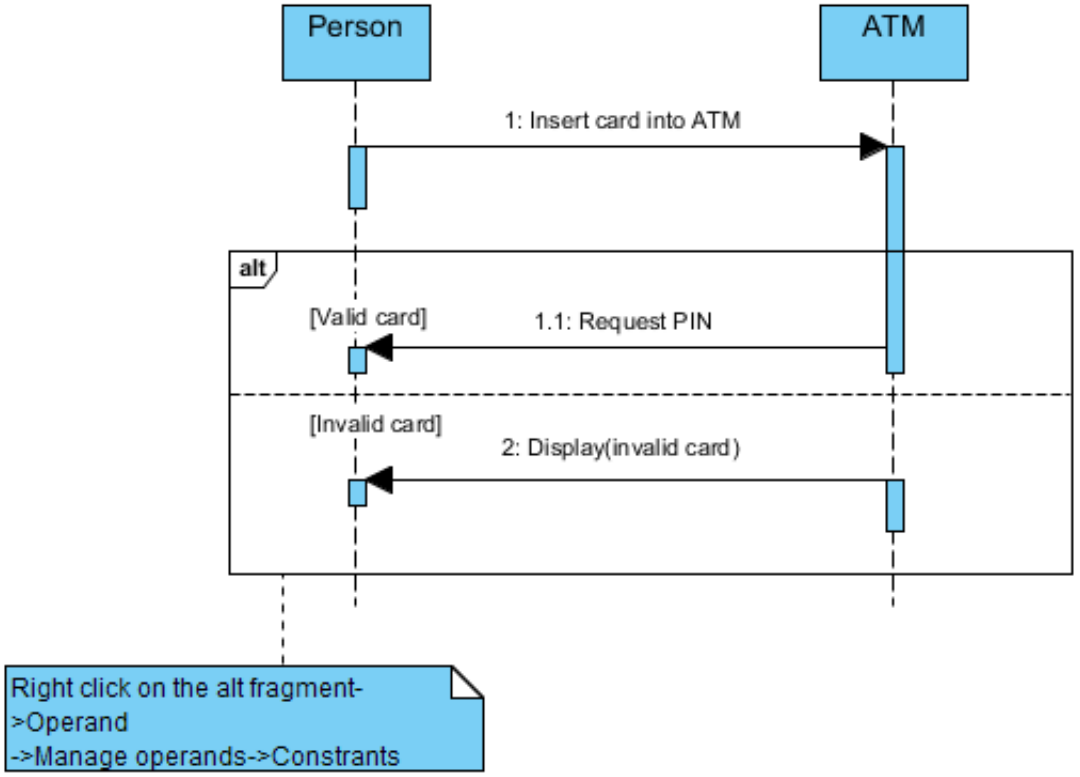


Combined Fragments -2

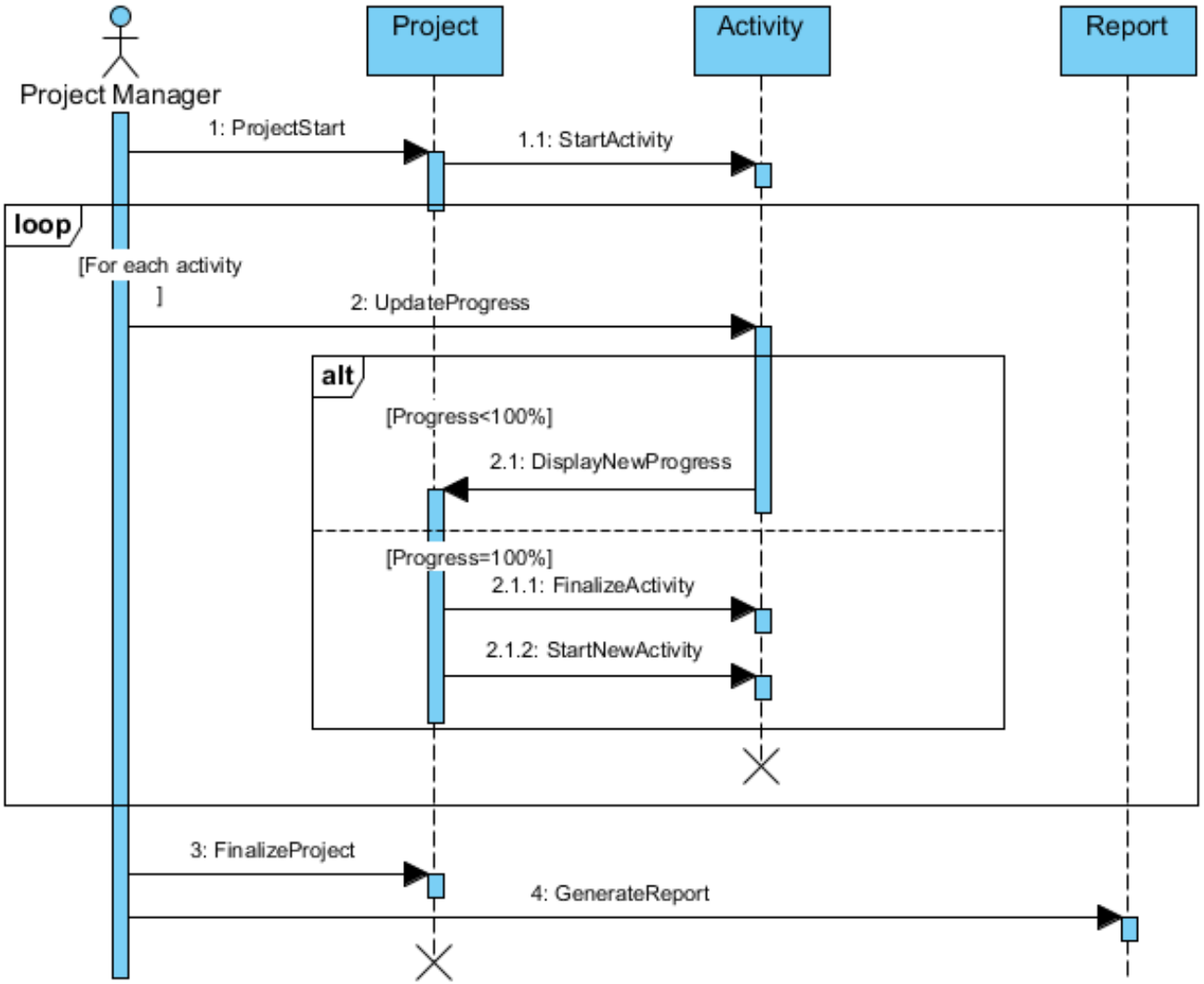
- 12 predefined types of operators
- Most commonly used fragments are:
  - Alternatives (**Alt**) that model 'if..then..else' logic.
  - Repetitives (**Loop**) containing a series of interactions that are repeated several times.
  - Optional (Opt) that model optional interactions

	Operator	Purpose
Branches and loops	alt	Alternative interaction
	opt	Optional interaction
	loop	Repeated interaction
	break	Exception interaction
Concurrency and order	seq	Weak order
	strict	Strict order
	par	Concurrent interaction
	critical	Atomic interaction
Filters and assertions	ignore	Irrelevant interaction
	consider	Relevant interaction
	assert	Asserted interaction
	neg	Invalid interaction

Combined fragments- example



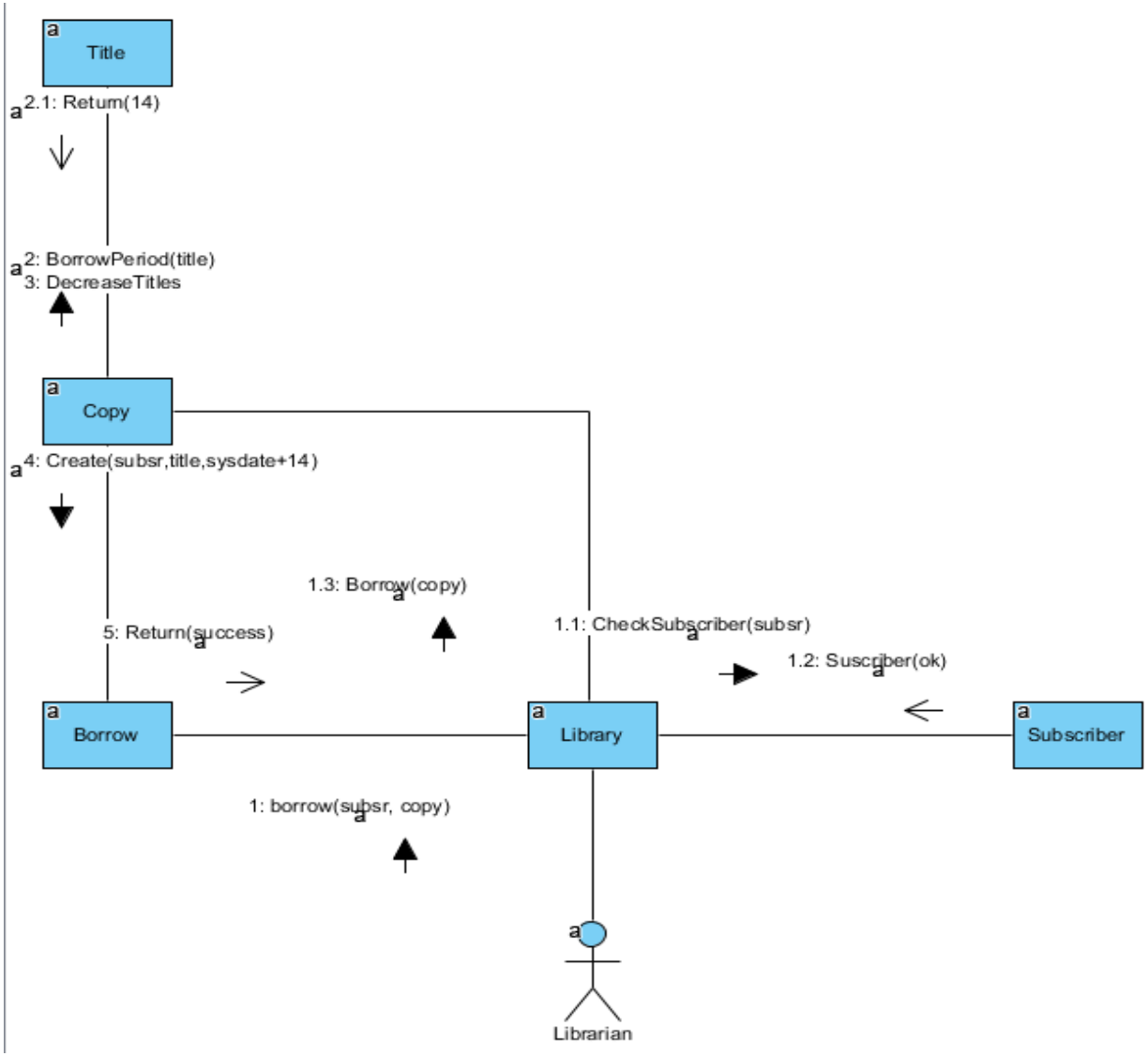
Combined fragments- example



Communication diagram

- **Communication diagram** (**collaboration –** as it is named in UML 1.4) is an interaction diagram that emphasizes the structural organization of objects that send and receive messages
- Graphically, a collaboration diagram is a collection of **vertices** and **edges**
- It represents the same information as a sequence diagram, but **emphasizes the organization of objects** participating in the interaction.
- Objects are placed first, as vertices of a graph, next the connection between objects are drawn as edges of the graph and then the messages the objects receive or send are added to the connections
- To indicate the **order**, the message should be **prefixed with a number** starting from 1 and increasing

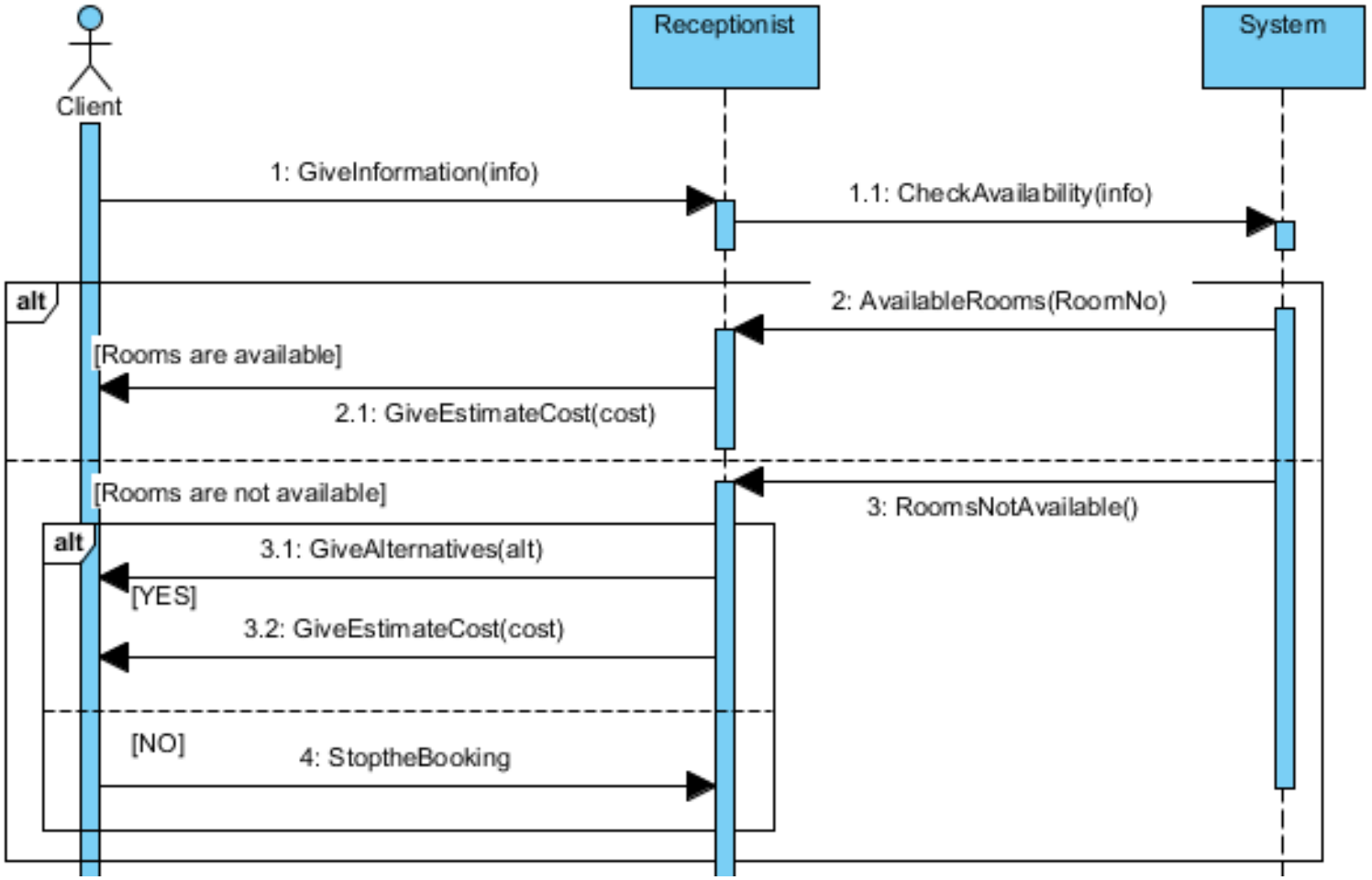
Communication diagram– objects and messages



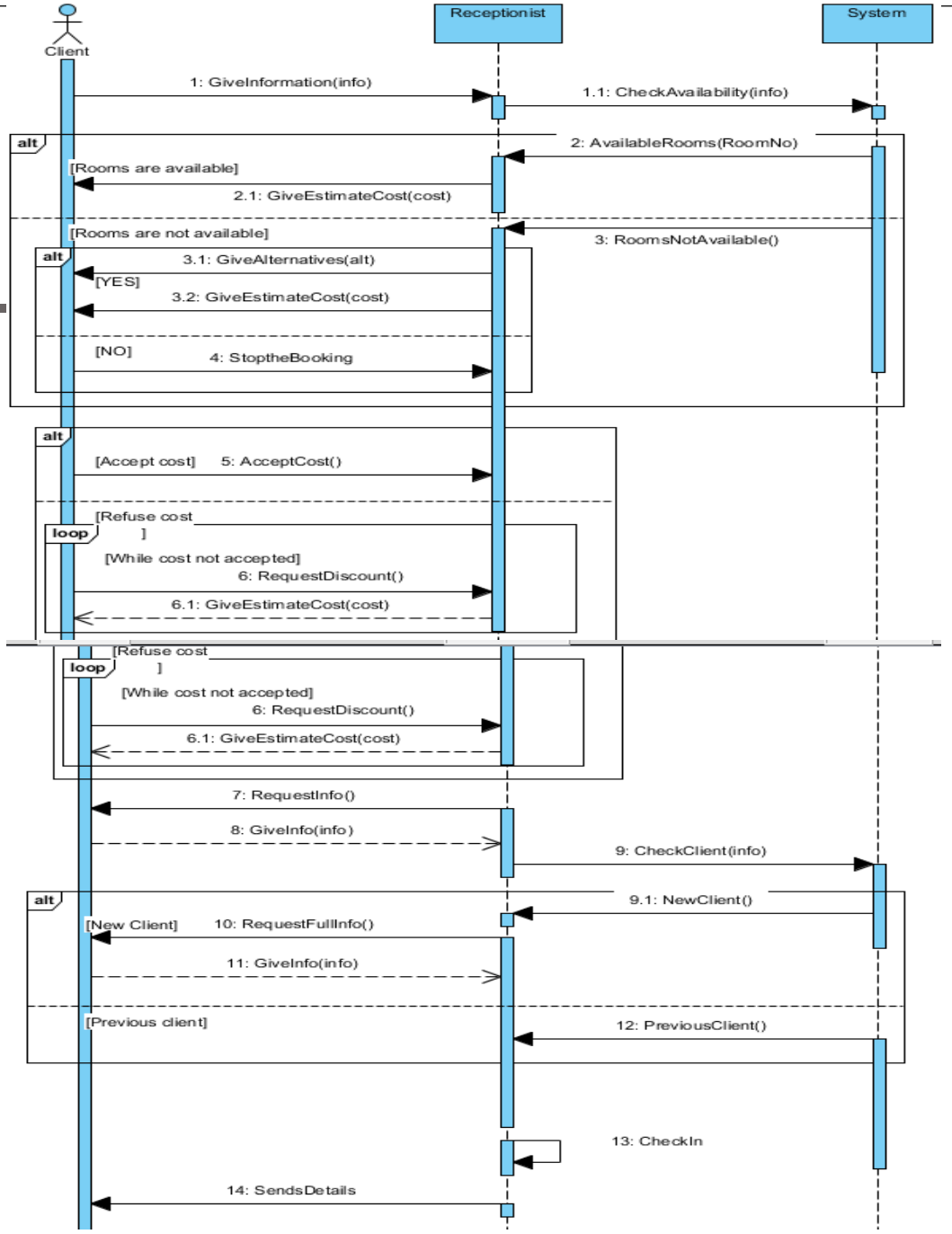
Interaction diagrams

- The two interaction diagrams are equivalent and can be converted one to another without loosing information.
- To convert a diagram to another in Visual Paradigm right-click on a diagram area and select the option Synchronize to Communication / Sequence diagram.
- Communication diagram shows how objects are connected, while the sequence diagram highlights returned messages and temporal order of interactions.

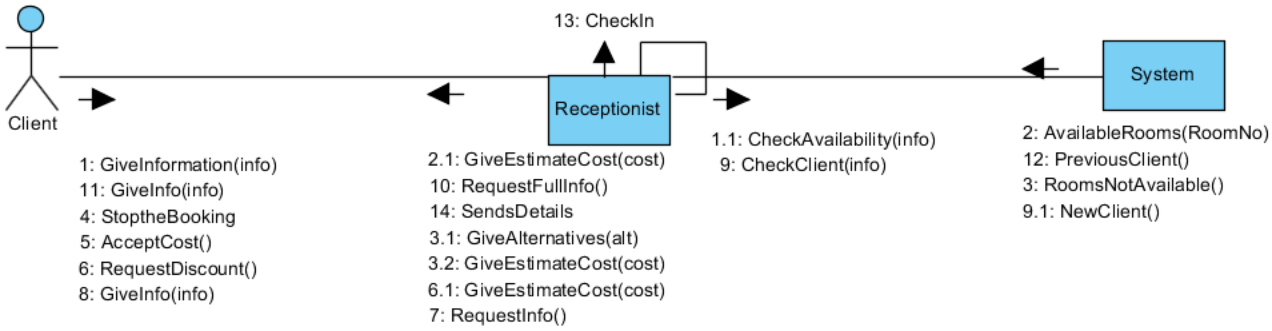
Sequence diagram – hotel example



Sequence diagram - hotel example



Communication diagram – hotel example



Information system design

Seminar 8- BPMN
Business process diagrams

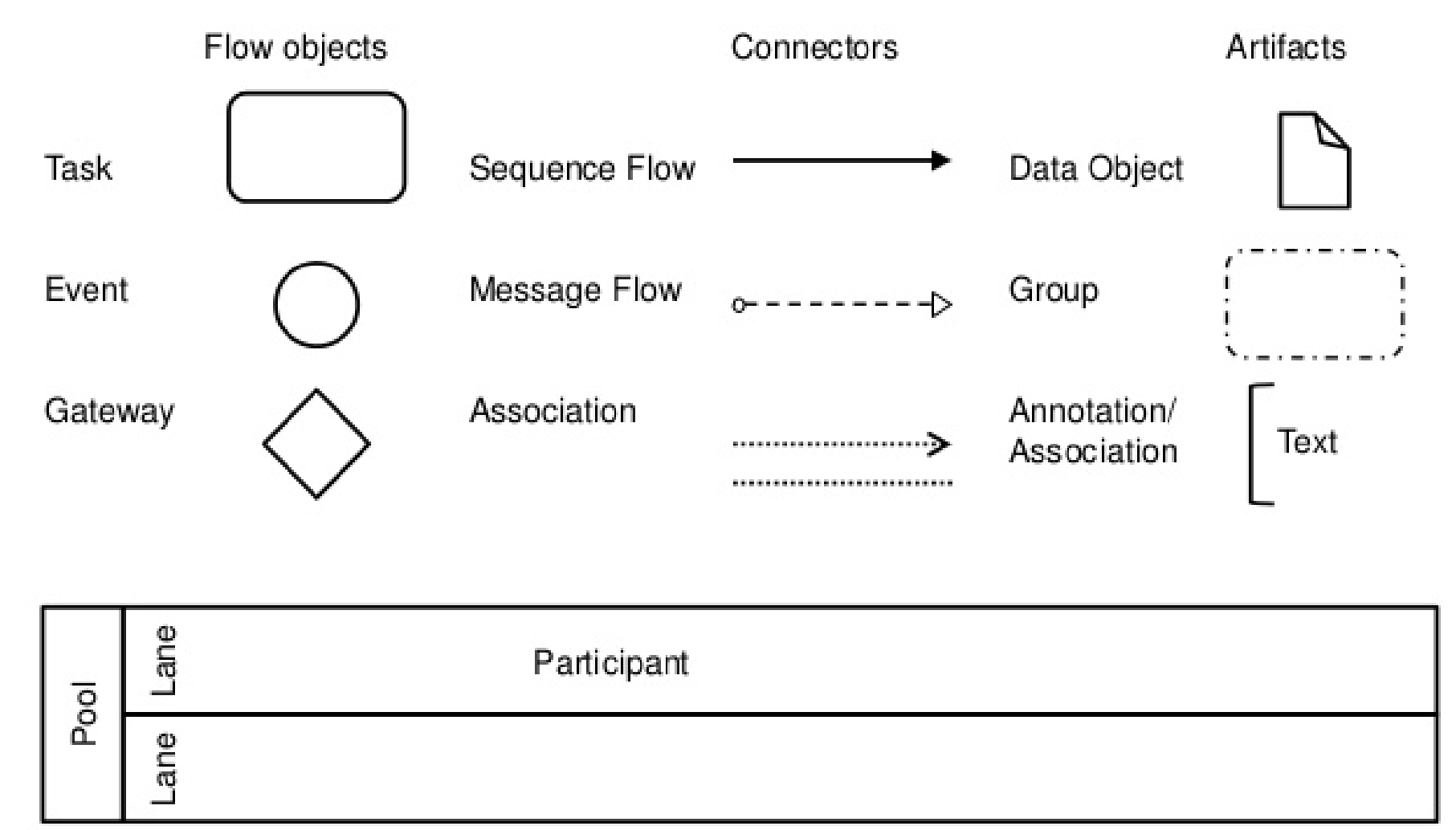
Business Process Model and Notations

Standard created and maintained by OMG, similar to UML language

BPMN models can be created for the **purpose** of identifying, validating, improving or automating processes.

It allows the **optimization** of real world processes through simulation.

BPMN– basic elements(1)



BPMN– basic elements(2)

1. **Flow objects** represent the main elements of process diagrams. They fall into one of the following categories:
   a) **Event**,
   b) **Activity**,
   c) **Gateway**.
2. **Connecting objects** have the role of connecting object flows with each other or with other object types. There are three types of connecting objects:
   a) **Sequence flow**,
   b) **Message flow** and
   c) **Association**.
3. **Swimlanes** – They set subgraphs in the process flow in order to logically separate some parts of it, depending on the **entities involved** in carrying out the process. There are two types:
   a) **Pool** and
   b) **Lane**.

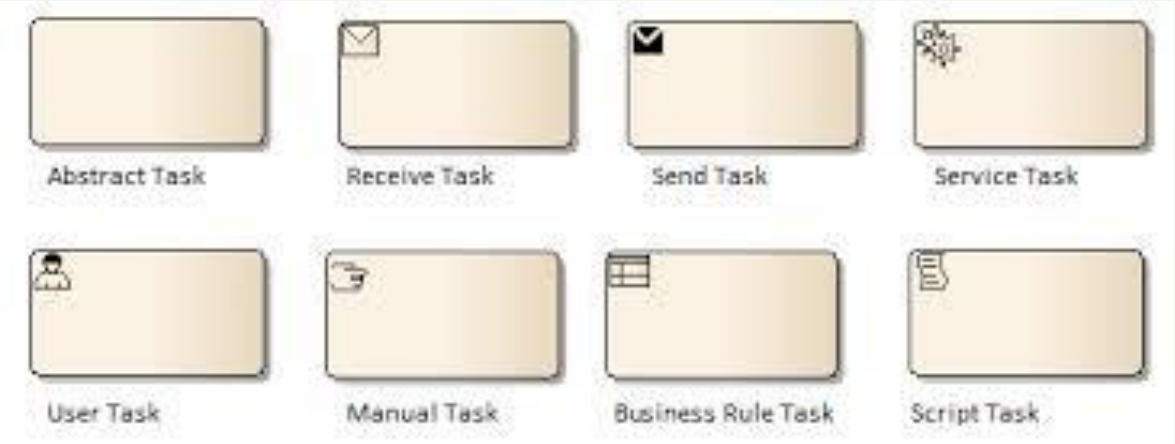
BPMN– basic elements(3)

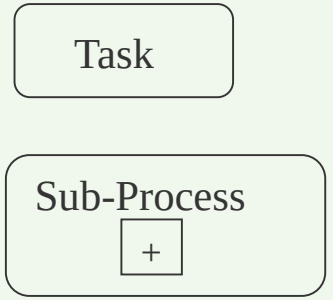
4. **Data objects** are necessary to show data needed in activities or resulted from activities. They fall into four categories:
   i. **Data object**,
   ii. **Data input**,
   iii. **Data output** and
   iv. **Data store**.
5. **Artifacts** - they provide additional information about how documents, data, and other objects are used and updated within a Process. There are two types of standard artifacts:
   i. **Group** and
   ii. **Annotation**,

   but both language and modeling tools provide option of adding any other custom user artifacts necessary to understand the model.

Flow objects: activities

An activity is a generic term for work that a company performs. It can be ***atomic (task)*** or ***compound (sub-process)***

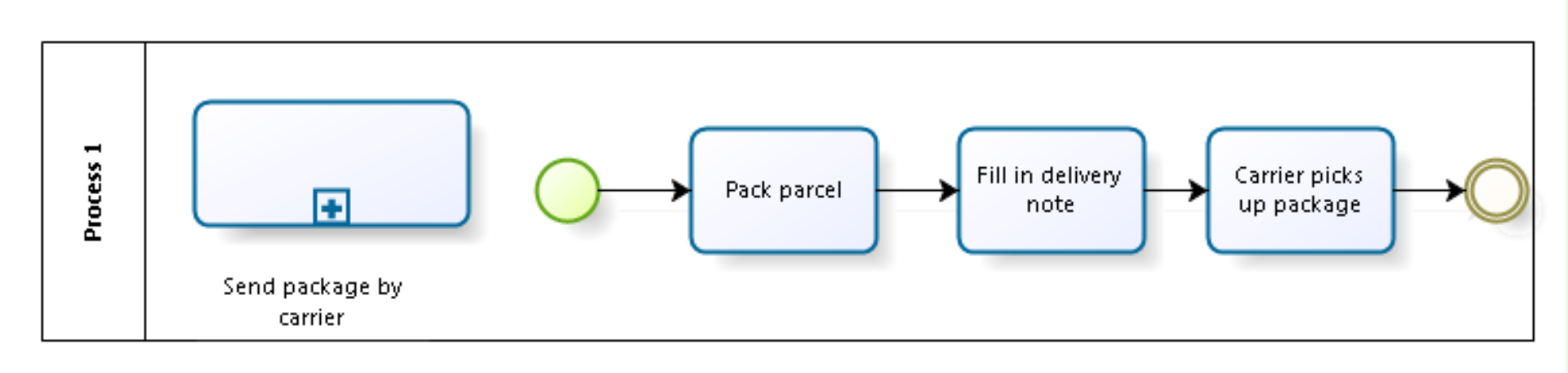




- **Abtract task** – task without any specialization
- **Receive task** – task that waits for a message to arrive from an external participant. After the message is reveived, that task is completed.
- **Send task** – task for sending a message to an external participant. After the message is send, that task is completed.
- **Service task** – task that uses some sort of service, like a web service.
- **User task** – task executed by a human with the assistance of a software application
- **Manual task** – task executed by a human without the assistance of any process execution engine or application.
- **Business rule task** – Provides input to a business rule engine. Reveives output from a business rule engine.
- **Script task** –It is executed by a business process execution engine. When the task is finished.

Sub processes

- They are compound activities included in a process.
- They can be hierarchically nested to any level of detail that is necessary to fully describe a process.
- A sub-process can be collapsed or expanded to show or hide its details
- Each expended description of a sub-process must include start and end events which do not specify a particular behavior.



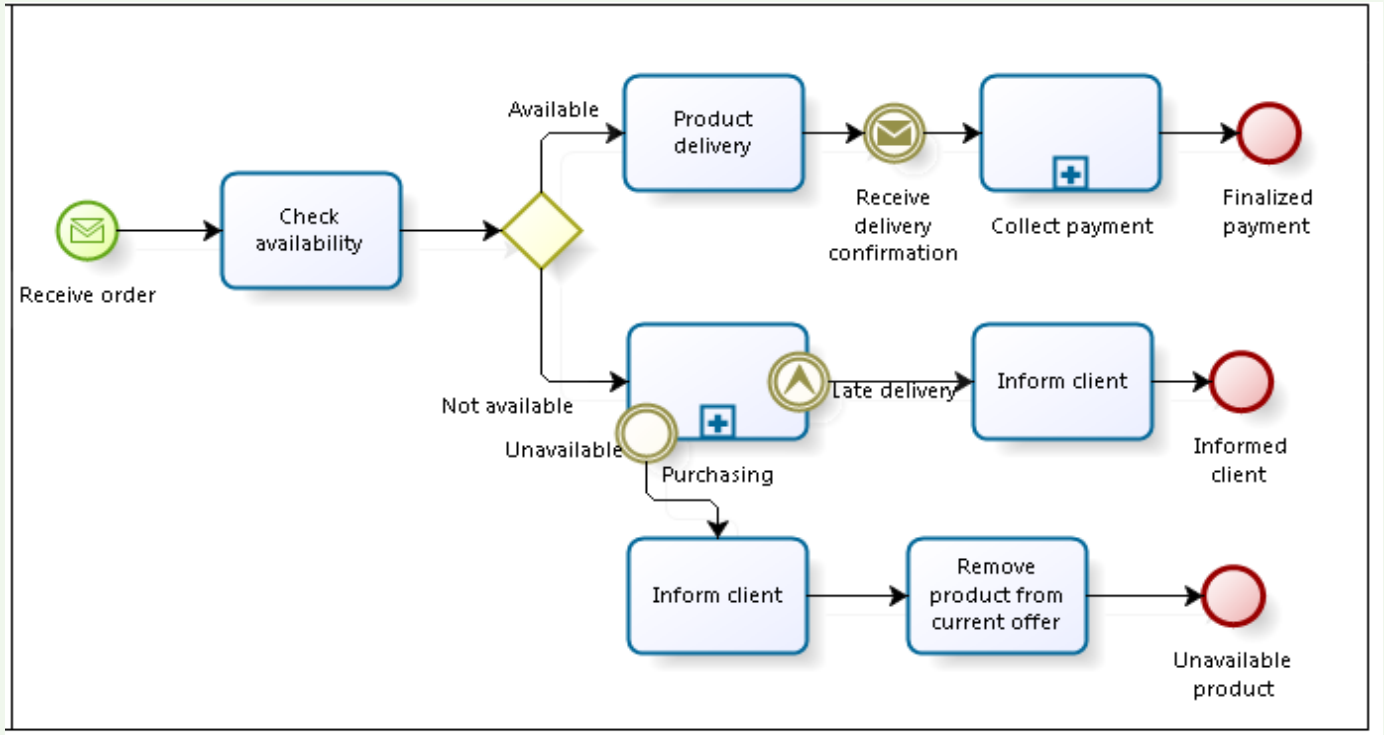
Flow objects: events

Start event

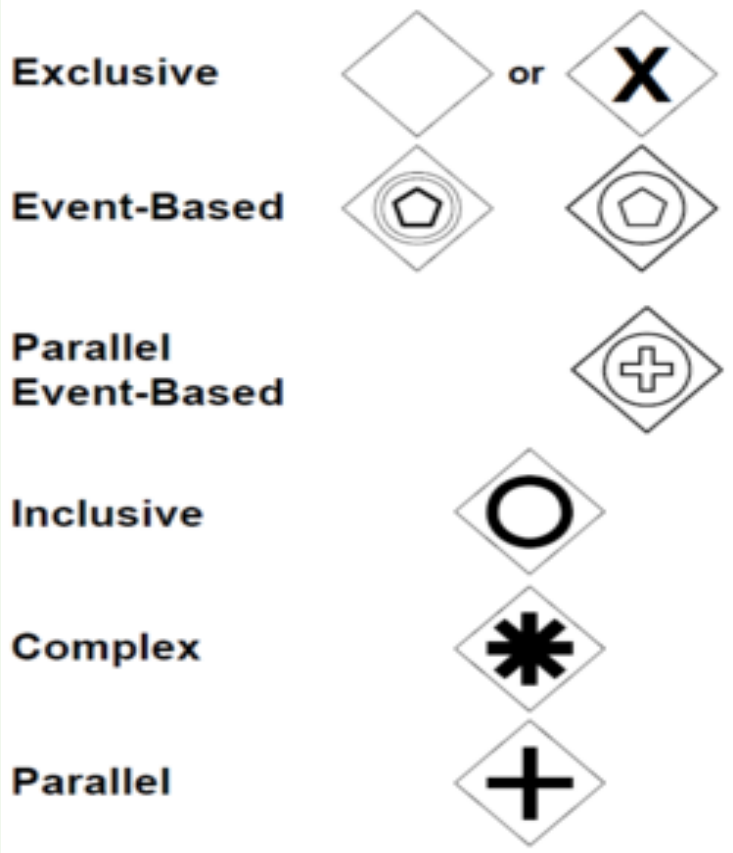
Intermediate event

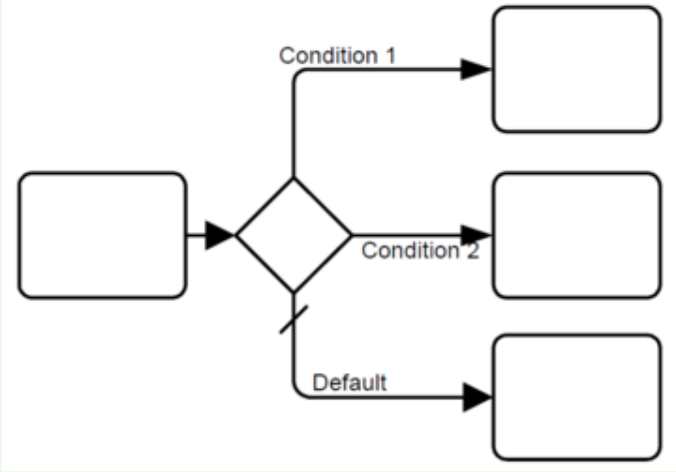
End event

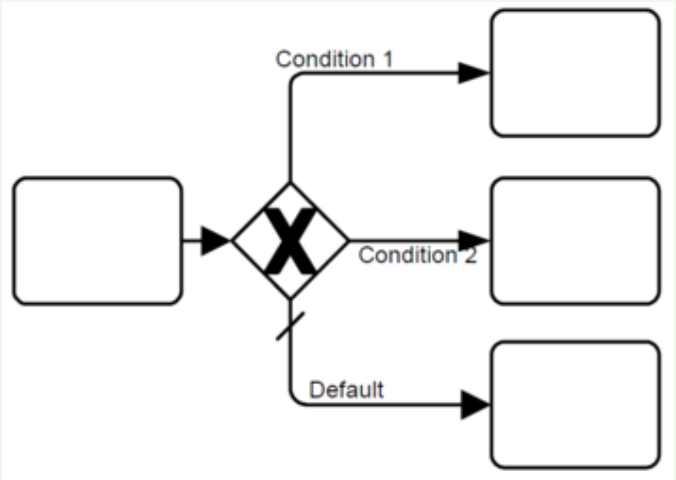
Symbol	Type
	Message
	Timer
	Signal
	Error
	Conditional
	Escalation



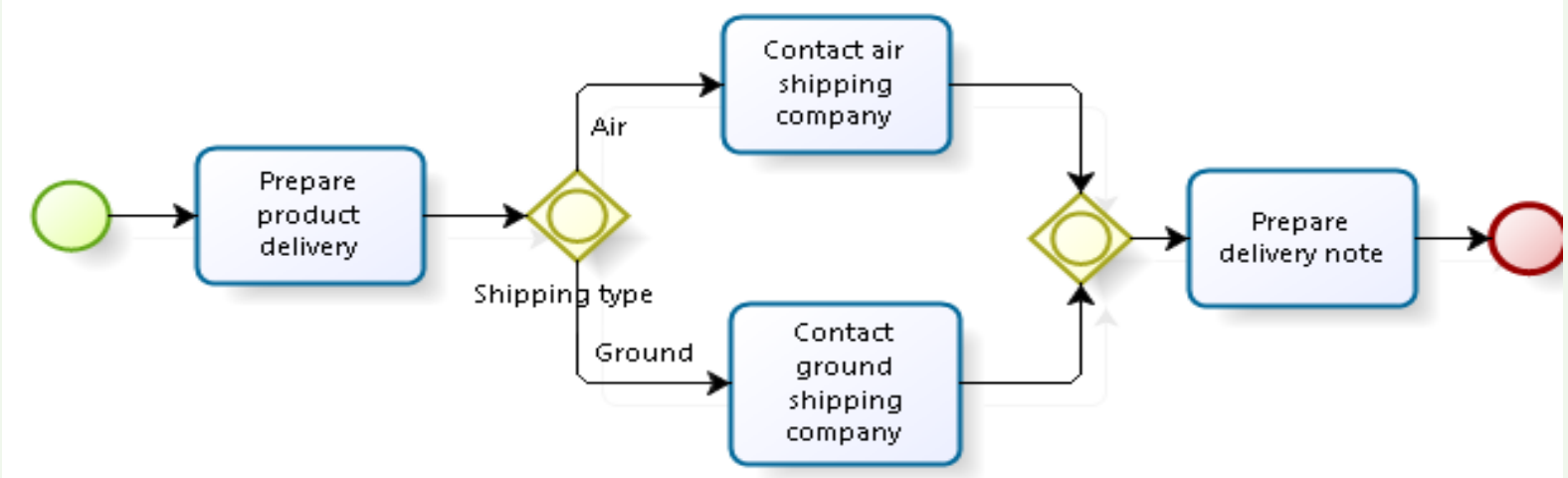
Gateways. Exclusive gateways

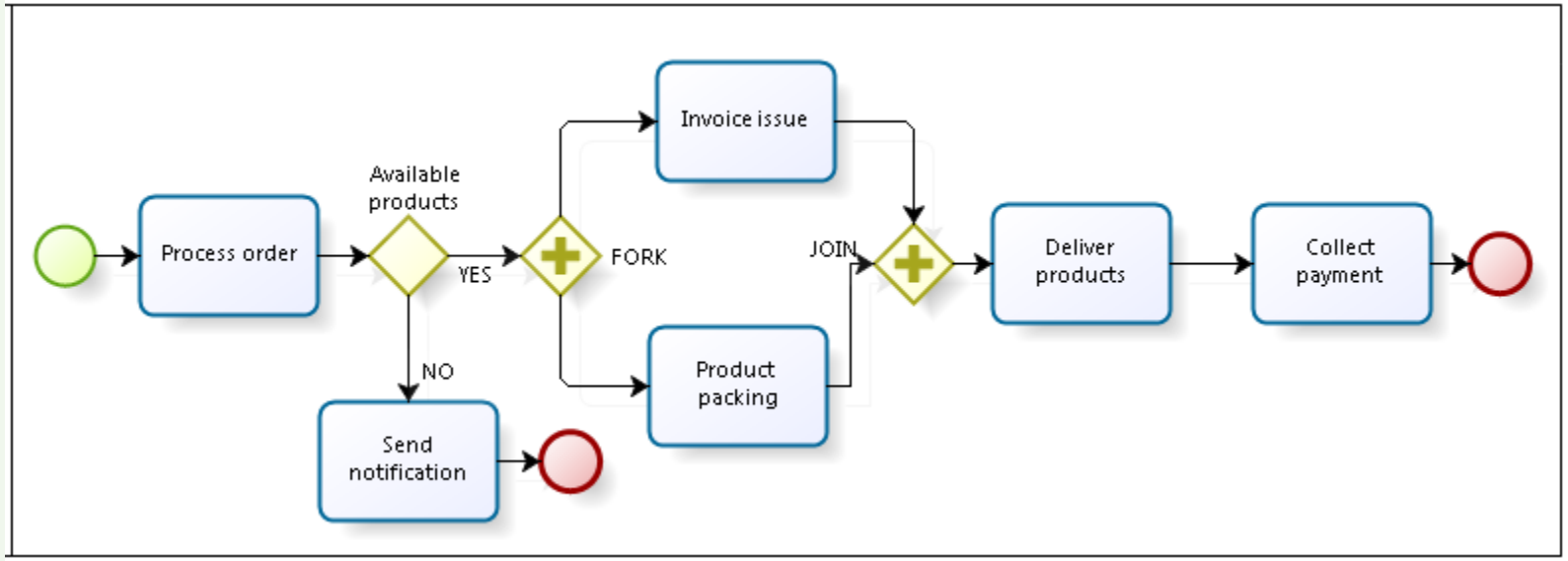




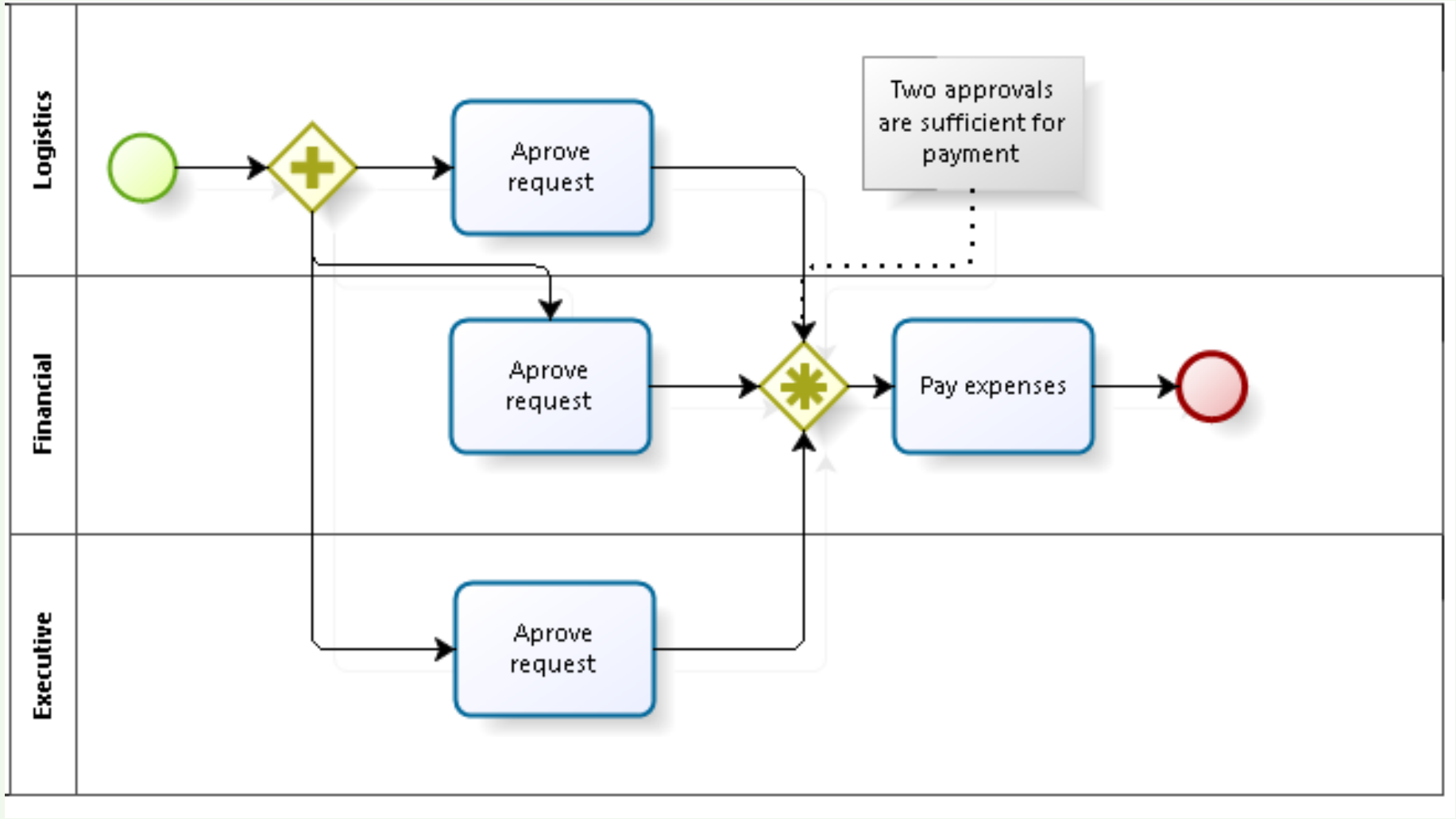


Gateways: inclusive and parallel gateway

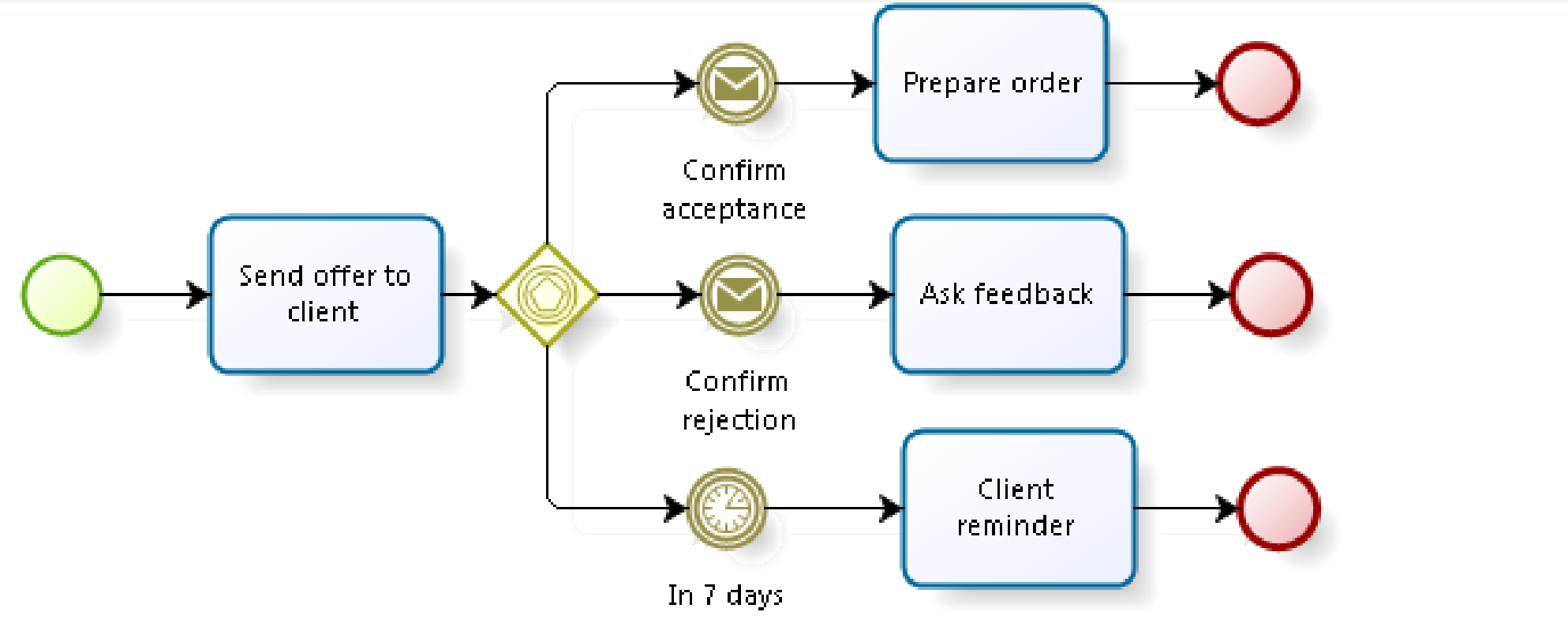




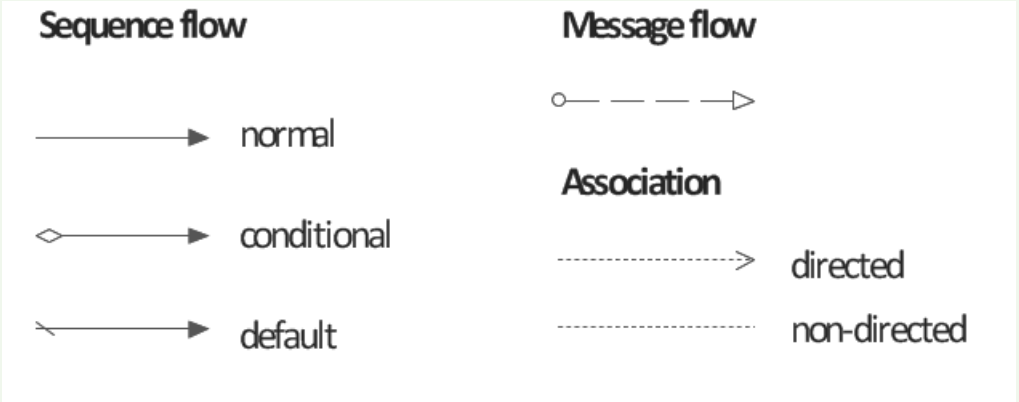
Gateways: complex gateway

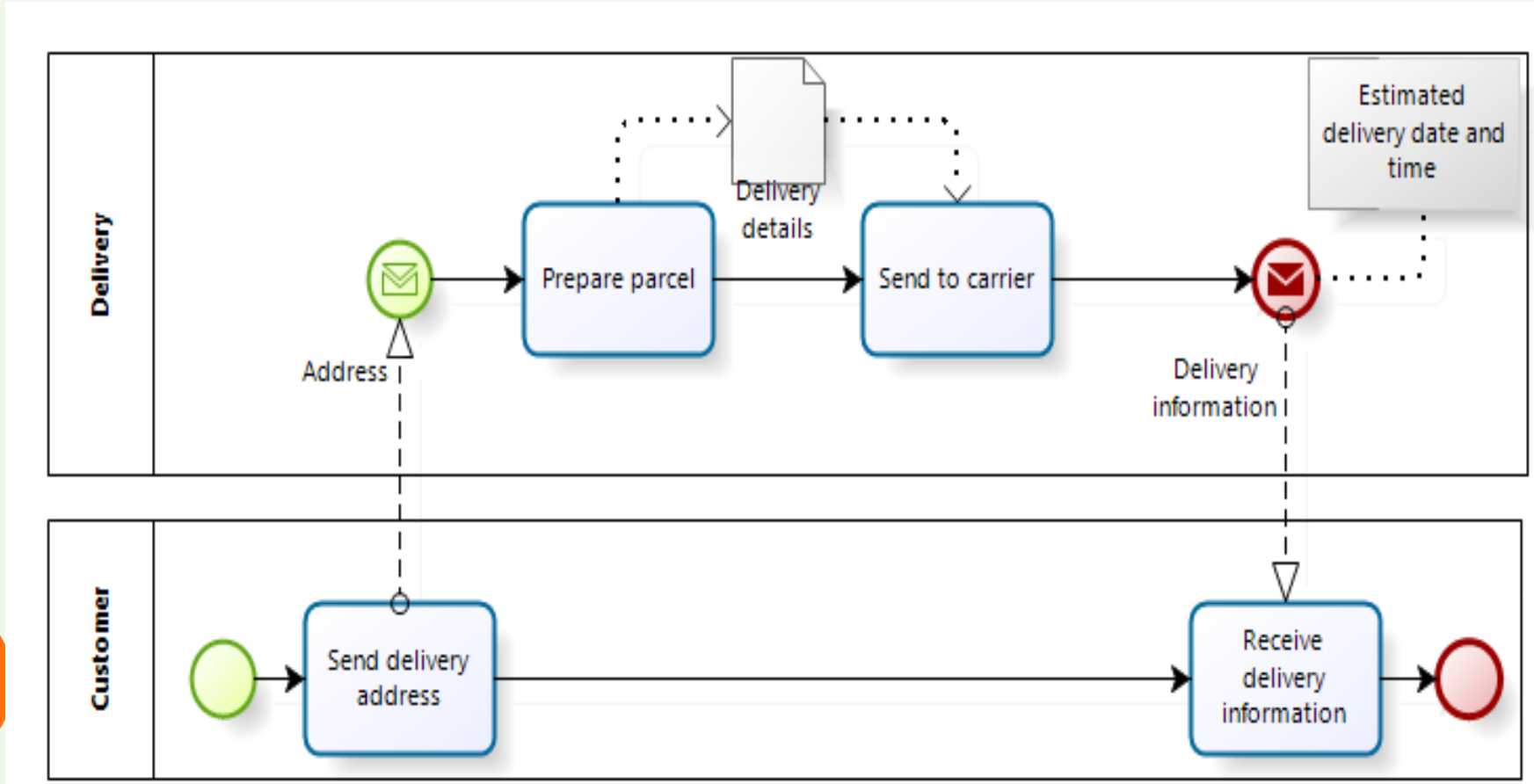


Gateways: event-based gateway

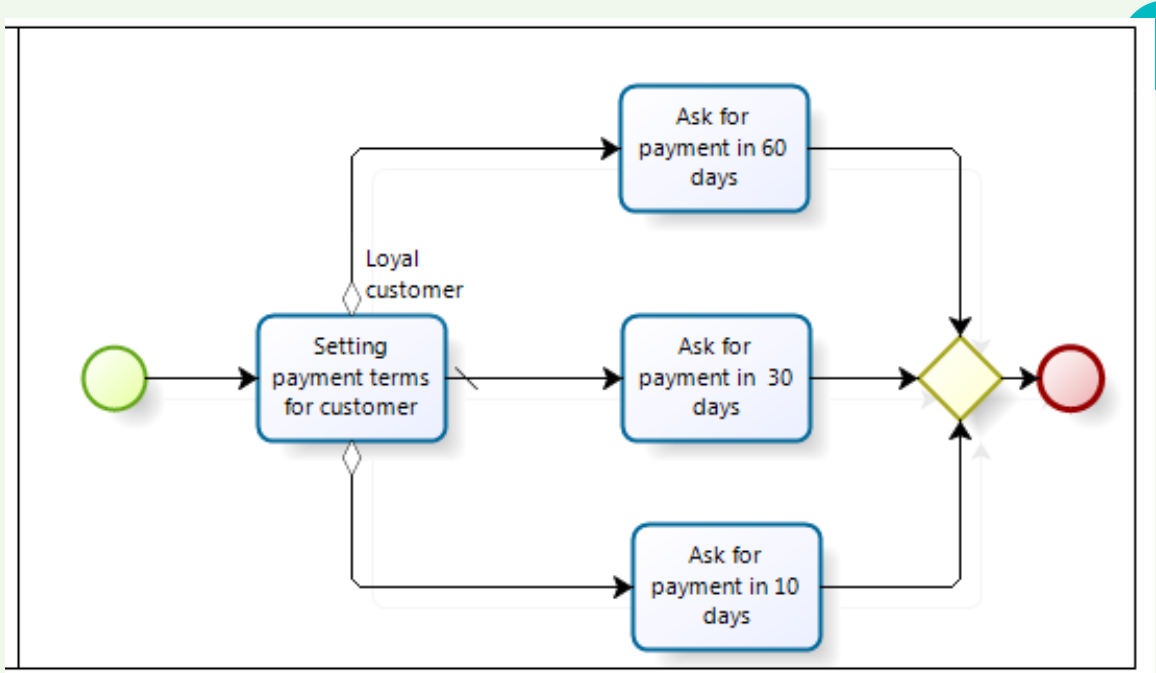


Connecting objects

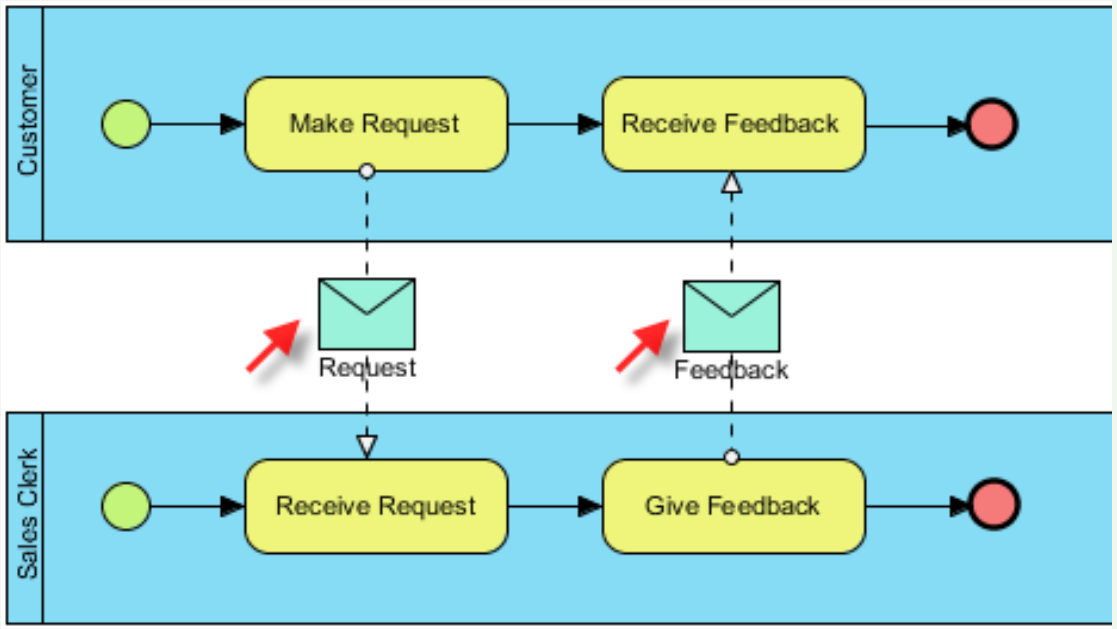




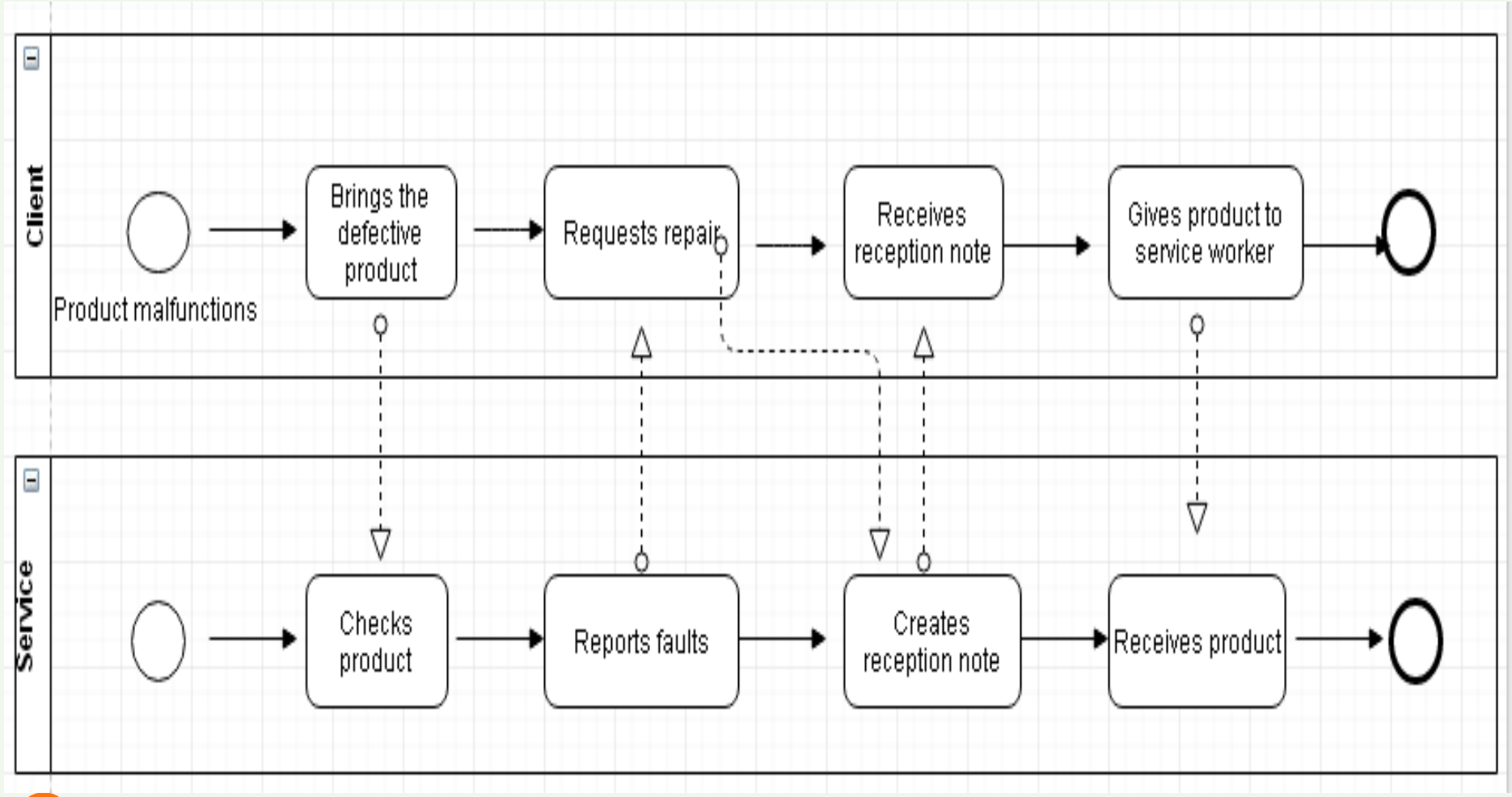
Expression sequence flows



Message flows



Partitioning objects: containers



Good practices

- ✓ The name of an action / subprocess begins with a verb.
- ✓ A gateway does not perform actions - Its role is to assess conditions or wait for events to occur.
- ✓ Sequence flows cannot cross containers - Message flows are used for interaction between participants.
- ✓ Use conditional sequence flows with caution - The set of conditions represented by the output flows must always have a valid result.
- ✓ Use mainly pairs of gates of the same type - This will increase the readability of the model and facilitate its validation.
- ✓ All actions performed by a participant are included in the same container - Subsequently, the container can be partitioned into swim lanes.
- ✓ There can be multiple start or end events in a process diagram - To remove confusion, they must be named differently.

Exercise 1-A

A) *Model the process of fulfilling the orders of a company producing finished products. The description of the process is as follows:*

Upon receipt of a purchase order, it is honored only if the product is in stock, otherwise the process is completed by rejecting the order. If the product is in stock, then it is taken from the warehouse and the order is confirmed. Subsequently, two groups of actions are performed independently: on the one hand, the delivery address is taken over and the product is delivered, and on the other hand, the invoice is issued and the payment confirmation is received. At the end, the order is archived and the process ends.

Exercise 1-B

B) *Extend the previously modeled process diagram with the possibility to manufacture a product, if it is not in stock. The following description will be considered:*

If the requested product is not in stock, then it must be manufactured before order management can continue. The necessary raw materials must be ordered to manufacture a product. The company works with two preferred suppliers who supply different types of raw materials. The availability of the necessary raw materials in the suppliers' offers is checked. Depending on the product to be manufactured, the raw materials can be ordered from either supplier 1, supplier 2 or both. Once the raw materials are available, the product can be manufactured and the order can be confirmed. On the other hand, if the product is in stock, it is taken from the warehouse before confirming the order. In both cases, the process continues normally.

Exercise 1-C

C) *Add to the previous diagram the data needed to execute the process. The following types of data have been identified:*

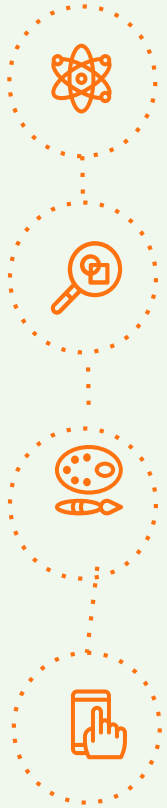
- Data objects: Order (with Confirmed and Paid status), Raw materials, Product (with Packed status), Delivery address, Invoice

- Stored data: Suppliers catalog, Warehouse DB, Warehouse Products, Orders DB

Information system design

Seminar 9 - BPMN Collaboration diagram

Collaboration diagram



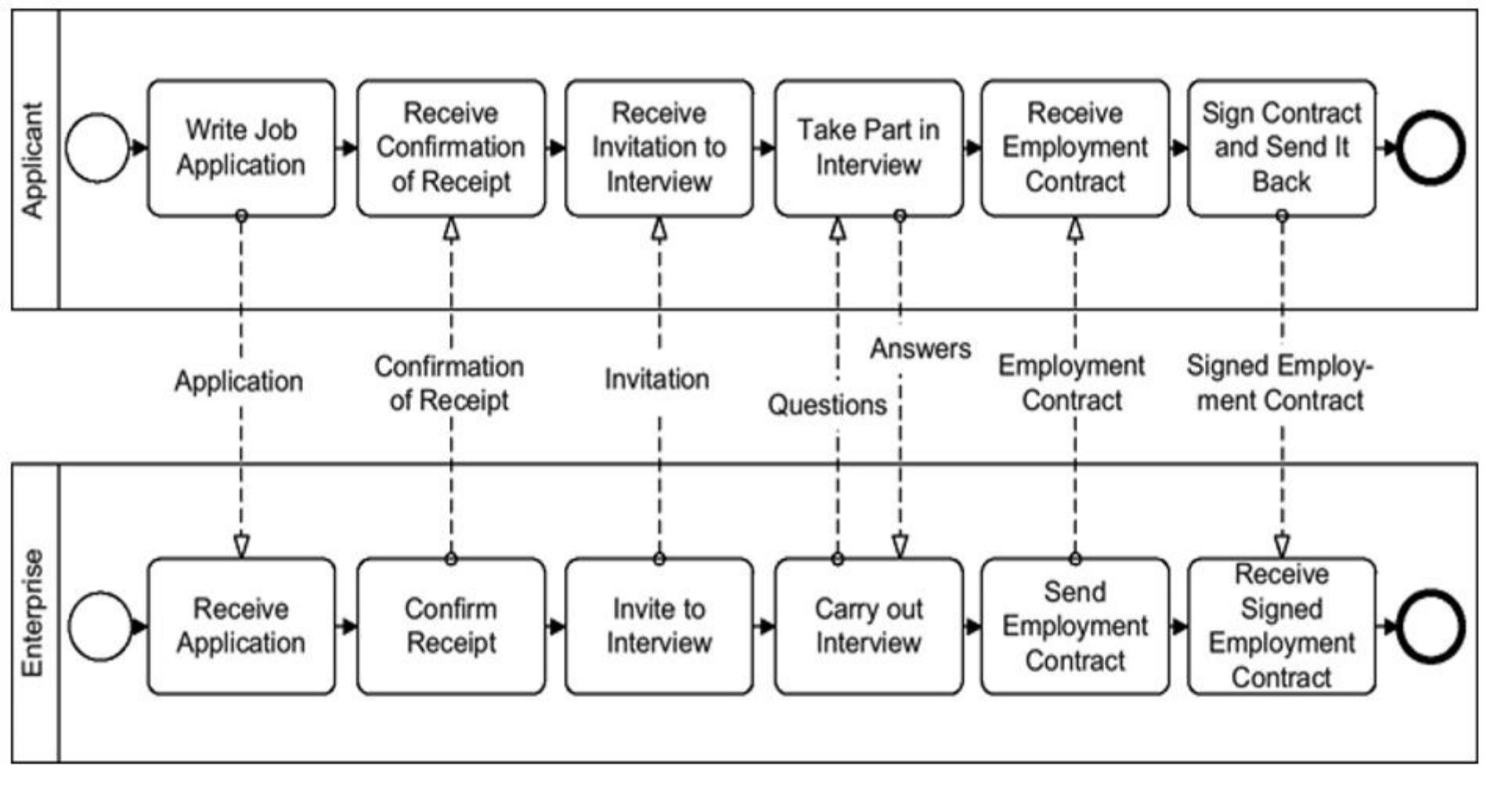
A collaboration is a synchronized interaction of two or more processes

Such an interaction is also called “choreography”

Collaboration diagrams are especially useful for documenting the co-operation of several companies.

Collaborations are modeled with two or more pools. Each pool contains a separate process.

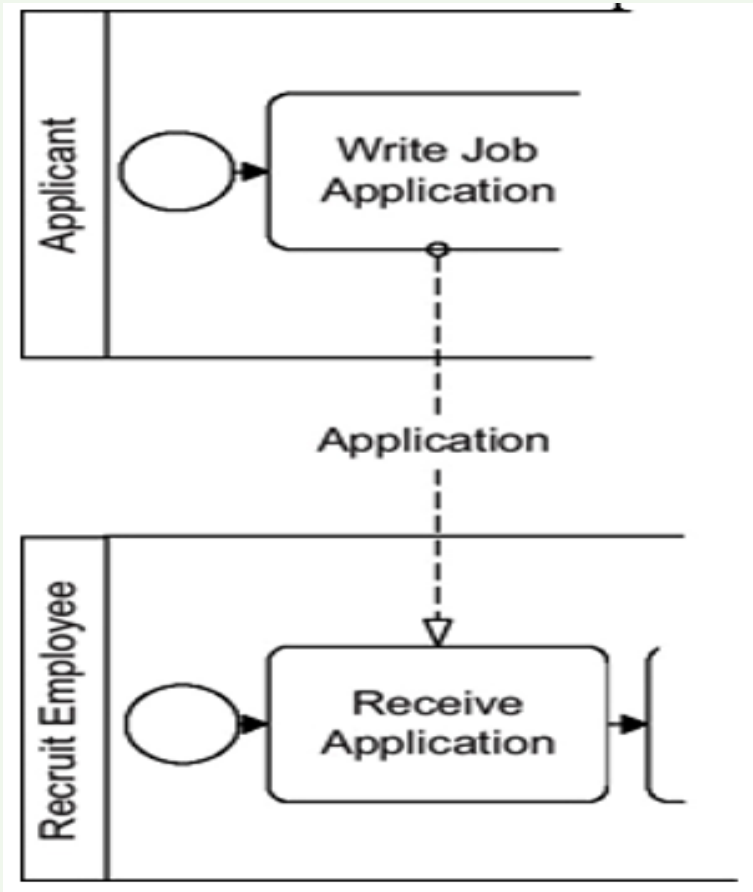
Collaboration diagram - example



Pool labels in a collaboration diagram

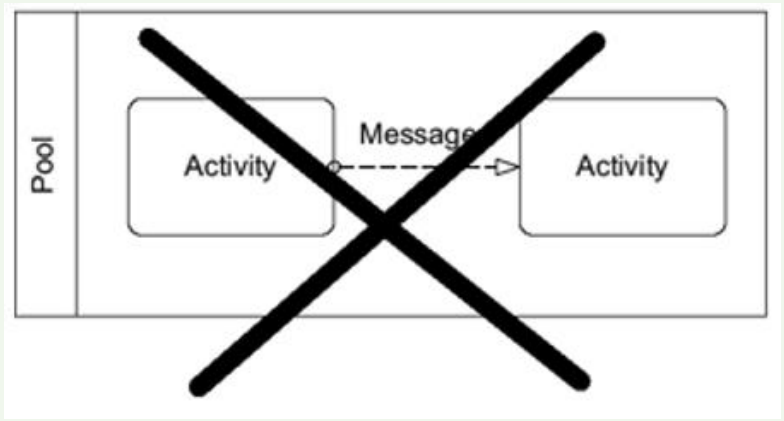
Partner role

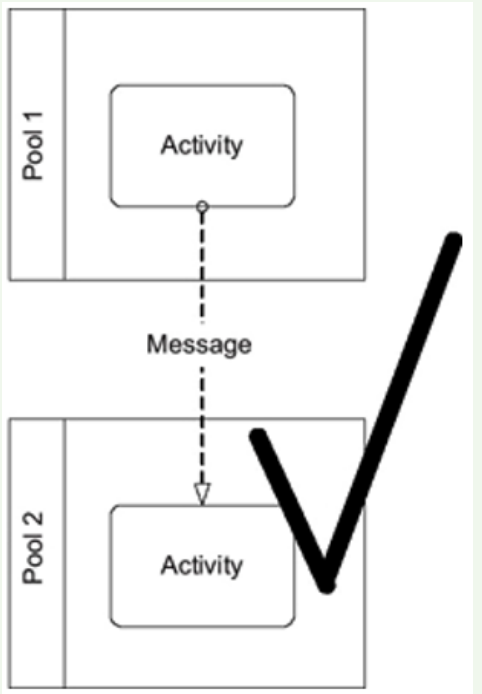
Process name



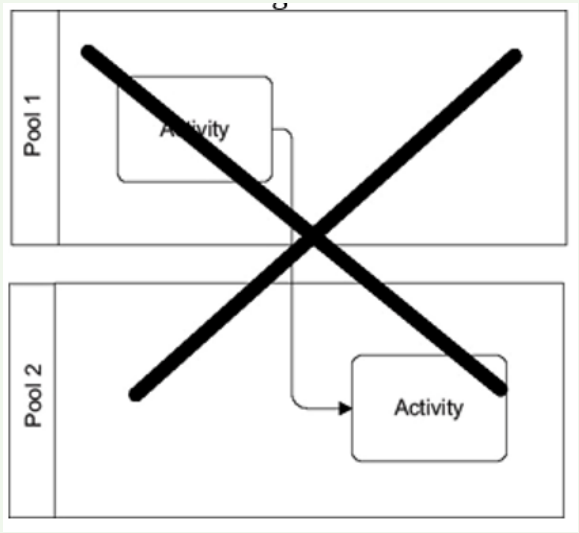
Modeling message/ sequence flows

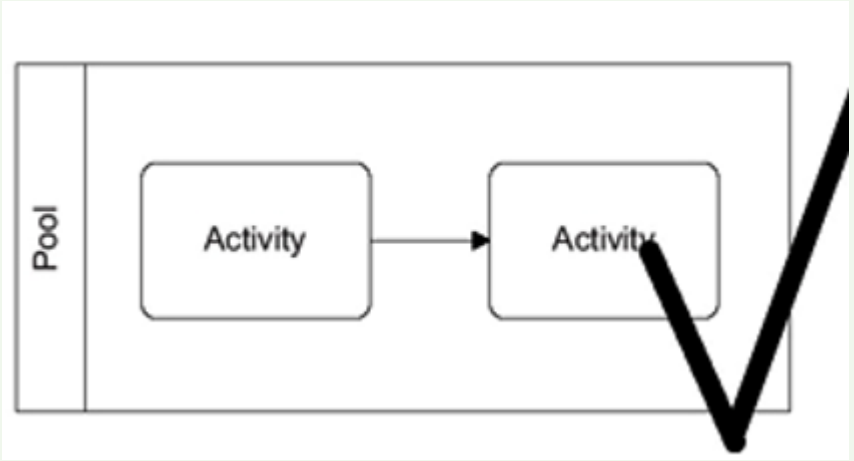
- ✓ **Message** flows are only allowed **between pools**, not within a pool



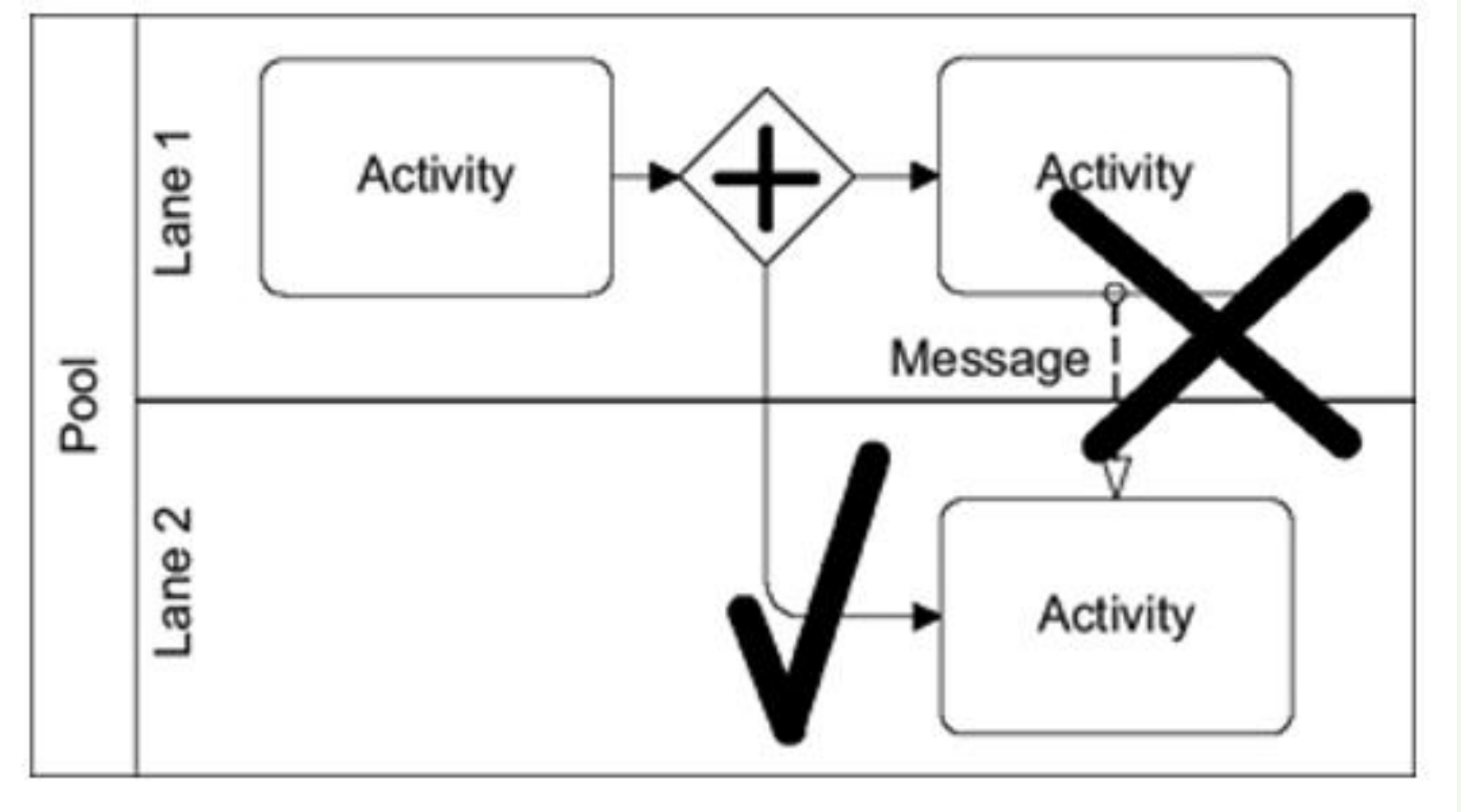


- ✓ **Sequence** flows, on the other hand, are used for modeling the flow of an independent process **within one pool.** Therefore, sequence flows must not cross borders of pools



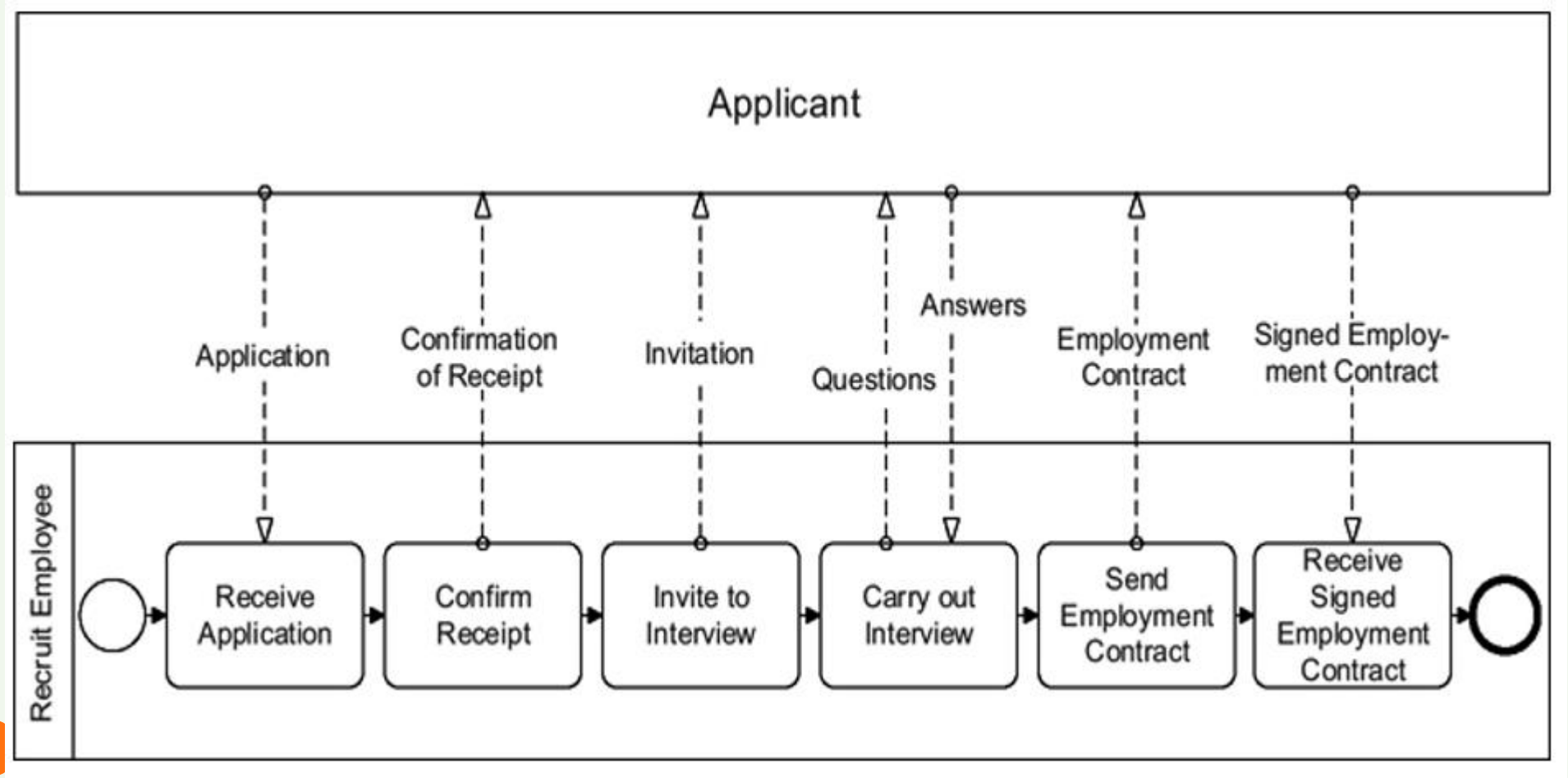


Modeling message/ sequence flows



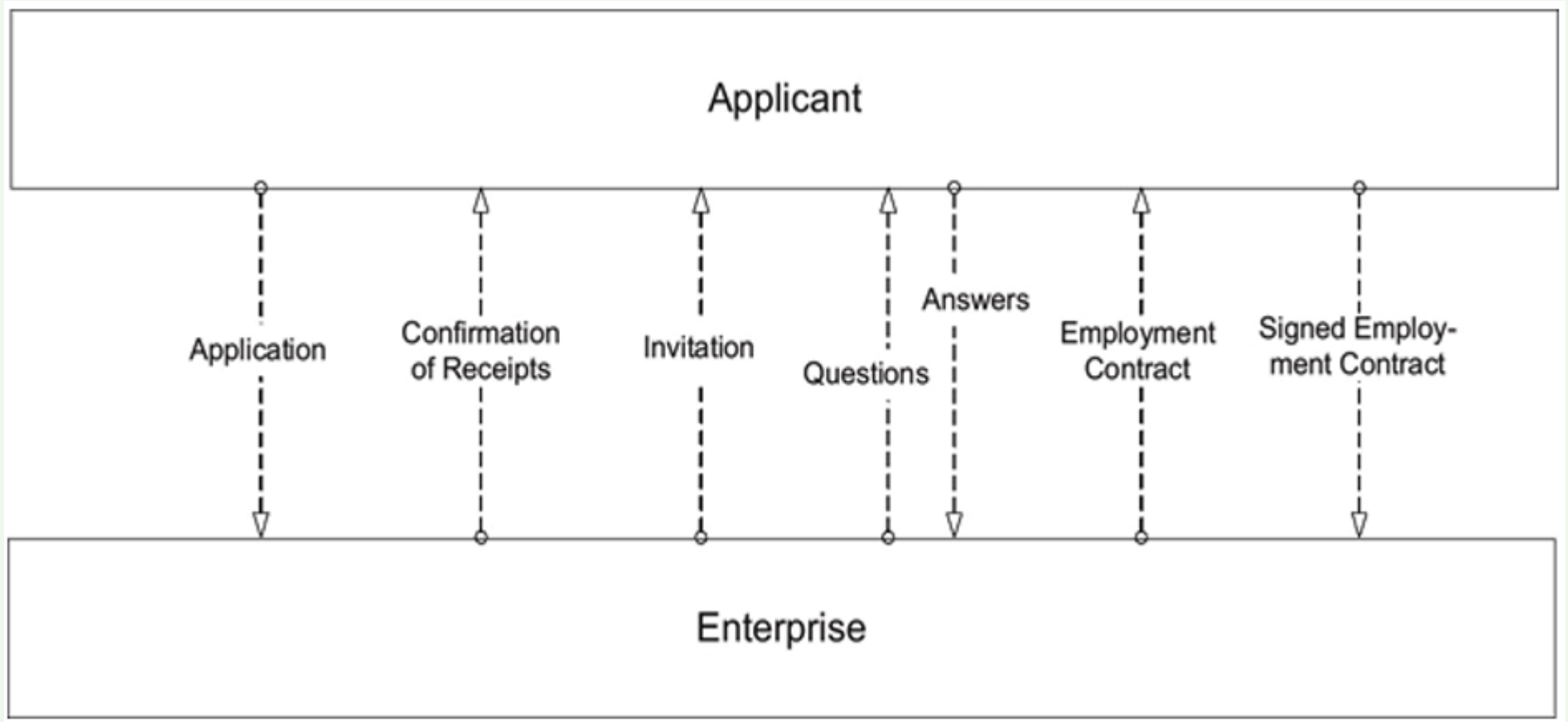
Message Flows to Black Box Pools

✓ In many cases only the internal processes of the own enterprise are known. In such a case, this partner's pool is drawn without the process.



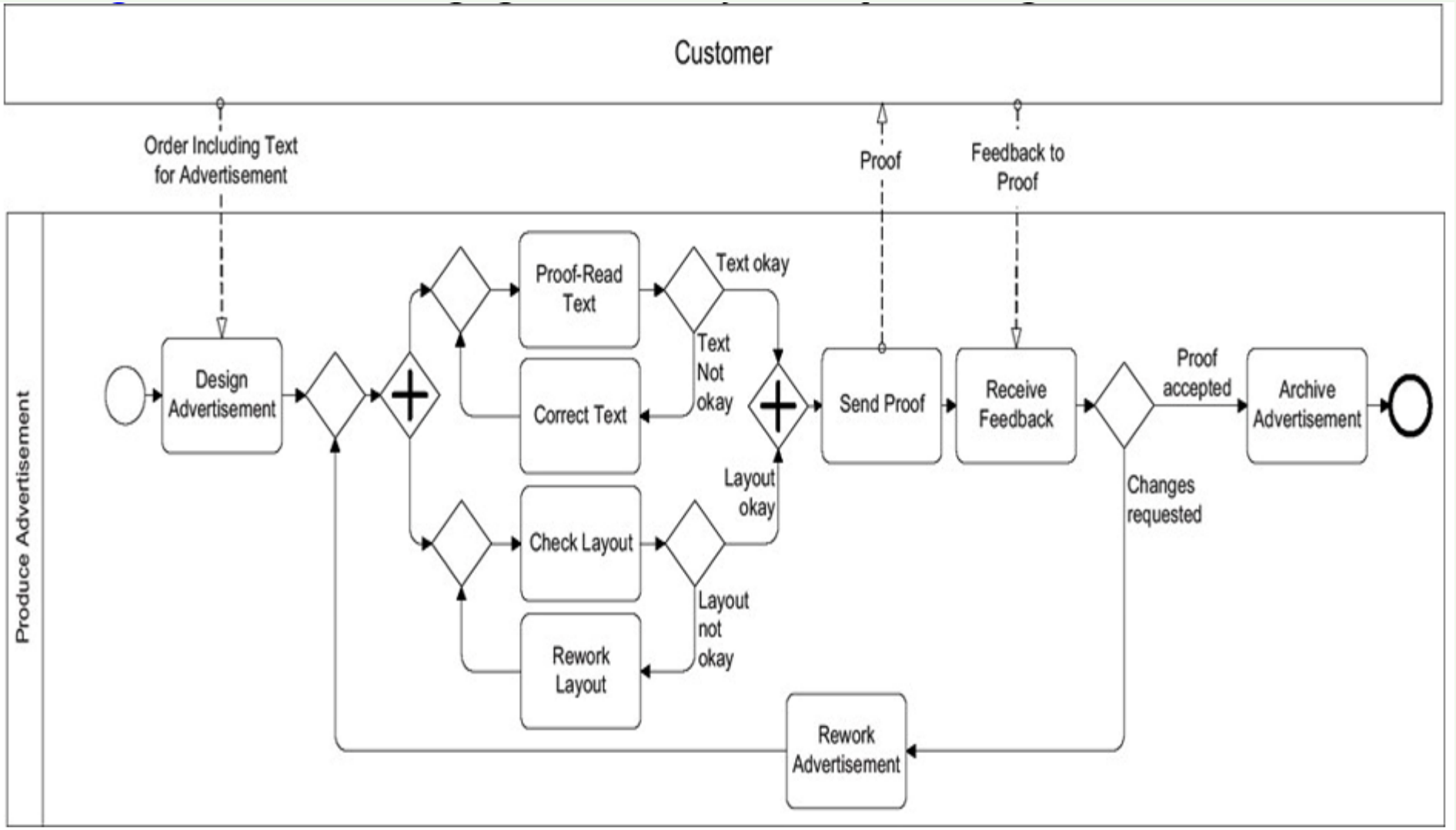
Message Flows to Black Box Pools

✓ If only the exchanged messages and their order are relevant, both pools can be drawn without their processes



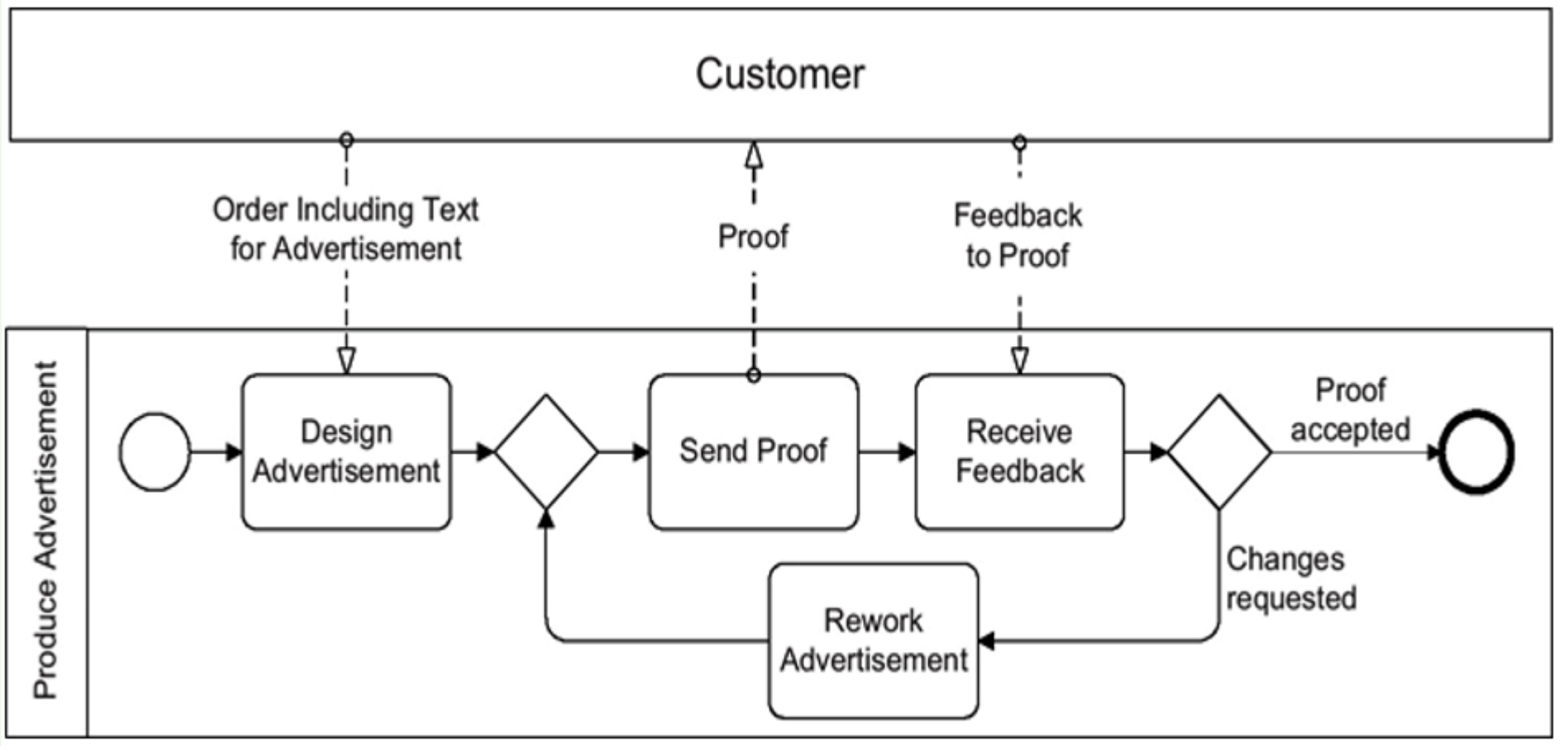
Private and Public Processes

- A **public process** provides a simplified view of the process to the partner.
- An internal process with all its details is called “**private process**”.

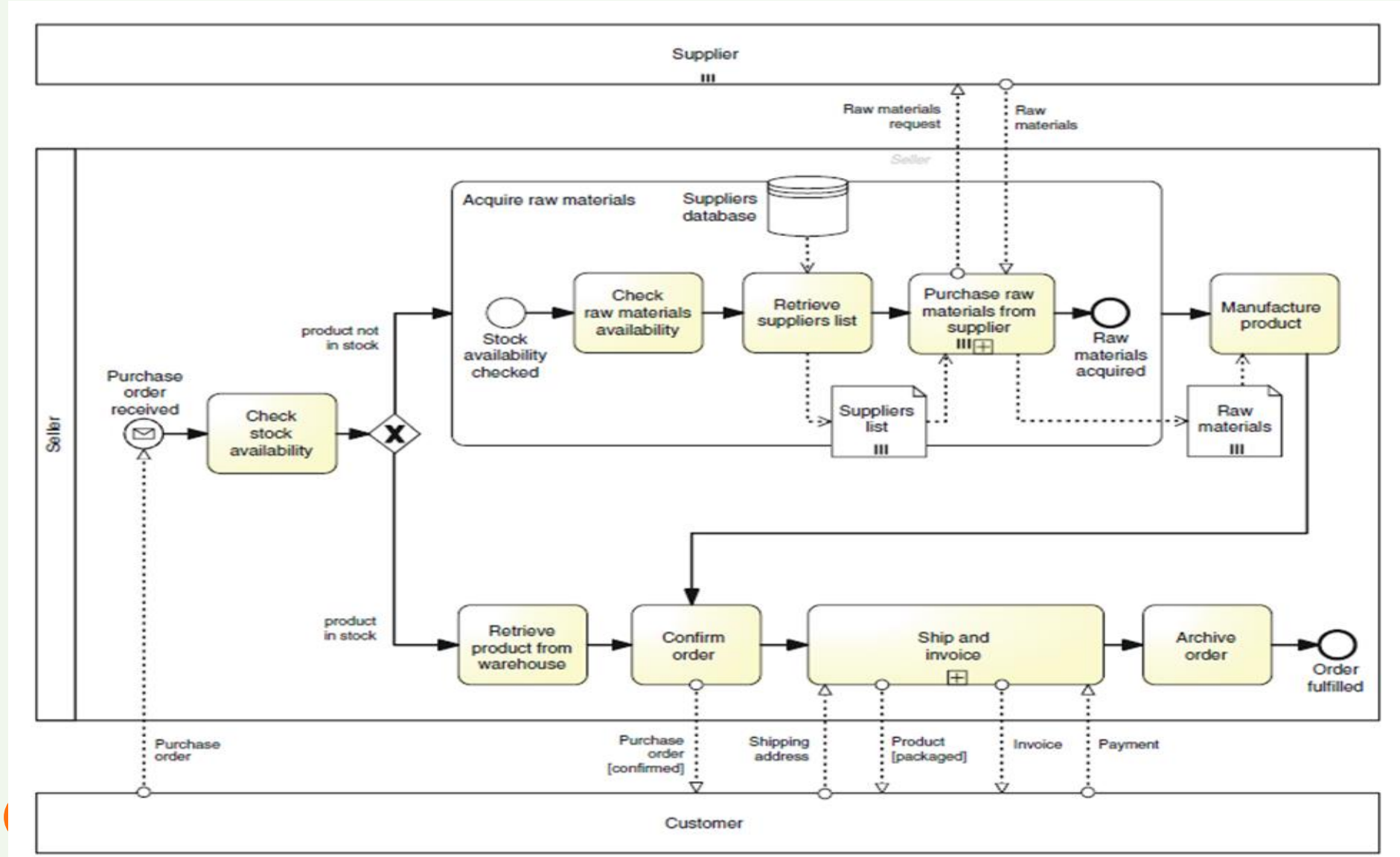


Private and Public Processes

- The public process…



Collaboration diagram - example



Information system design

Seminar 10 -

Design class diagram

Database design

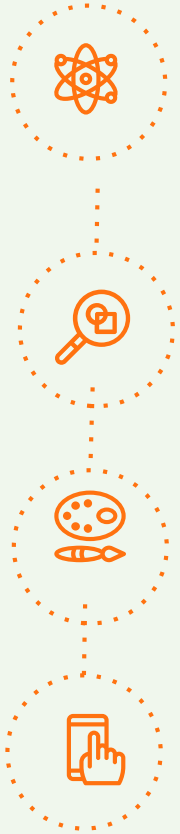
User interface design

1

Design class diagram

Design phase

Design class diagram



Go through the **use cases.** Select the **domain classes** that are involved in each use case

Add a **controller class** to be responsible for the use case

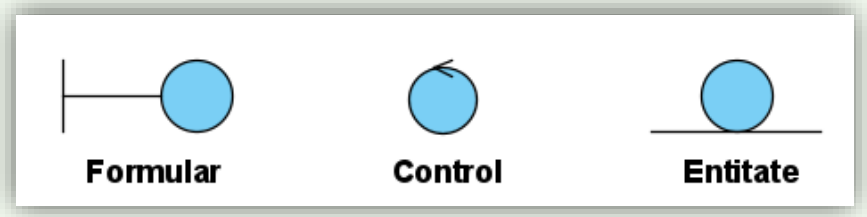
Determine the requirements for **navigation visibility**

Fill in the **attributes** of each class with *visibility* and *type*. Identify the responsabilities of every class and add specific methods to the class.

Design class stereotypes

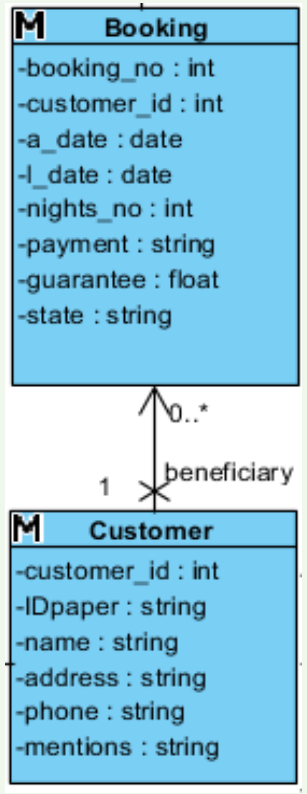
Stereotypes are added for classes

1. **Persistent class**– a class whose objects have to exist after the system is turned off
2. **Entity class**– an identifier for a problem domain class.
3. **Boundary / view class** – a class that is situated at the automated boundary of a system, such as in input form or a Web page
4. **Control class** – a class that mediates between boundary and entity classes, acting as a control panel between user level (visualization) and business level
5. **Data access class**– a class that is used to receive or send data from/to a database



Navigation visibility

- It is an object ability to view and interact with another object
- It is implemented by adding into a class an object reference variable;
- It is symbolized as an arrow on one of the association ends – Customer can view and interact with Booking

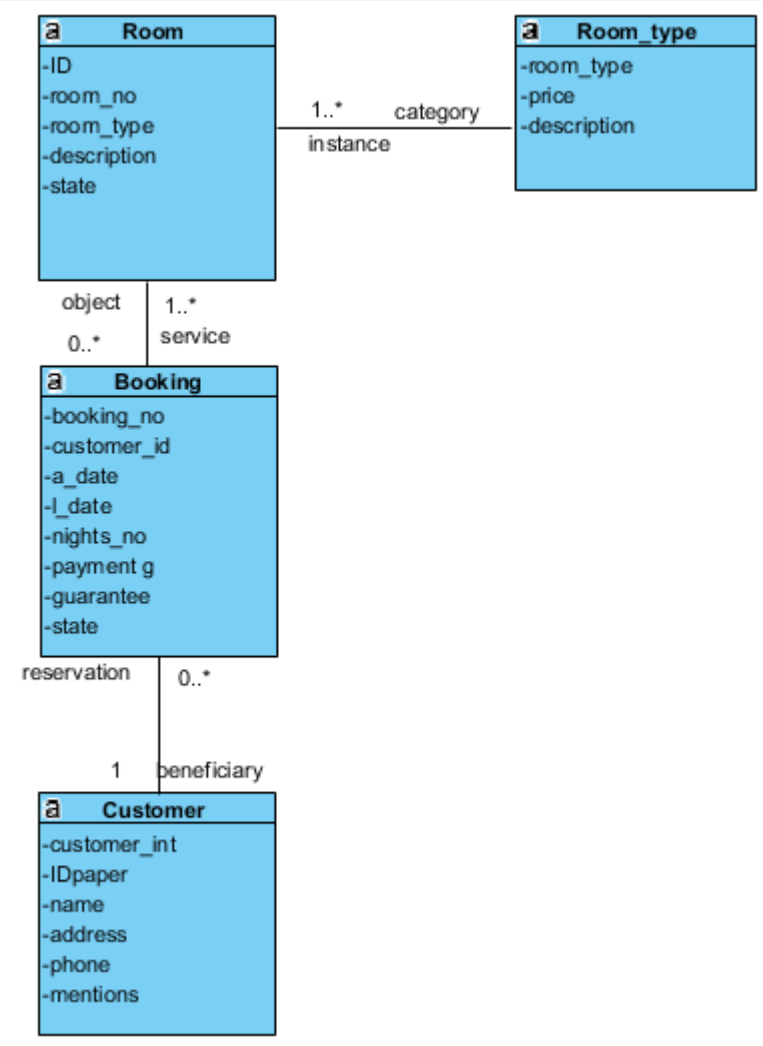


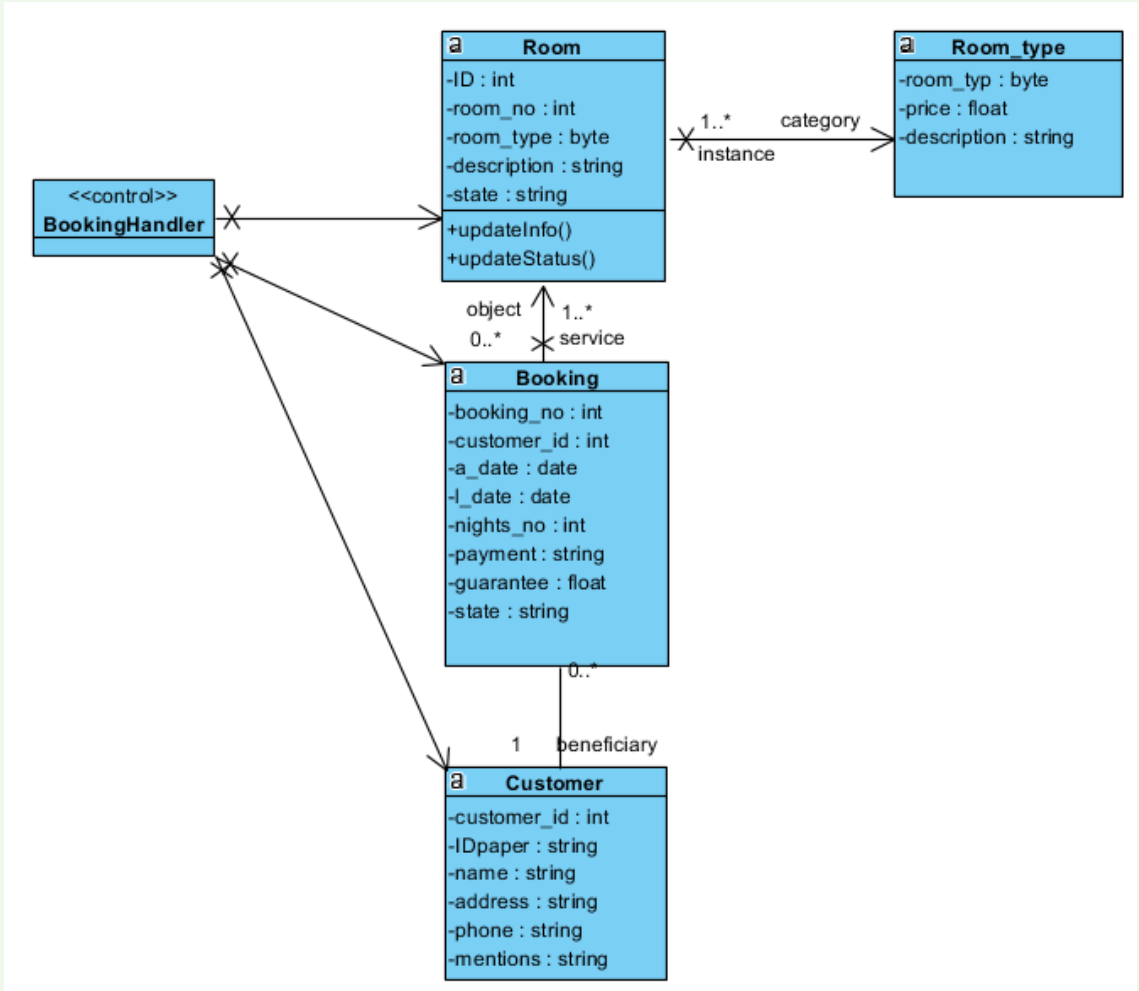

```
public class Client {
  private int _id_client;
  private String _act;
  private String _nume;
  private String _adresa;
  private String _telefon;
  private Object _mentiuni;
  public Vector<Rezervare> _unnamed_Rezervare_ = new Vector<Rezervare>();
...
```

Rules for navigability

- One-to-many associations that indicate a superior-subordinate relationship ensure navigation from superior to subordinates
- Mandatory associations, in which objects in a class can not exist without objects in another class, usually provide navigation from the most independent to the most dependent class
- When an object needs information from another object, you may need a navigation arrow

Example: Class diagram transformation

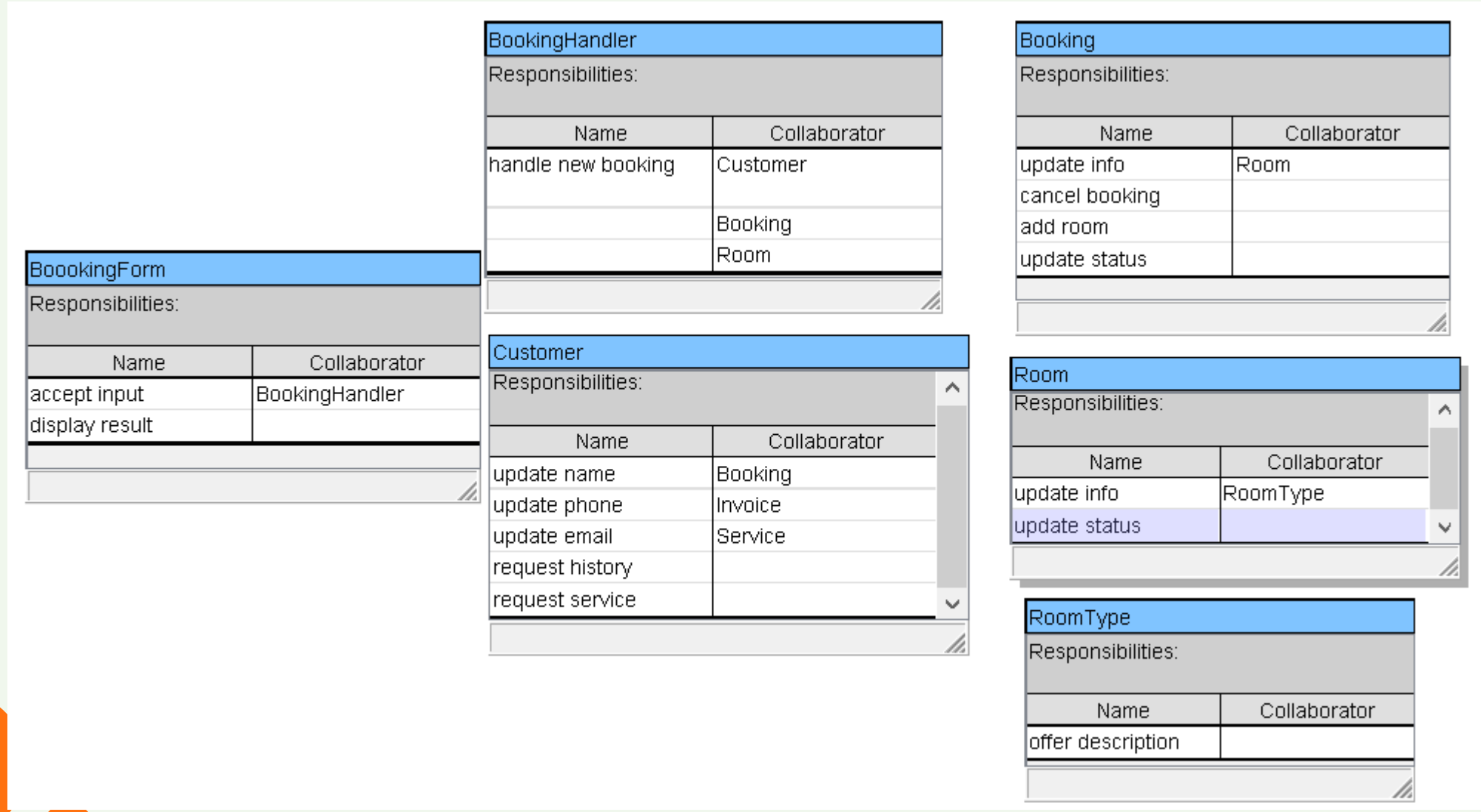




Class methods

- You can use the **CRC - Class, Responsibility, Collaboration** cards technique
  - What are the responsibilities of a class and how it collaborates with other classes to realize the use case
- It is obtained through brainstorming
- You can use **detailed sequence diagrams** - each message received by an object of a class must have a corresponding method in that class

CRC Example



BoookingForm

Responsibilities:

Name	Collaborator
accept input	BookingHandler
display result	

BookingHandler

Responsibilities:

Name	Collaborator
handle new booking	Customer
	Booking
	Room

Booking

Responsibilities:

Name	Collaborator
update info	Room
cancel booking	
add room	
update status	

Customer

Responsibilities:

Name	Collaborator
update name	Booking
update phone	Invoice
update email	Service
request history	
request service	

Room

Responsibilities:

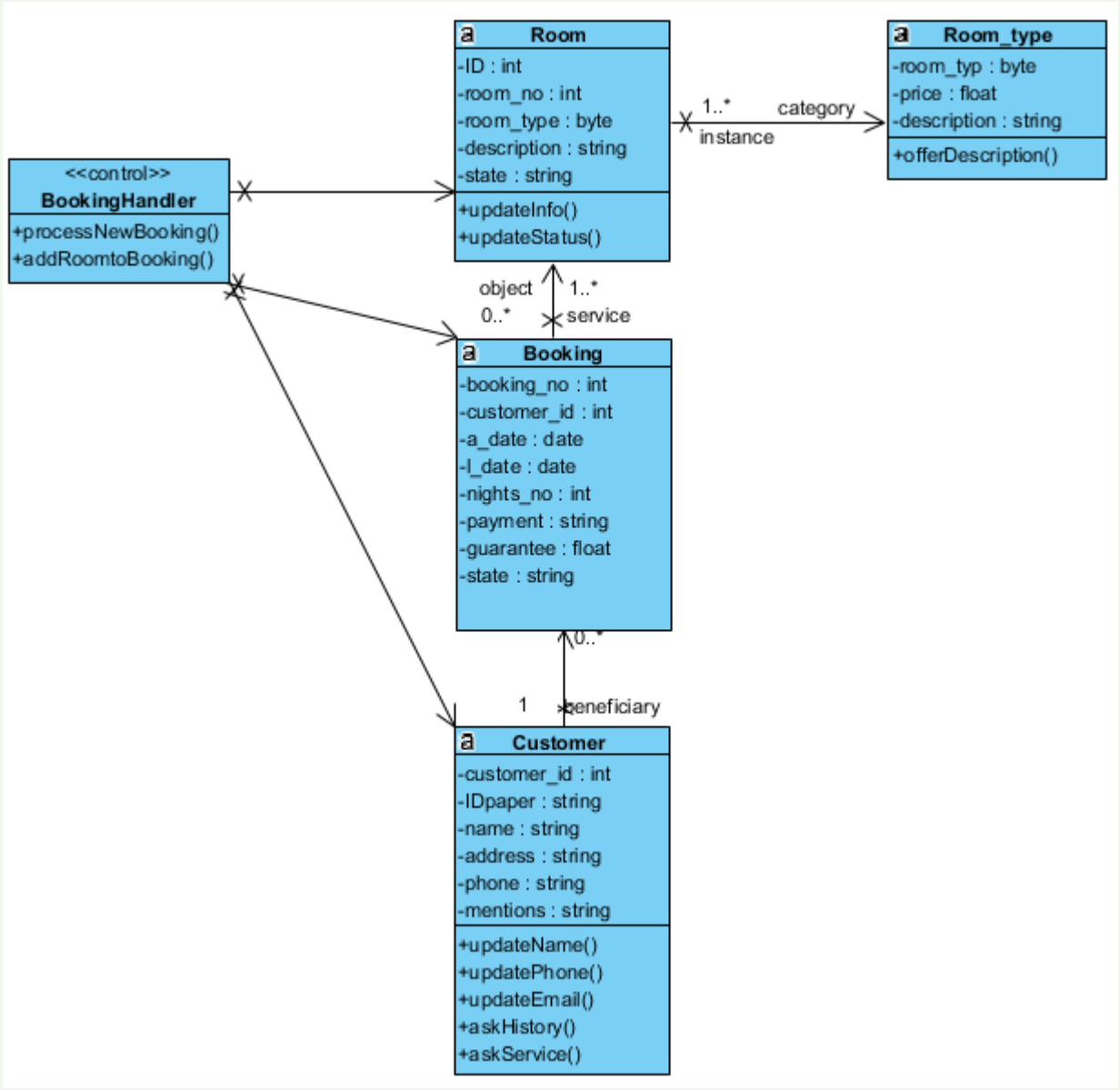
Name	Collaborator
update info	RoomType
update status	

RoomType

Responsibilities:

Name	Collaborator
offer description	

Adding methods



Protection against change

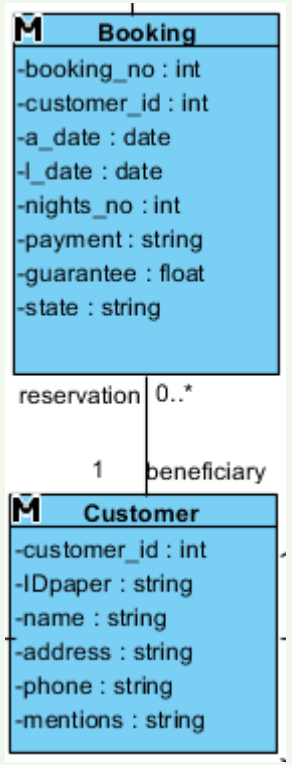
- A design principle is to separate the parts that are stable from the parts that undergo numerous changes.
- Separate **forms** and **pages in the user interface** that have a high probability of changing from the logic of the application.
- The **connection to the database** and **SQL logic** that are likely to change is kept in **separate classes** from the application logic
- Use **adapter classes** that can change for interaction with other systems
- If you choose between two design variants, choose one that offers greater protection against change

Association relationships in design

- It indicates a bidirectional connection between classes. Associations get implemented through attributes in class definitions.
- **For example:** Customer - Booking (one to many)

```
public class Customer {
  Collection<Service> used;
  private int customer_id;
  private string IDpaper;
  private string name;
  private string address;
  private string phone;
  private string mentions;
  Collection<Booking> reservation;
  Collection<Registry> arrival;
  Collection<Feedback> form;
  BookingHandler bookingHandler;
  }
```

```
public class Booking {
  Collection<Room> object;
  Customer beneficiary;
  private int booking_no;
  private int customer_id;
  private date a_date;
  private date l_date;
  private int nights_no;
  private string payment;
  private float guarantee;
  private string state;
}
```

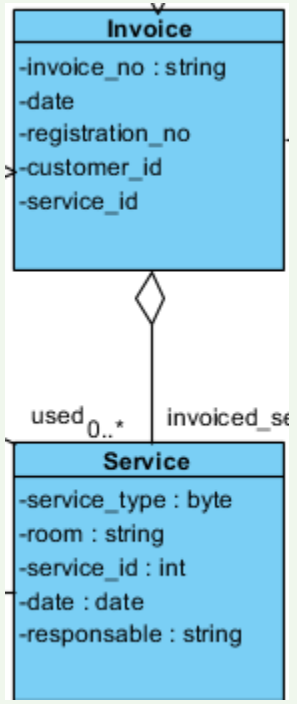


Implementing aggregations: by reference

- **Shared aggregation**: an object will contain only a reference to the second object; the contained object can be referred to by other objects as well
- For example, Invoice - Service

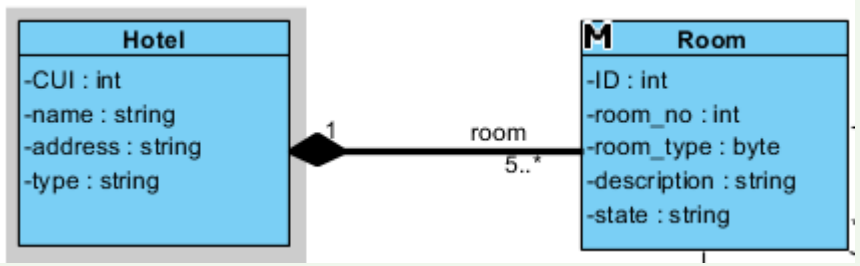
```
public class Invoice {
   Registry invoiced_registr;
   Customer beneficiary;
   Collection<Service> invoiced_service;
   Collection<Payment> achitare;
   private string invoice_no;
   private int date;
   private int registration_no;
   private int customer_id;
   private int service_id;
}
```

```
public class Service {
  Service_type category;
  Customer consumer;
  private byte service_type;
  private string room;
  private int service_id;
  private date date;
  private string responsable;
 }
```



Implementing aggregations: by value

- **Composed aggregation:** the main class is made up entirely of objects in the other class and cannot exist independently

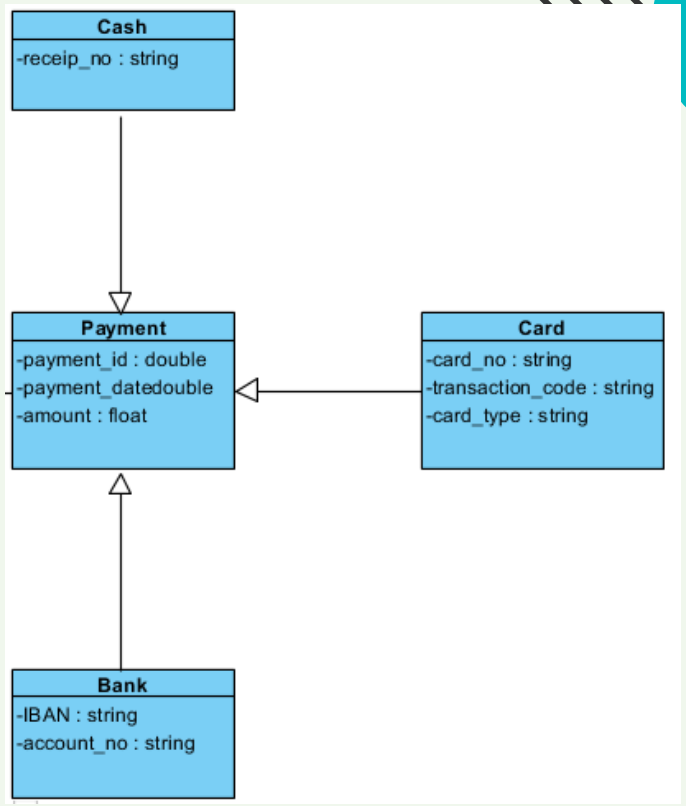



```
public class Hotel {
    Collection<Room> room;
    private int CUI;
    private string name;
    private string address;
    private string type;
 }
```

```
public class Room {
  Registry registry;
  BookingHandler bookingHandler;
  Room_type category;
  Collection<Booking> service;
  private int ID;
  private int room_no;
  private byte room_type;
  private string description;
  private string state;
 }
```

Implementing generalization relationship

- A subclass inherits from a superclass three specific elements: attributes, operations, and relationships. A practical hierarchy of inheritance is usually deep on a maximum of three levels
- For example, for the three types of payments, the inheritance mechanism allows the reuse of items from the base class, Payment
- Polymorphism: polymorphic behavior implies, during running, that the same message has different behavioral effects.




```
public class Payment {
    Invoice debt;
    private double payment_id;
    private int payment_datedouble;
    private float amount;
 }
```

```
public class Card extends Payment {
  private string card_no;
  private string transaction_code;
  private string card_type;
}
```

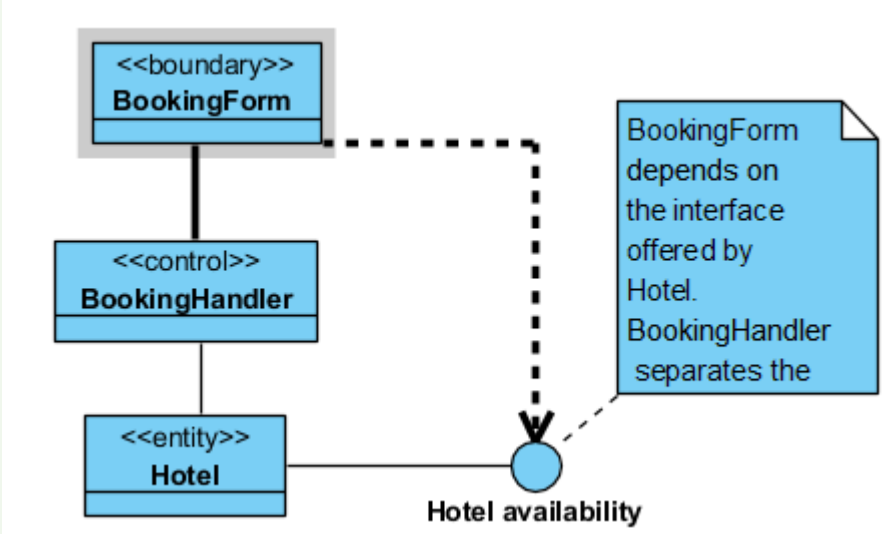
```
public class Bank extends Payment {
  private string IBAN;
  private string account_no;
}
```

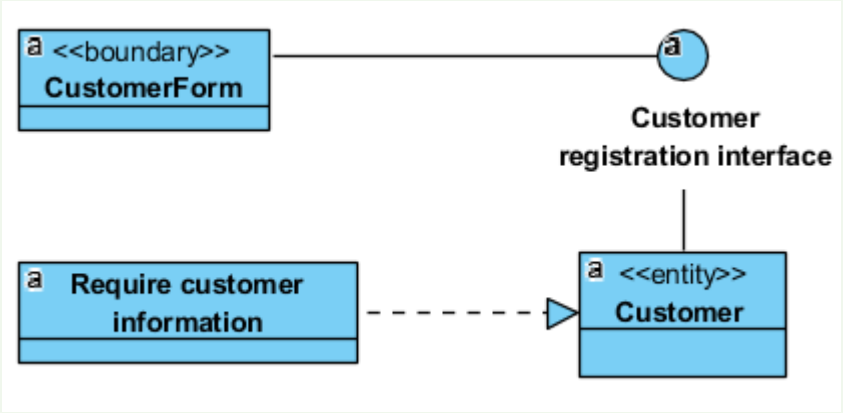
```
public class Cash extends Payment {
 private string receip_no;
}
```

Dependency and realization relationships

- **A dependency** is a relationship that shows that a modeled element requires other elements for its specification or implementation.

- **A realization** is a specialized abstraction relationship between two sets of model elements, one representing a specification (the supplier) and the other representing an implementation of the latter (the customer).



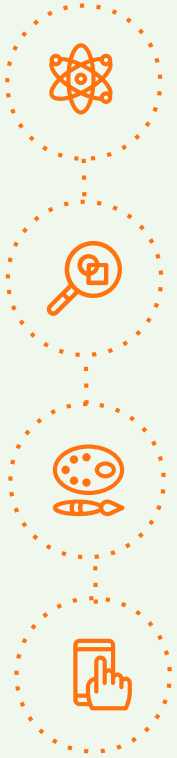


2

Database design

- ERD schema
- DDL generated script

Database design



It starts from the class model

The structure of the database is selected: relational, object-oriented, praph, message based, key-value pairs etc

The DB architecture is designed (distributed, centralized, etc)

The schema of the database is designed (tables and columns in relational model). The design the referential integrity constraints is added.

Mapping problem domain objects for RDBMS

Rule 1: Map all concrete-problem domain classes to the RDBMS tables. Also, if an abstract Problem Domain class has multiple direct subclasses, map the abstract class to a RDBMS table.

Rule 2: Map single-valued attributes to columns of the tables.

Rule 3: Map methods to stored procedures or to program modules.

Rule 4: Map single-valued aggregation and association relationships to a column that can store the key of the related table, i.e., add a foreign key to the table. Do this for both sides of the relationship.

Rule 5: Map multivalued attributes and repeating groups to new tables and create a one-to-many association from the original table to the new ones.

Rule 6: Map multivalued aggregation and association relationships to a new associative table that relates the two original tables together. Copy the primary key from both original tables to the new associative table, i.e., add foreign keys to the table.

Rule 7: For aggregation and association relationships of mixed type, copy the primary key from the single-valued side (1..1 or 0..1) of the relationship to a new column in the table on the multivalued side (1..* or 0..*) of the relationship that can store the key of the related table, i.e., add a foreign key to the table on the multivalued side of the relationship.

For generalization/inheritance relationships:

Rule 8a: Ensure that the primary key of the subclass instance is the same as the primary key of the superclass. The multiplicity of this new association from the subclass to the "superclass" should be 1..1. If the superclasses are concrete, that is, they can be instantiated themselves, then the multiplicity from the superclass to the subclass is 0..1, otherwise, it is 1..1. Furthermore, an exclusive-or (XOR) constraint must be added between the associations. Do this for each superclass.

OR

Rule 8b: Flatten the inheritance hierarchy by copying the superclass attributes down to all of the subclasses and remove the superclass from the design.*

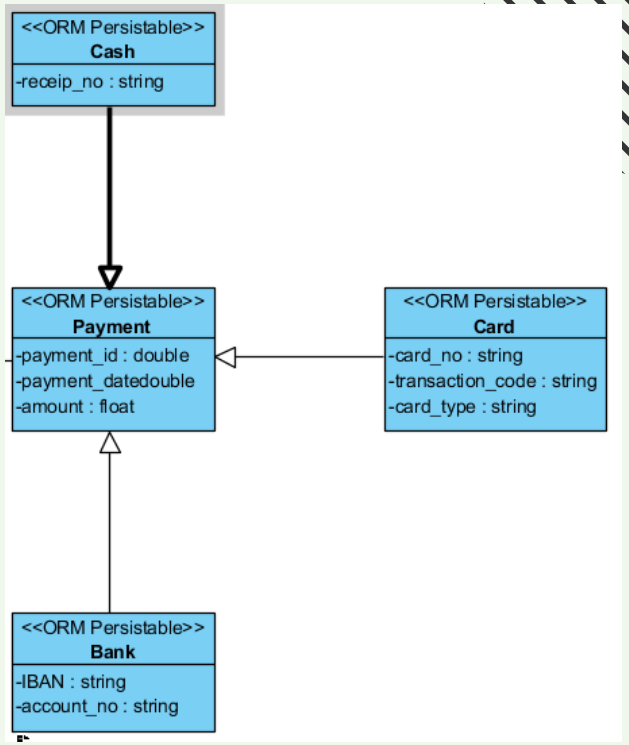
* It is also a good idea to document this modification in the design so that in the future, modifications to the design can be maintained easily.

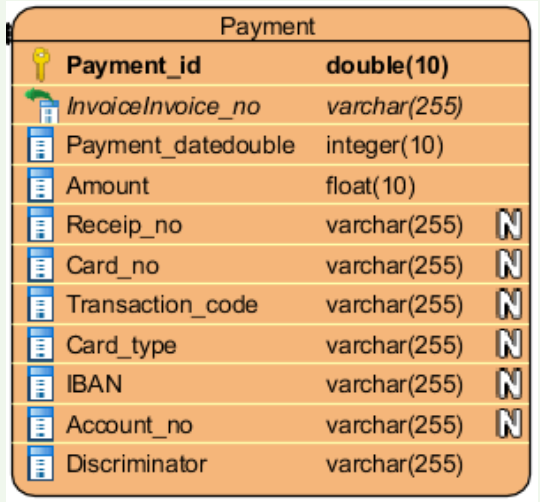
Inheritance relationships

a) The easiest way is to map all the attributes from the parent class, as well as subclasses, to the columns of a single huge table. Wasted space: when a Card object is stored in this particular table, it would leave the columns specific to Cash

b) Create tables for all the child classes and append the parent class attributes to it. It becomes more challenging for multiple levels of inheritance

c) Create separate tables for parent as well as child classes. These tables are then linked using the primary key of the table representing the parent class (PersonID)





Steps for DB generation in Visual Paradigm

- For persistent classes, add *<<ORM Persistable>>* stereotype
- Entity relationship diagram will be automatically generated (Synchronize to entity-relationship diagram)
- Refinements will be applied on the initial ERD
- The DB can be generated in the selected DBMS (or DDL file for DB objects)

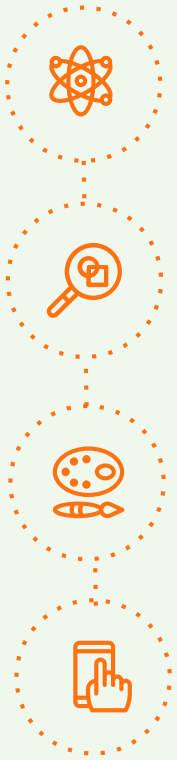
- See the tutorial for ERD and DB generation on online.ase.ro

2

Interface design

- UI interface

Interface design



The use case diagrams are completed and the first stable version of interaction and classes diagrams are developed

It is recommended to implement a prototype of the IT system,called interface prototype

It helps in refining the relationships between actors and interface classes

The main goal is to obtain feedback from the client (client) on the visual aspect of the application

Interface design -questionnaires

- The first step - investigating the actors' expectation on the interface by completing specific questionnaires consisting of the following questions:
  - What level of training (computer science) the actor requires to achieve a certain functionality?
  - Does the actor have working experience in window-based environments?
  - Does the actor have experience in using other automated process modeling systems?
  - Is it necessary to consult documents / catalogs in parallel with the use of the application?
  - Does the actor want to implement 'rescue / restoration' facilities?

Interface design goals

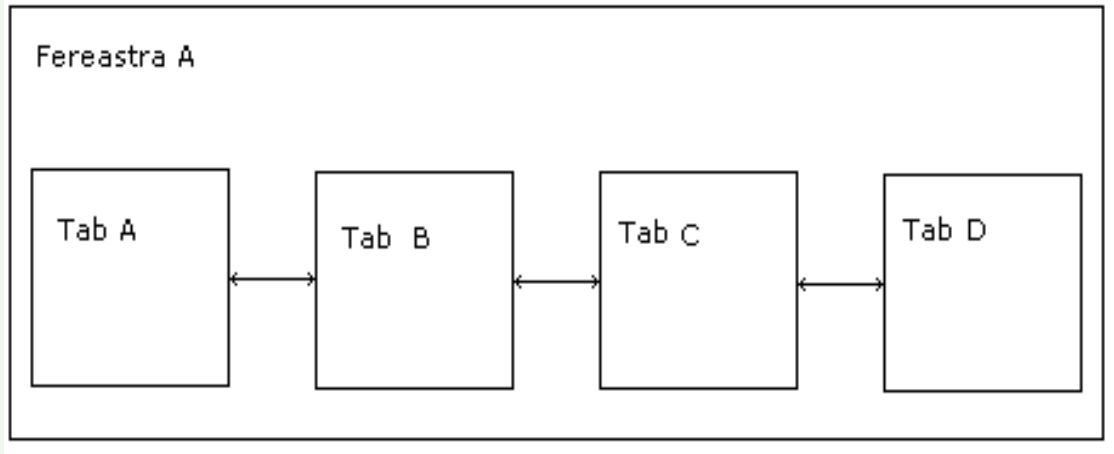
- The goals of the prototype are:
  - Setting interface requirements for key application functionalities;
  - It demonstrates to the client (in a visual form) that the project requirements have been well understood and are achievable;
  - The start of the development phase of the standard interface elements.

Screen structure maps

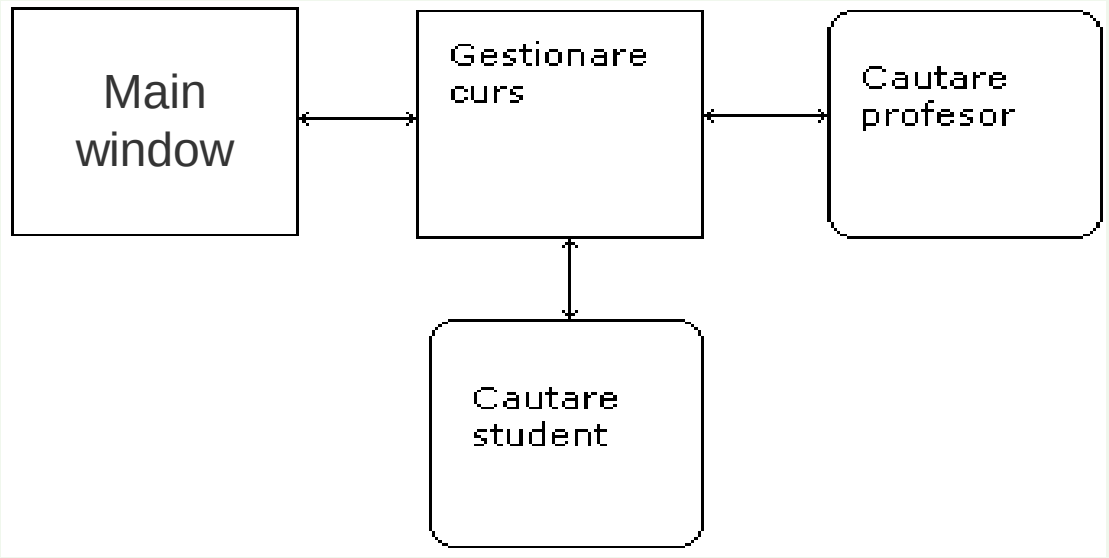
- Screen structure maps (charts) are used to describe the flow of the application following the main ways of use.
- Representation :
- Square shapes for modal window representation (requires a user response to continue an activity).
- Square shapes with rounded corners for the representation of non-modal windows
- The crossing direction shows the window navigation path.

Interface design

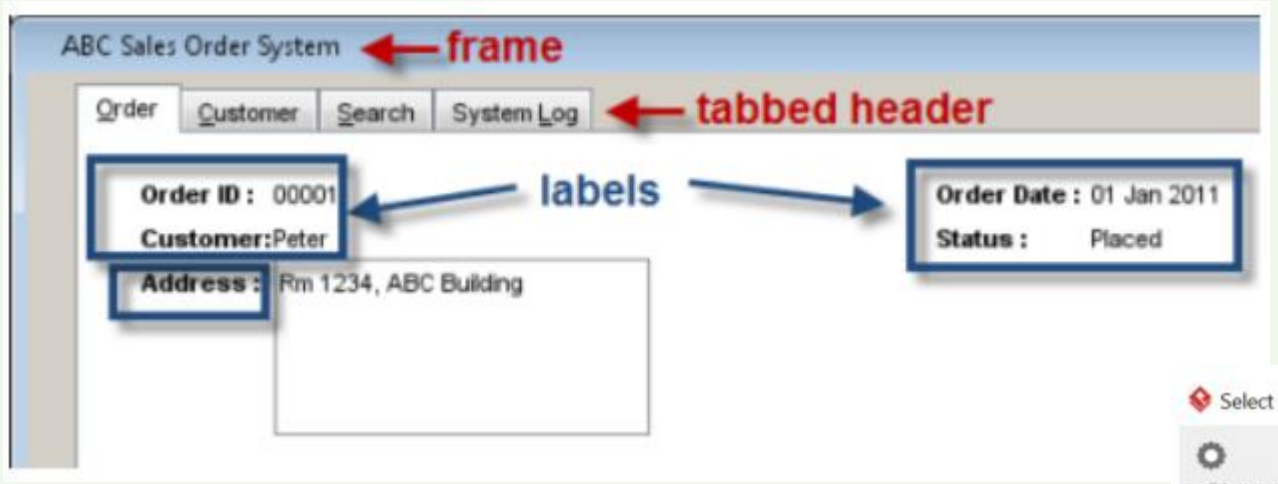
Windows that contain Tabs

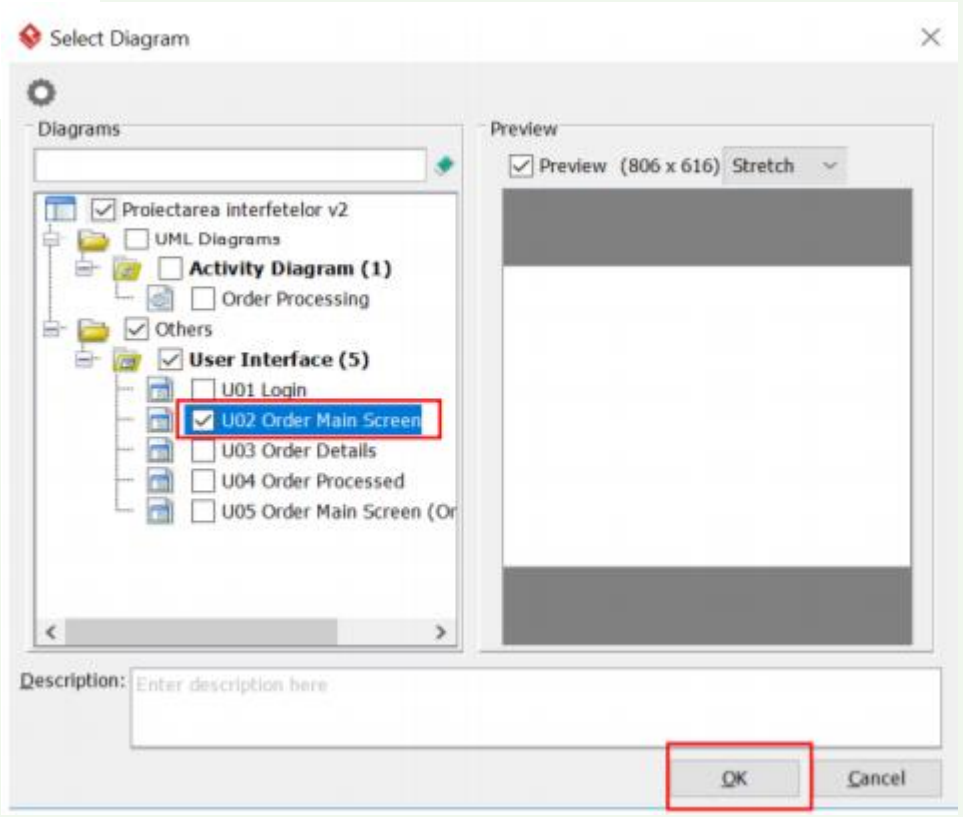


Screen structure diagram



Interface design – example (see VP tutorial)





Interface design – Balsamiq

- *Example for mobile banking app:*

https://balsamiq.com/tutorials/articles/mobileapplication/

Information system design

Seminar 11 - UML

- Component diagram
- Deployment diagram

Component diagram

A component diagram shows the **dependencies** between different **software components** that make up a software system.

.

These dependencies can be: a) **Static** - occur in the compilation or link-editing steps; b) **Dynamic** - occur during execution

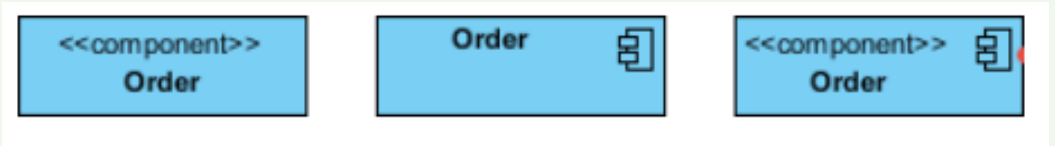
This diagram models overall system architecture and the logical components within it. It contains:

Logical, reusable, and transportable system components that define the system architecture.

Well-defined interfaces, or public methods, that can be accessed by other programs or external devices: the application program interface (API)

Component

- It is a software file, module or an executable program (source code, binary code, dll, executable etc) with a well defined interface.
- Each component is responsible for one clear aim within the entire system and only interacts with other essential elements on a need-to-know basis.
- Components are considered a**utonomous, encapsulated units** within a system or subsystem that provide one or more interface
- By classifying a group of classes as a component the entire system becomes more modular as components may be interchanged and reused.
- In UML 2, a component is drawn as a **rectangle** with **optional** compartments stacked vertically.
- A component can be represented as a rectangle with a name and <<component>> stereotype.



Component stereotypes

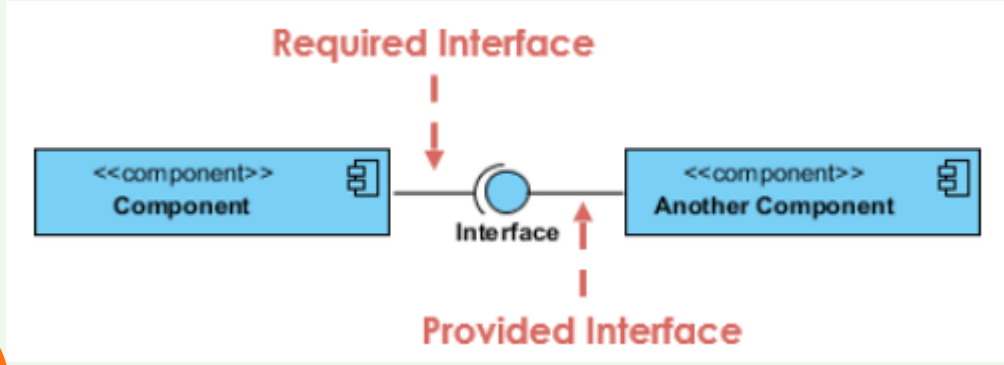
- Stereotypes have the role of specifying the type of software component. You can choose from predefined stereotypes or add new stereotypes.
- Examples of predefined stereotypes for components:
  - <<Main Program>>
  - <<SubProgram>>
  - <<Package>>
  - <<DLL>>
  - <<Task>>
  - <<EXE>>

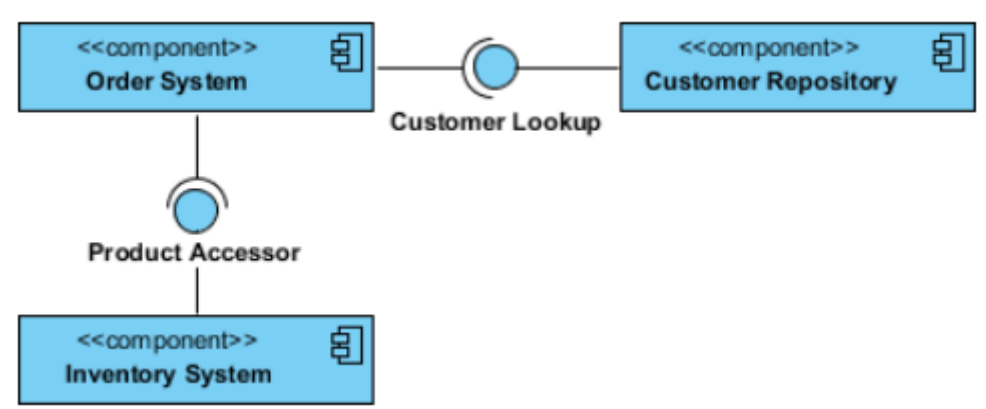
Interface

- The interface specifies a contract consisting of a set of attributes and public operations for a class.
- There are two types of component interfaces:

i. **Provided interfaces** symbols with a complete circle at their end represent an interface that the component provides - this "lollipop" symbol is shorthand for a realization relationship of an interface classifier.

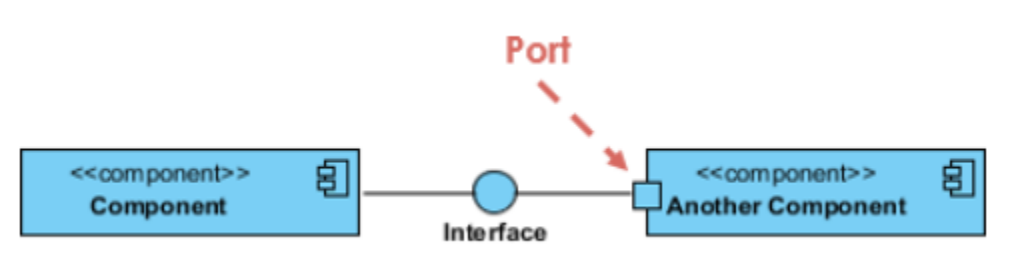
ii. **Required interfaces** symbols with only a half circle at their end (a.k.a. sockets) represent an interface that the component requires (in both cases, the interface's name is placed near the interface symbol itself).

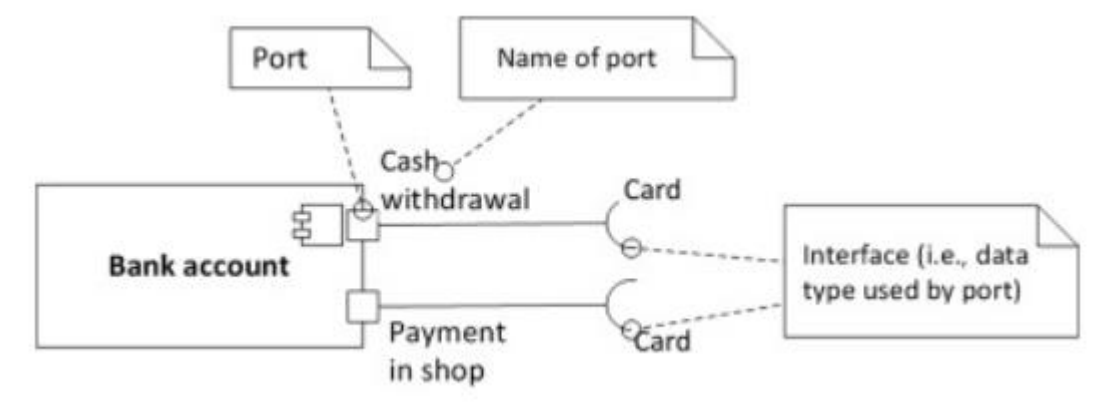




Port

- Ports are represented using a square along the edge of the system or a component.
- A port is often used to help **expose** required and provided interfaces of a component.
- It is an explicit window into an encapsulated component. All of the interactions into and out of such component pass through ports.
- Each port provides or requires one or more specific interfaces.





Relationships among components

Dependency relationship

They are graphically represented through dashed lines between a client component and a service provider component, targeting the provider component

The classes included in the client component can **inherit, instantiate or use** classes included in the provider component.

There may also be relationships of dependence between components and interfaces of other components

Composition relationship

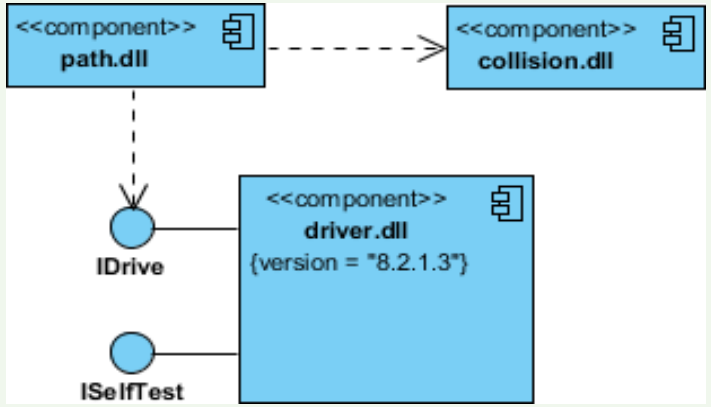
It describes components that are physically contained within other components

Source code modeling

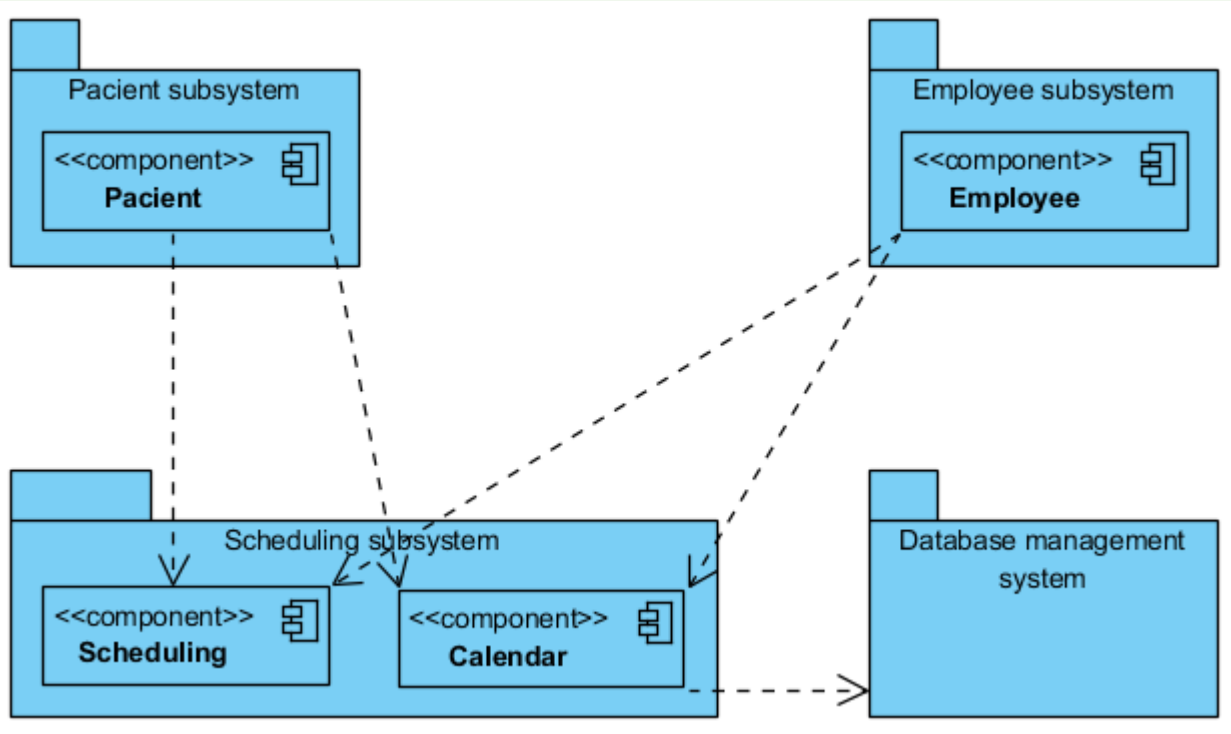
- Either by forward or reverse engineering, identify the set of source code files of interest and model them as components stereotyped as files.
- For larger systems, use packages to show groups of source code files.
- Consider exposing a tagged value indicating such information as the version number of the source code file, its author, and the date it was last changed.
- Model the compilation dependencies among these files using dependencies. Again, use tools to help generate and manage these dependencies.

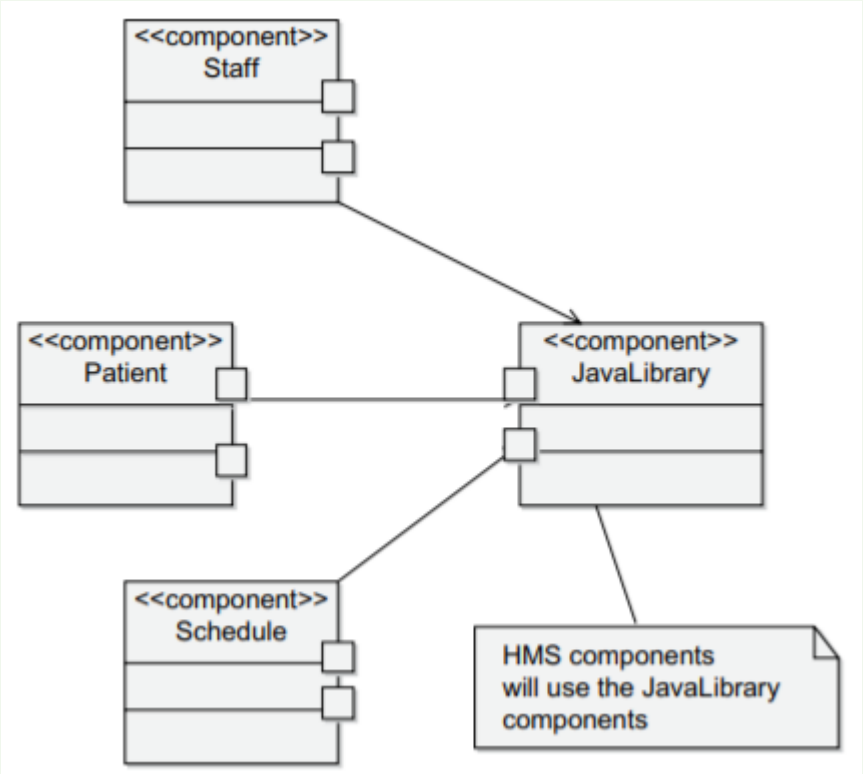
Modeling an executable release

- Identify the set of components you'd like to model. Typically, this will involve some or all the components that live on one node, or the distribution of these sets of components across all the nodes in the system.
- Consider the stereotype of each component in this set. For most systems, you'll find a small number of different kinds of components (such as executables, libraries, tables, files, and documents).
- For each component in this set, consider its relationship to its neighbors. Most often, this will involve interfaces that are exported (realized) by certain components and then imported (used) by others. If you want your model at a higher level of abstraction, elide these relationships by showing only dependencies among the components.

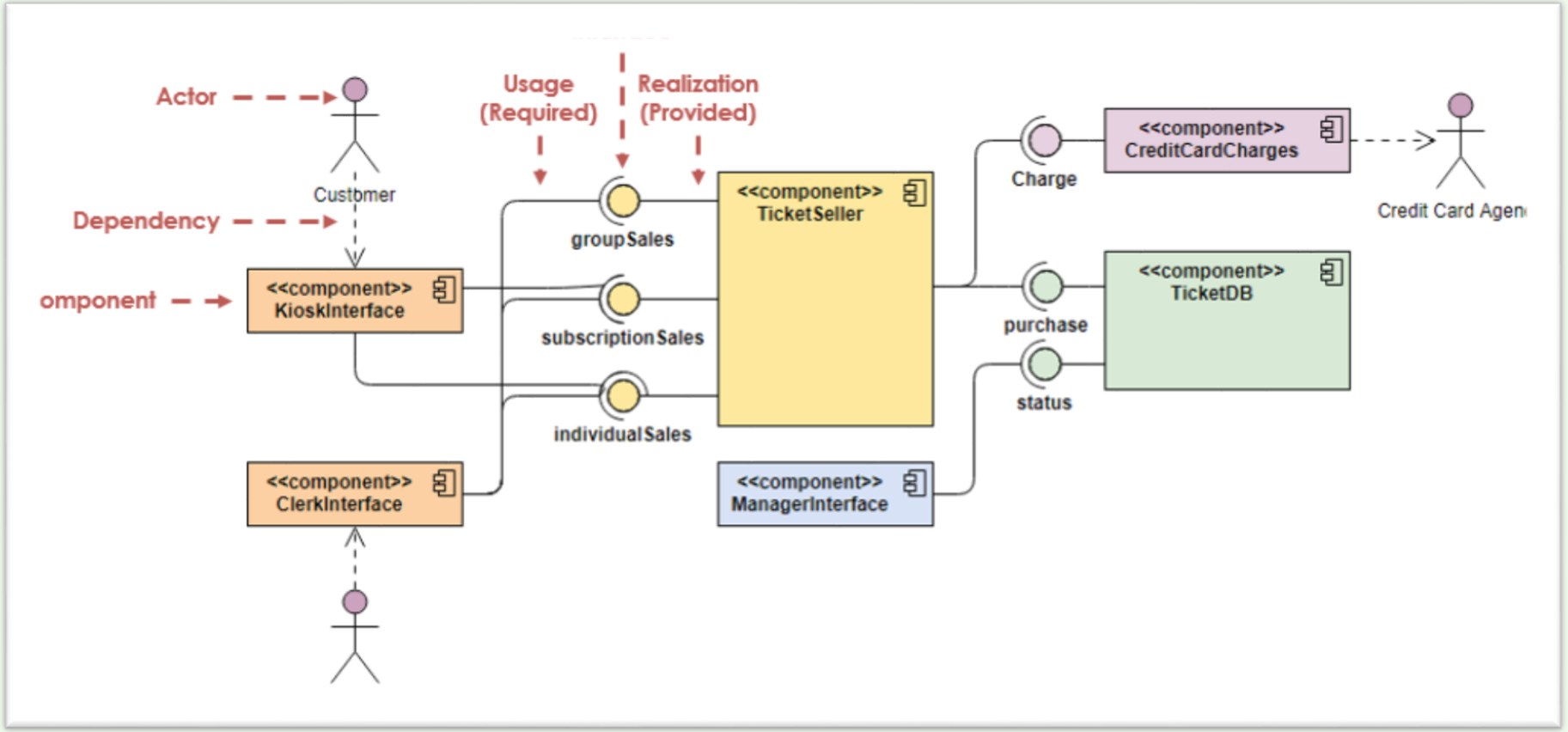


Dependencies modeling trough packages

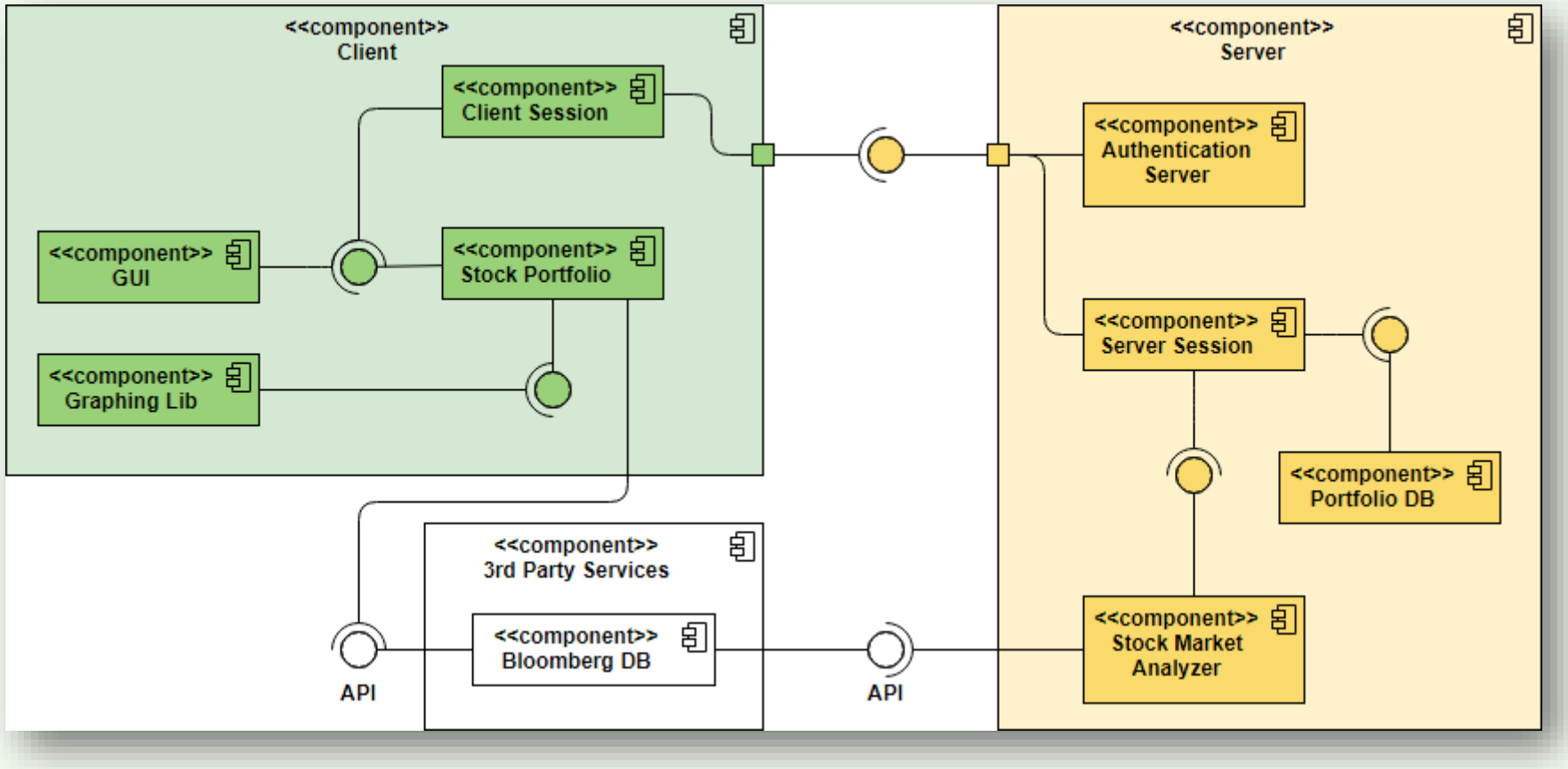




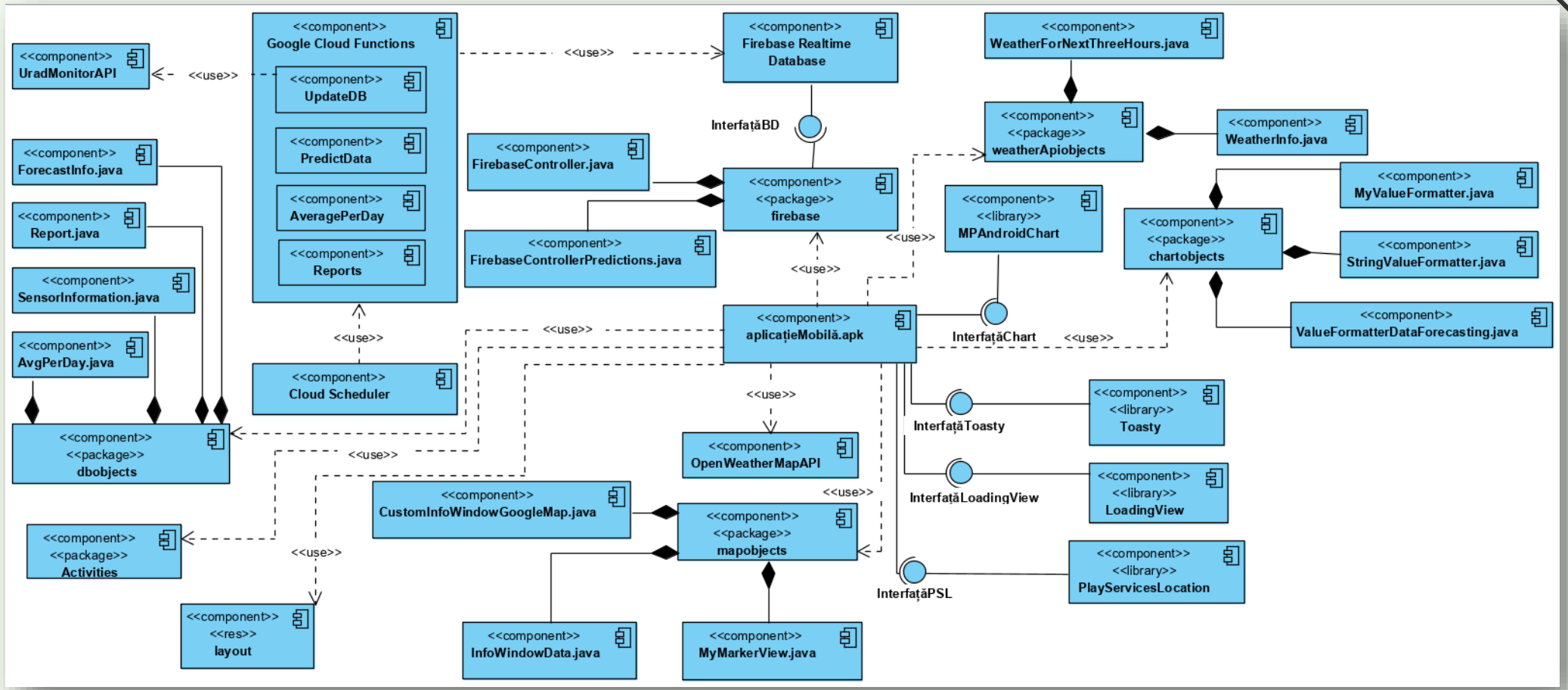
Component diagram – example 1



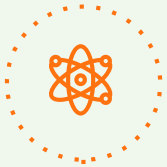
Component diagram - example 2



Component diagram - example -3



Deployment diagram



It shows the configuration of the processing elements during execution and the components, processes and objects they contain

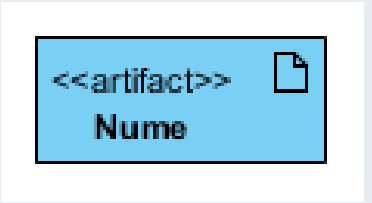
It is a graph of nodes connected through communication associations.

.

It can be used to represent components that may belong to certain nodes by embedding the component symbol within the symbol representing the node.

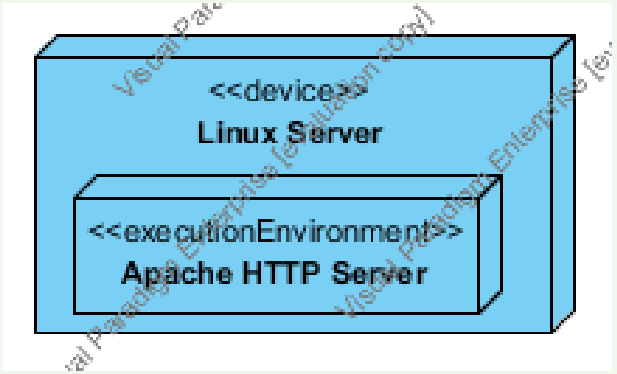
When components are included, a deployment diagram represents the environment for software components execution

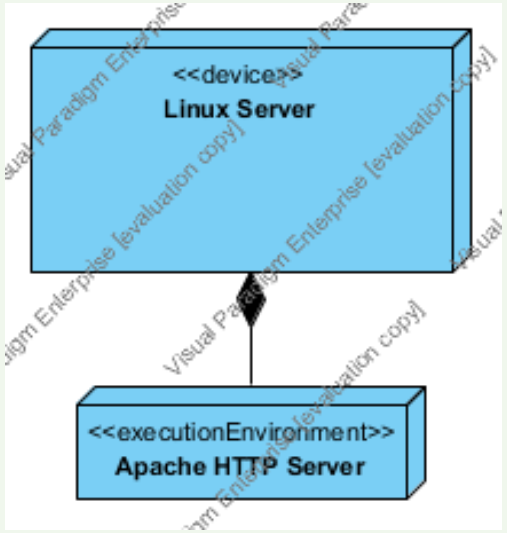
Deployment diagram – notations

Element	Reprezentare
Node: ▪ It is a physical entity that is a processing resource with memory and processing capabilities (computing devices, human resources, mechanical processing resources). ▪ It is labeled with its name (a noun) ▪ It may have a stereotype to indicate a specific node type, i.e., device, client workstation, application server, mobile device etc.	
Artefact: ▪ It is a specification of a software component. ▪ It has a label with its name. ▪ It may have a stereotype to indicate a specific artifact type (Source file, table in a database, executable file).	
Communication link: ▪ It is an association between two nodes that indicates the existence of a communication path between nodes. ▪ It may have a stereotype to indicate a specific communication type (Internet, serial, parallel).	 

Node types

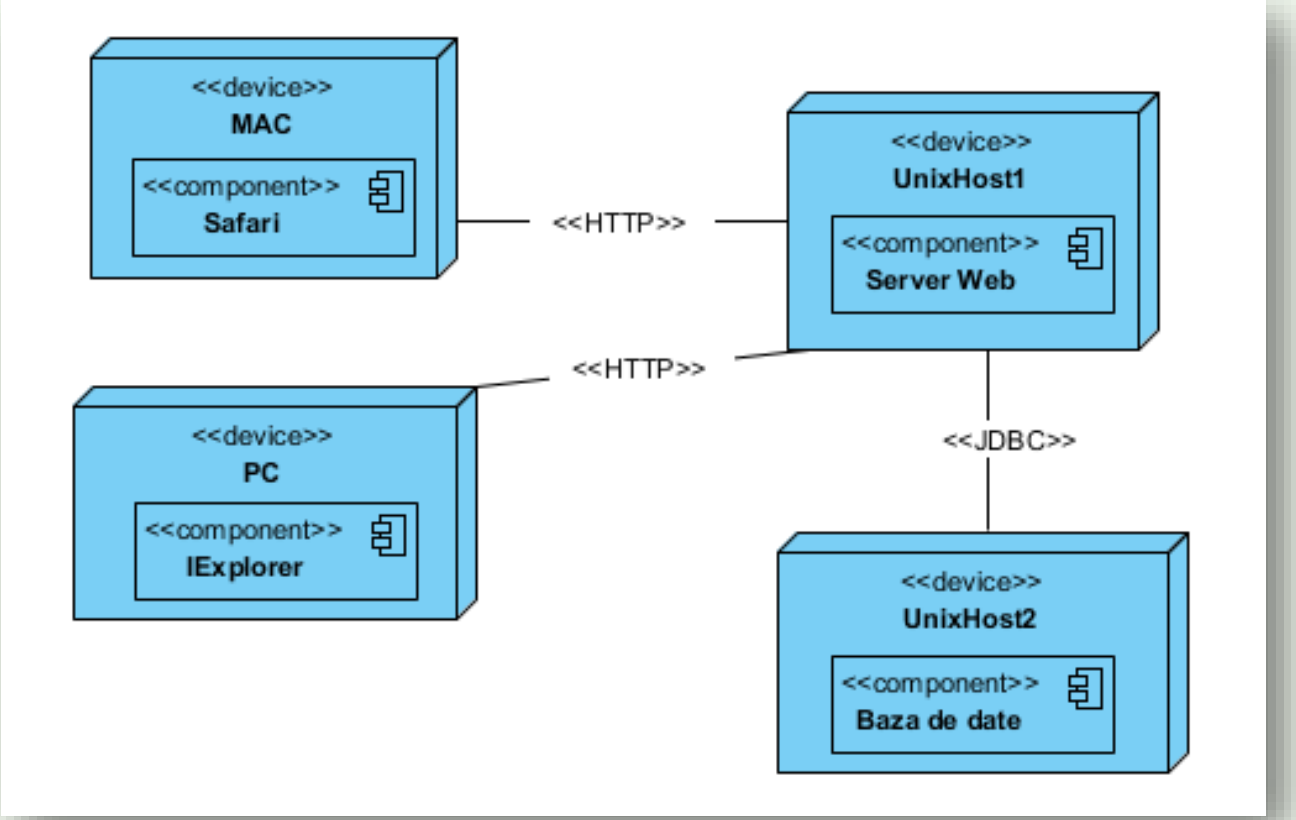
- Deployment diagrams may include two type of nodes: devices and execution environments.
  - Devices are computing resources with processing capabilities and the ability to execute programs. Some examples of device nodes include PCs, laptops, and mobile phones.
  - EEN – Execution Environment Node is any computer system that resides within a device node. It could be an operating system, a JVM, or another servlet container, a Web server (Apache orMicrosoft's Internet Information Server (IIS)).



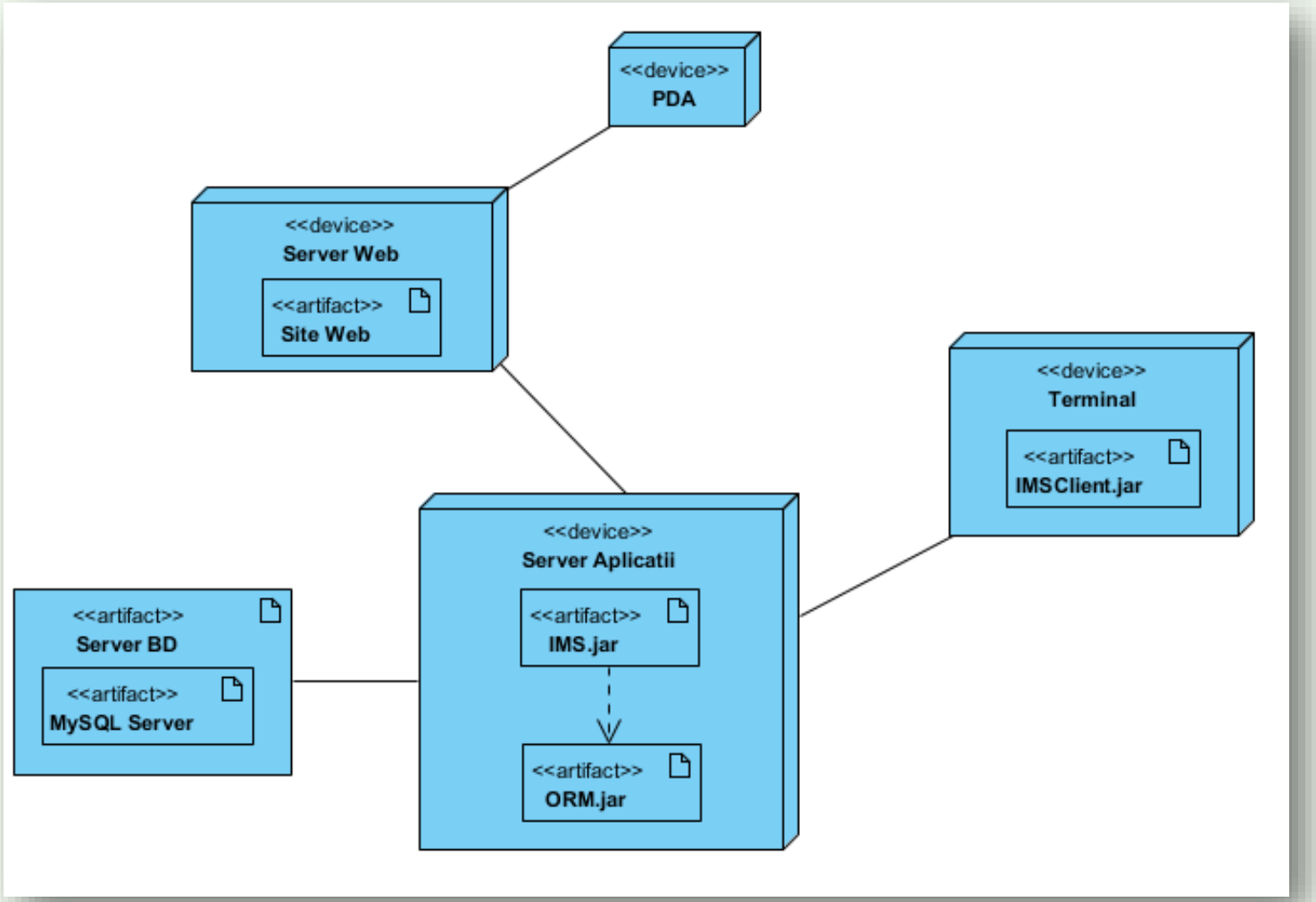


Component hosted by nodes

- Deployment diagrams can be used to represent the components that belong to a node by enclosing the component symbol inside the node symbol.



Deployment diagram – example 1



Deployment diagram – example 2