CS335FZ

Systems Modelling

Dr. Lanlan Gao

Objectives

- Understand why we need to
- Understand the fundamental system modelling perspectives
    - Context models
    - Interaction models
    - Structure models
    - Behaviour models

- Understand the principal diagram types in the Unified Modeling Language (UML)

- Apply UML to system modelling

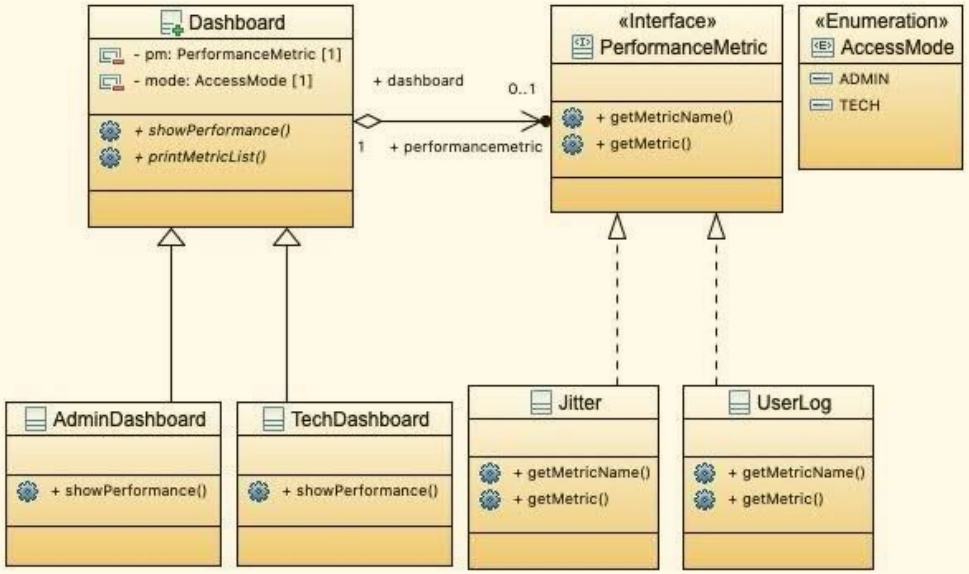
What is a Model?

“A model is an abstraction of the system being studied.”

--Sommerville, I., 2011. Software engineering 9th Edition.

“A model is an abstract representation of a specification, a design or a system, from a particular point of view.”

-- Stevens, P. and Pooley, R., 1999. Using UML: software engineering with objects and components.



Graphical Representation

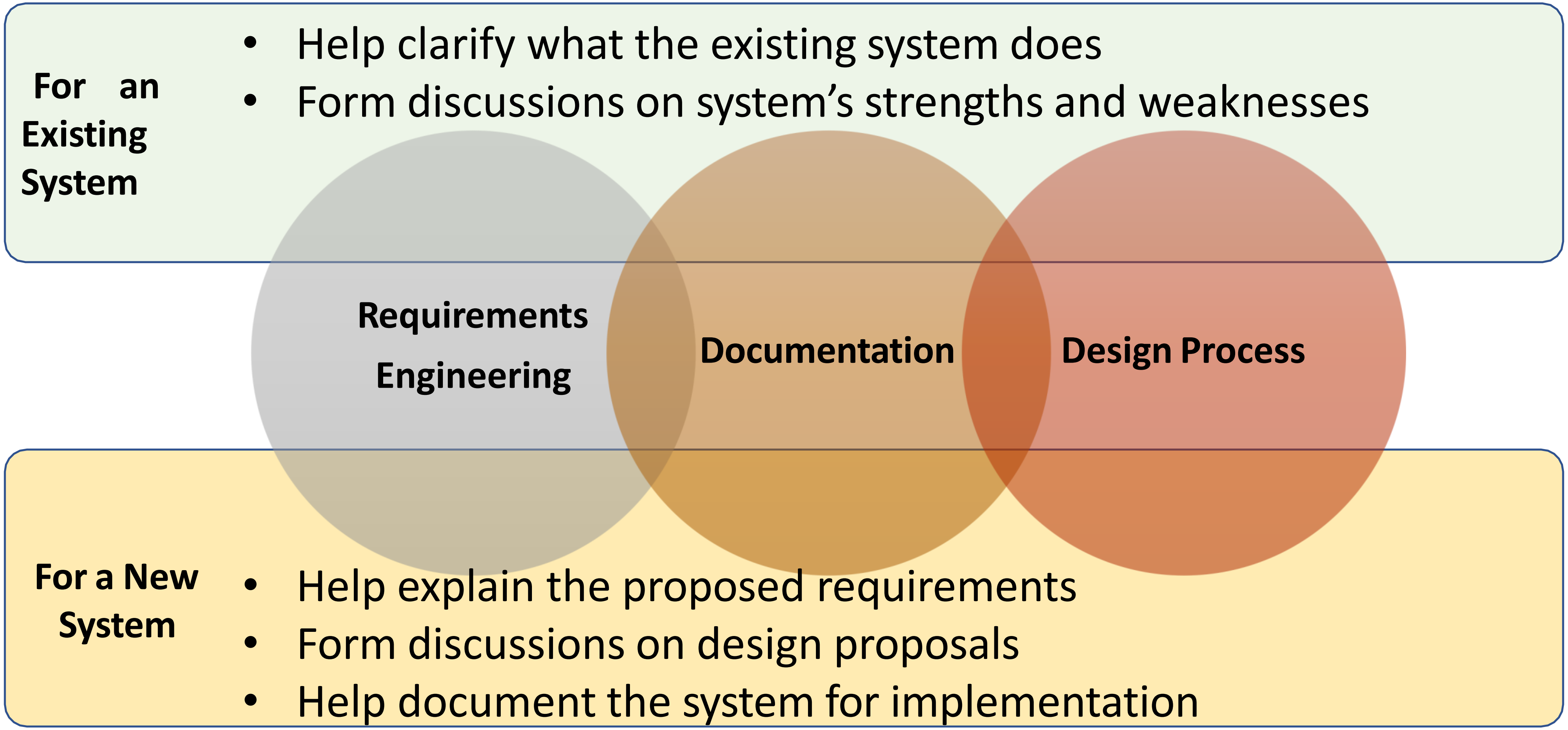
(UML Class Diagram <Network Monitor>)

$$\frac{\beta_0 \vdash^{\mathcal{D}} D_1 \fatsemi \sigma_1 \quad \ldots \quad \beta_{n-1} \vdash^{\mathcal{D}} D_n \fatsemi \sigma_n}{\begin{array}{l}\Lambda \vdash^{\mathcal{S}} \texttt{ZED}\ \text{section}\ i \\ \qquad\qquad \text{parents}\ i_1, \ldots, i_m\ \texttt{END} \\ \qquad\quad D_1 \fatsemi \sigma_1 \ldots D_n \fatsemi \sigma_n \ \fatsemi \Gamma\end{array}} \left(\begin{array}{l} i \notin \mathit{dom}\ \Lambda \\ \{i_1, \ldots, i_m\} \subseteq \mathit{dom}\ \Lambda \\ \gamma_{-1} = \text{if}\ \ i = \mathit{prelude}\ \text{then}\ \{\}\ \text{else}\ \Lambda\ \mathit{prelude} \\ \gamma_0 = \gamma_{-1} \cup \Lambda\ i_1 \cup \ldots \cup \Lambda\ i_m \\ \beta_0 = \gamma_0 \fatsemi \mathit{second} \\ \mathit{disjoint}\ \langle \mathit{dom}\ \sigma_1, \ldots, \mathit{dom}\ \sigma_n \rangle \\ \Gamma \in (\_ \mathrel{+\!\!\!\!\twoheadrightarrow} \_) \\ \Gamma = \gamma_0 \cup \{j : \texttt{NAME};\ \tau : \texttt{Type} \mid j \mapsto \tau \in \sigma_1 \cup \ldots \cup \sigma_n \bullet j \mapsto (i, \tau)\} \\ \beta_1 = \beta_0 \cup \sigma_1 \\ \quad \vdots \\ \beta_{n-1} = \beta_{n-2} \cup \sigma_{n-1} \end{array}\right)$$

Mathematical Representation

(Z Formal Specification <Inheriting Section>)

The Purposes of Building Models



For an Existing System

- Help clarify what the existing system does
- Form discussions on system’s strengths and weaknesses

Requirements Engineering

Documentation

Design Process

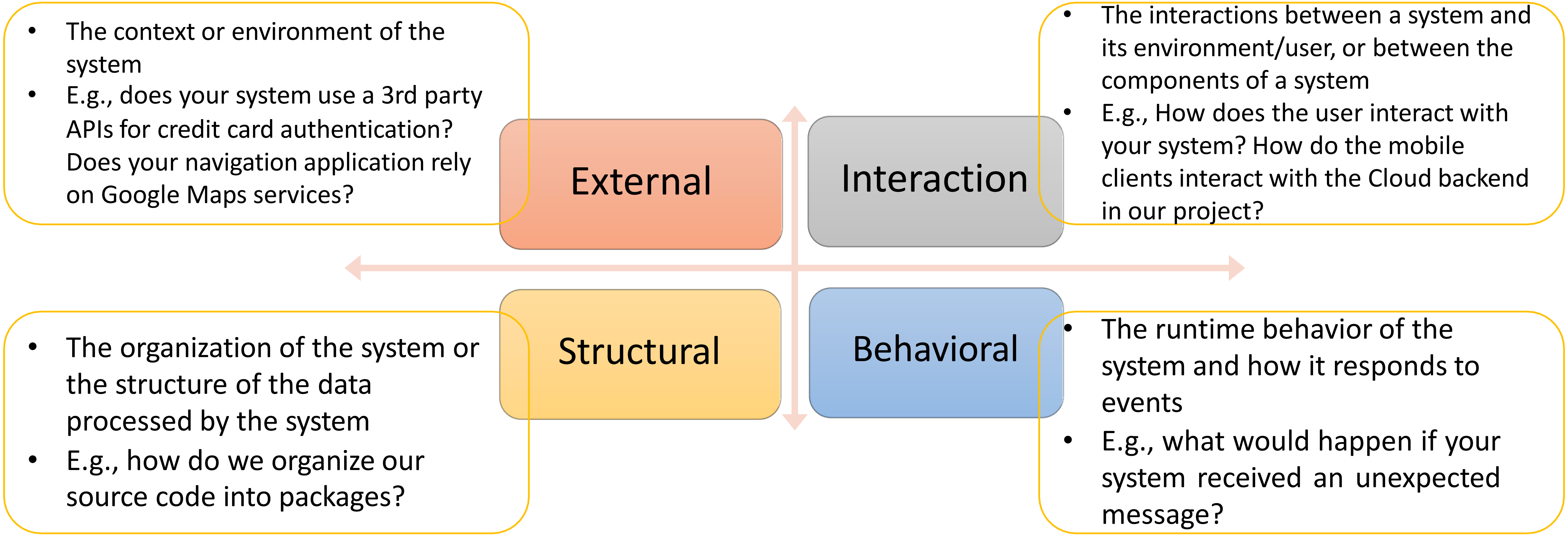
For a New System

- Help explain the proposed requirements
- Form discussions on design proposals
- Help document the system for implementation

A system model is not a complete representation of system. It is an alternative representation of the system.

System Modelling Perspectives

> *"System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system."*
>
> *--Sommerville, I., 2011. Software engineering 9th Edition.*



External

- The context or environment of the system
- E.g., does your system use a 3rd party APIs for credit card authentication? Does your navigation application rely on Google Maps services?

Interaction

- The interactions between a system and its environment/user, or between the components of a system
- E.g., How does the user interact with your system? How do the mobile clients interact with the Cloud backend in our project?

Structural

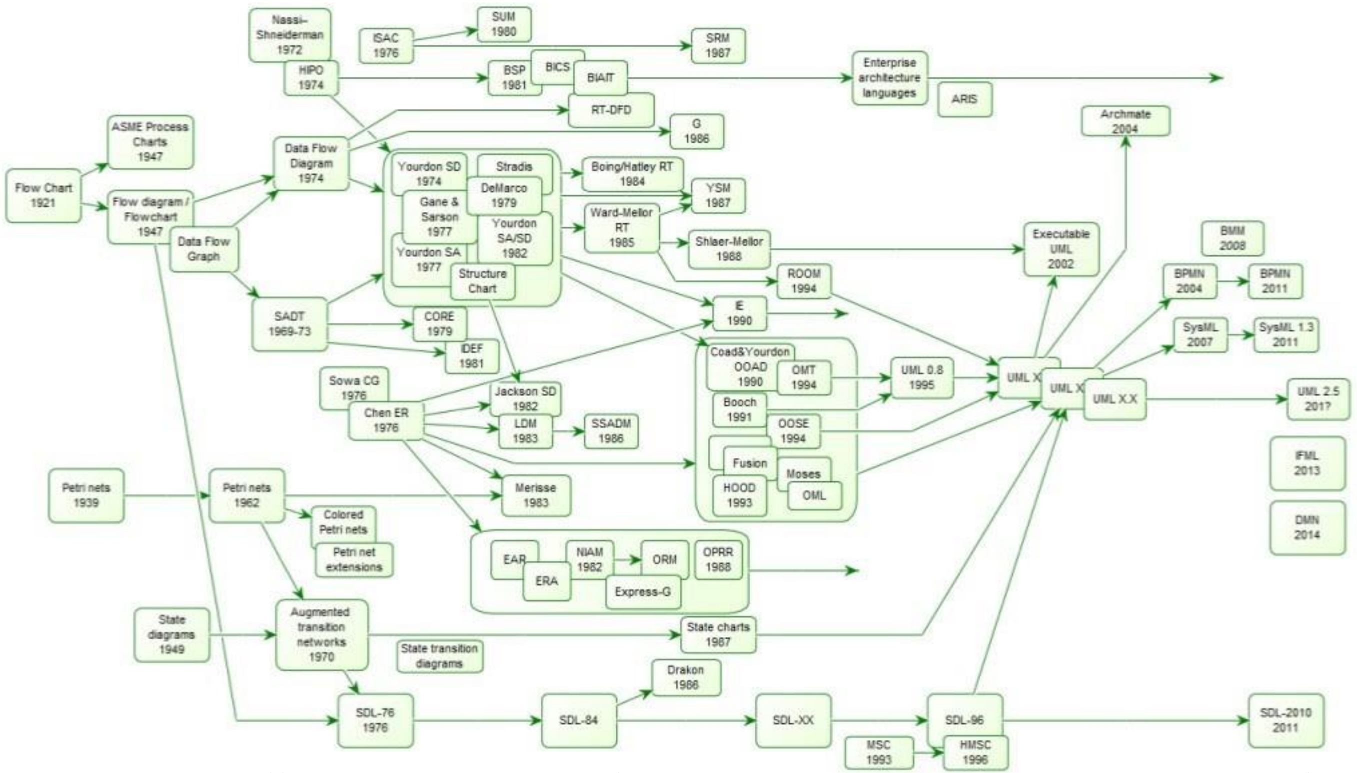
- The organization of the system or the structure of the data processed by the system
- E.g., how do we organize our source code into packages?

Behavioral

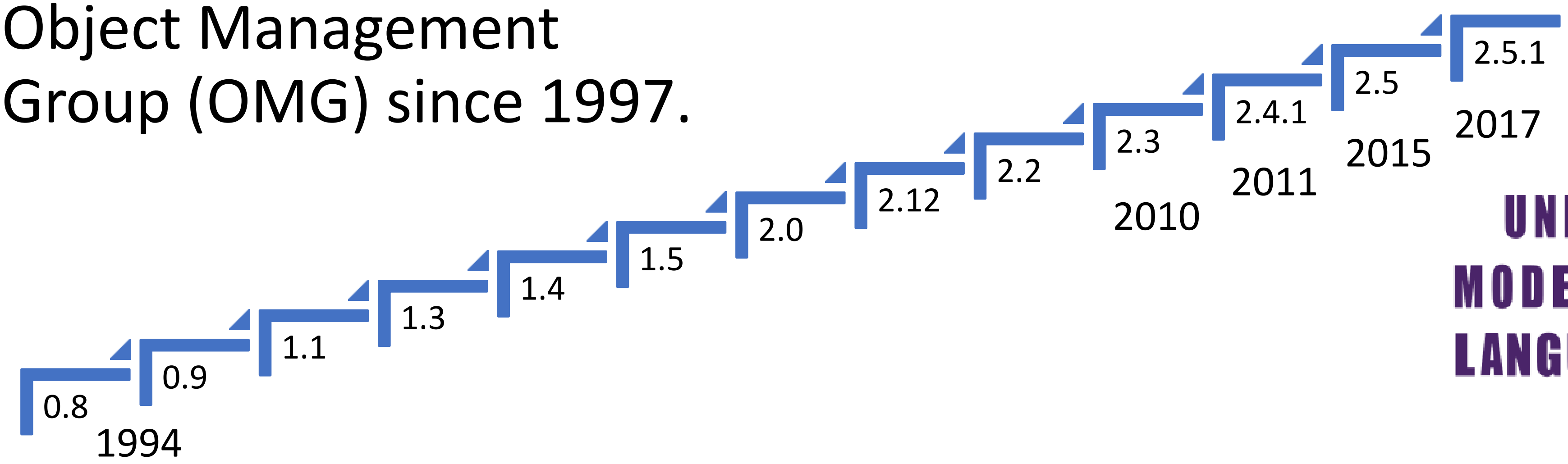
- The runtime behavior of the system and how it responds to events
- E.g., what would happen if your system received an unexpected message?

Unified Modeling Language (UML)

- Designed and developed to standardize the notational systems for software design (1994 – 1996).
- Adopted and managed by Object Management Group (OMG) since 1997.

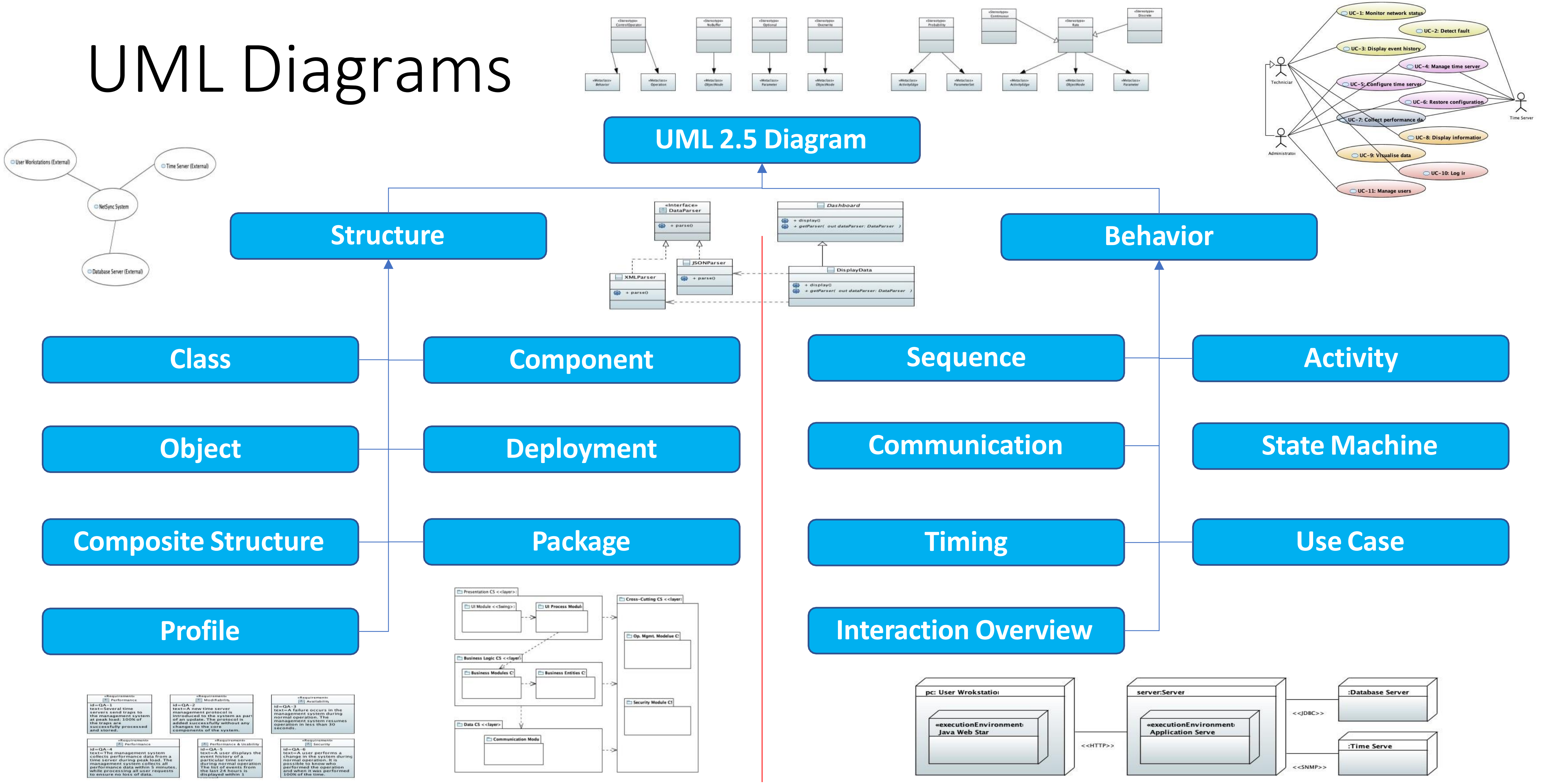


Source: *https://modeling-languages.com/history-modeling-languages-one-picture-j-p-tolvanen/*



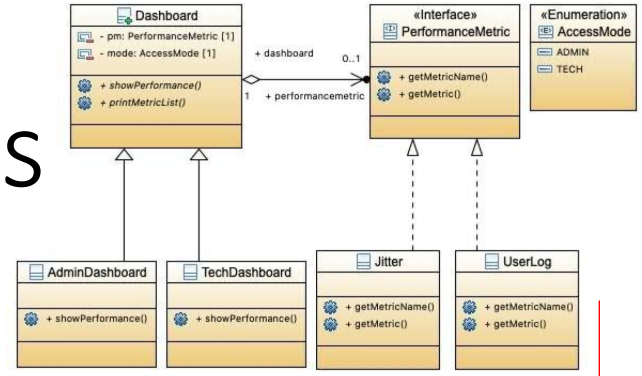
UNIFIED MODELING LANGUAGE™ UML®

UML Diagrams



UML Diagrams: *Object Management Group. OMG Unified Modeling Language (UML), Superstructure. Technical Report Version 2.5.1, OMG, 2017.*

UML Tools



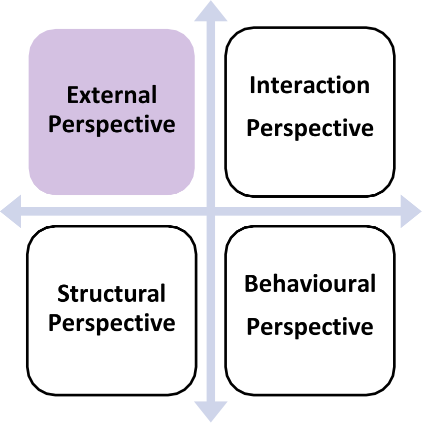
```
<?xml version="1.0" encoding="UTF-8"?>
<uml:Model xmi:version="20131001" xmlns:xmi="http://www.omg.org/spec/XMI/20131001" xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
xmlns:uml="http://www.eclipse.org/uml2/5.0.0/UML" xmi:id="_24HiIE5zEeuE1cFsHgqcjg" name="SESP">\
 <packagedElement xmi:type="uml:Class" xmi:id="_JFHNUGSJEeuiFuOemMXICg" name="Dashboard" visibility="public">
  <ownedAttribute xmi:type="uml:Property" xmi:id="_sTIaQGSJEeuiFuOemMXICg" name="pm" visibility="private" type="_TRW-4GSJEeuiFuOemMXICg"/>
  <ownedAttribute xmi:type="uml:Property" xmi:id="_3lffUGSJEeuiFuOemMXICg" name="mode" visibility="private" type="_YbHG8GSJEeuiFuOemMXICg"/>
  <ownedAttribute xmi:type="uml:Property" xmi:id="_QRzLkGSLEeuiFuOemMXICg" name="performancemetric" type="_TRW-4GSJEeuiFuOemMXICg" aggregation="shared"
association="_QRvhMGSLEeuiFuOemMXICg">
   <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_QR1n0GSLEeuiFuOemMXICg"/>
   <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_QR1n0WSLEeuiFuOemMXICg" value="1"/>
  </ownedAttribute>
  <ownedOperation xmi:type="uml:Operation" xmi:id="_95MSAGSJEeuiFuOemMXICg" name="showPerformance" isAbstract="true"/>
  <ownedOperation xmi:type="uml:Operation" xmi:id="_G73zcGSKEeuiFuOemMXICg" name="printMetricList" isAbstract="true"/>
 </packagedElement>
 <packagedElement xmi:type="uml:Interface" xmi:id="_TRW-4GSJEeuiFuOemMXICg" name="PerformanceMetric">
  <ownedOperation xmi:type="uml:Operation" xmi:id="_UUIOAGSKEeuiFuOemMXICg" name="getMetricName"/>
  <ownedOperation xmi:type="uml:Operation" xmi:id="_eS49MGSKEeuiFuOemMXICg" name="getMetric"/>
 </packagedElement>
 <packagedElement xmi:type="uml:Enumeration" xmi:id="_YbHG8GSJEeuiFuOemMXICg" name="AccessMode">
  <ownedLiteral xmi:type="uml:EnumerationLiteral" xmi:id="_hmgDAGSKEeuiFuOemMXICg" name="ADMIN"/>
  <ownedLiteral xmi:type="uml:EnumerationLiteral" xmi:id="_i11j8GSKEeuiFuOemMXICg" name="TECH"/>
 </packagedElement>
 <packagedElement xmi:type="uml:Class" xmi:id="_clF0EGSJEeuiFuOemMXICg" name="AdminDashboard">
  <generalization xmi:type="uml:Generalization" xmi:id="_-xJ_UGSKEeuiFuOemMXICg" general="_JFHNUGSJEeuiFuOemMXICg"/>
  <ownedOperation xmi:type="uml:Operation" xmi:id="_6s5fsGSKEeuiFuOemMXICg" name="showPerformance"/>
 </packagedElement>
 <packagedElement xmi:type="uml:Class" xmi:id="_e4SlwGSJEeuiFuOemMXICg" name="TechDashboard">
  <generalization xmi:type="uml:Generalization" xmi:id="_AYnsYGSLEeuiFuOemMXICg" general="_JFHNUGSJEeuiFuOemMXICg"/>
  <ownedOperation xmi:type="uml:Operation" xmi:id="_2g-R0GSKEeuiFuOemMXICg" name="showPerformance"/>
 </packagedElement>
 <packagedElement xmi:type="uml:Class" xmi:id="_gLFDUGSJEeuiFuOemMXICg" name="Jitter">
  <interfaceRealization xmi:type="uml:InterfaceRealization" xmi:id="_QEWeEGSOEeuiFuOemMXICg" client="_gLFDUGSJEeuiFuOemMXICg" supplier="_TRW-
4GSJEeuiFuOemMXICg" contract="_TRW-4GSJEeuiFuOemMXICg"/>
  <ownedOperation xmi:type="uml:Operation" xmi:id="_vkmVkGSKEeuiFuOemMXICg" name="getMetricName"/>
  <ownedOperation xmi:type="uml:Operation" xmi:id="_x41pEGSKEeuiFuOemMXICg" name="getMetric"/>
 </packagedElement>
 <packagedElement xmi:type="uml:Class" xmi:id="_g6K98GSJEeuiFuOemMXICg" name="UserLog">
  <interfaceRealization xmi:type="uml:InterfaceRealization" xmi:id="_LE63AGSLEeuiFuOemMXICg" client="_g6K98GSJEeuiFuOemMXICg" supplier="_TRW-
4GSJEeuiFuOemMXICg" contract="_TRW-4GSJEeuiFuOemMXICg"/>
  <ownedOperation xmi:type="uml:Operation" xmi:id="_nN_P8GSKEeuiFuOemMXICg" name="getMetricName"/>
  <ownedOperation xmi:type="uml:Operation" xmi:id="_psItYGSKEeuiFuOemMXICg" name="getMetric"/>
 </packagedElement>
 <packagedElement xmi:type="uml:Realization" xmi:id="_JdiCcGSLEeuiFuOemMXICg" client="_gLFDUGSJEeuiFuOemMXICg" supplier="_24HiIE5zEeuE1cFsHgqcjg"/>
 <packagedElement xmi:type="uml:Association" xmi:id="_QRvhMGSLEeuiFuOemMXICg" memberEnd="_QRzLkGSLEeuiFuOemMXICg _QR2O4GSLEeuiFuOemMXICg">
  <eAnnotations xmi:type="ecore:EAnnotation" xmi:id="_QRx9cGSLEeuiFuOemMXICg" source="org.eclipse.papyrus">
   <details xmi:type="ecore:EStringToStringMapEntry" xmi:id="_QRykgGSLEeuiFuOemMXICg" key="nature" value="UML_Nature"/>
  </eAnnotations>
  <ownedEnd xmi:type="uml:Property" xmi:id="_QR2O4GSLEeuiFuOemMXICg" name="dashboard" type="_JFHNUGSJEeuiFuOemMXICg"
association="_QRvhMGSLEeuiFuOemMXICg"/>
 </packagedElement>
</uml:Model>
```

Rational
the software development company
be liberated
Rational Rose
Rational Software Corporation

ArgoUML

Borland Together

System Modelling – External Perspective



- Context Model
  - Created at the early stage of the requirements engineering
  - To know and to decide the boundary of the system being developed
  - To establish a high-level view on the interactions between the system and its operational environment without details.
  - Use simple block diagrams or empty class diagrams

UML does NOT provide dedicated types of diagram for context model.

- Business Process Model
  - Model business processes
  - Depict how systems are involved in a particular business process
  - Use activity diagrams or dedicated Business Process Model and Notation (BPMN)

BPMN is a standard for business process modelling that provides graphical notations, similar to UML, for expressing business processes. (https://www.omg.org/spec/BPMN/2.0/)

Context modeling and business process modeling are both important concepts in the realm of business analysis and system design, but they serve different purposes.

- Context Modeling:

- Definition: Context modeling focuses on understanding the environment or setting in which a system operates. It involves identifying the various stakeholders, their roles, interactions, constraints, and external factors that might influence the system.
- Purpose: The purpose of context modeling is to provide a holistic view of the system’s environment so that designers and stakeholders can better understand the scope and requirements of the system. It helps in defining boundaries and identifying dependencies on external entities.
- Techniques: Context modeling techniques may include stakeholder analysis, use case modeling, scenario analysis, and environmental analysis.

Business Process Modeling:

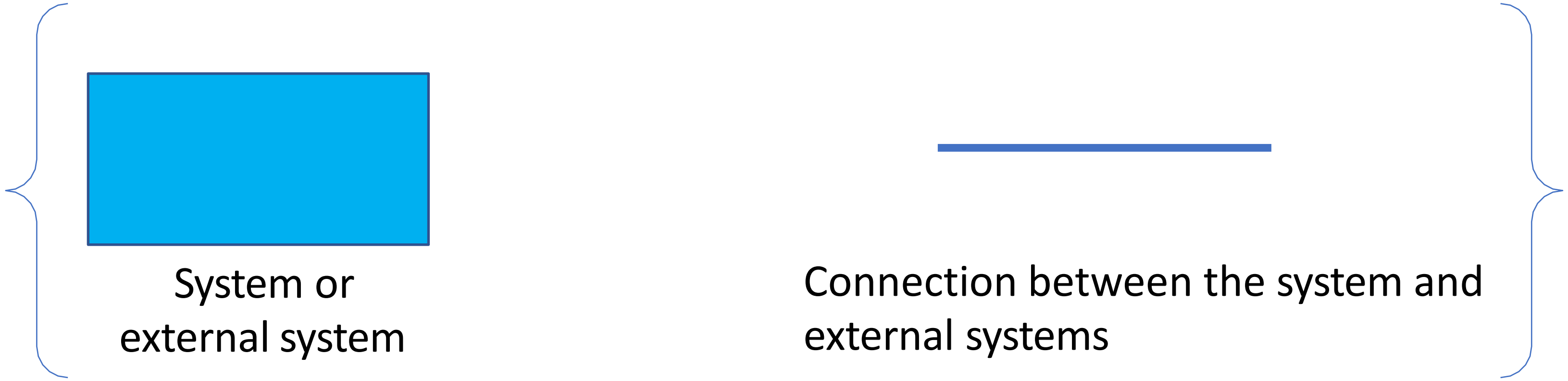
Definition: Business process modeling involves representing the sequence of activities, tasks, decisions, and interactions that occur within an organization to achieve specific business objectives. It aims to capture the structure and flow of business operations.
Purpose: The purpose of business process modeling is to improve efficiency, streamline operations, and facilitate communication and understanding within the organization. It helps in identifying bottlenecks, redundancies, and areas for improvement.
Techniques: Business process modeling techniques may include flowcharts, BPMN (Business Process Model and Notation), UML (Unified Modeling Language) activity diagrams, and process mapping.
Key Differences:

Scope: Context modeling focuses on the broader environment and stakeholders surrounding the system, whereas business process modeling zooms in on the internal operations and workflows of the organization.
Focus: Context modeling emphasizes understanding the context in which the system operates, including external influences and interactions, while business process modeling focuses on the specific activities and processes within the organization.
Outcome: The outcome of context modeling is a comprehensive understanding of the system’s environment and stakeholders, while the outcome of business process modeling is a visual representation of the organization’s workflows and operations.

In summary, while both context modelling and business process modelling are essential for system design and analysis, they serve different purposes and address different aspects of the system and its environment.

Context Model

The elements of context model diagram:



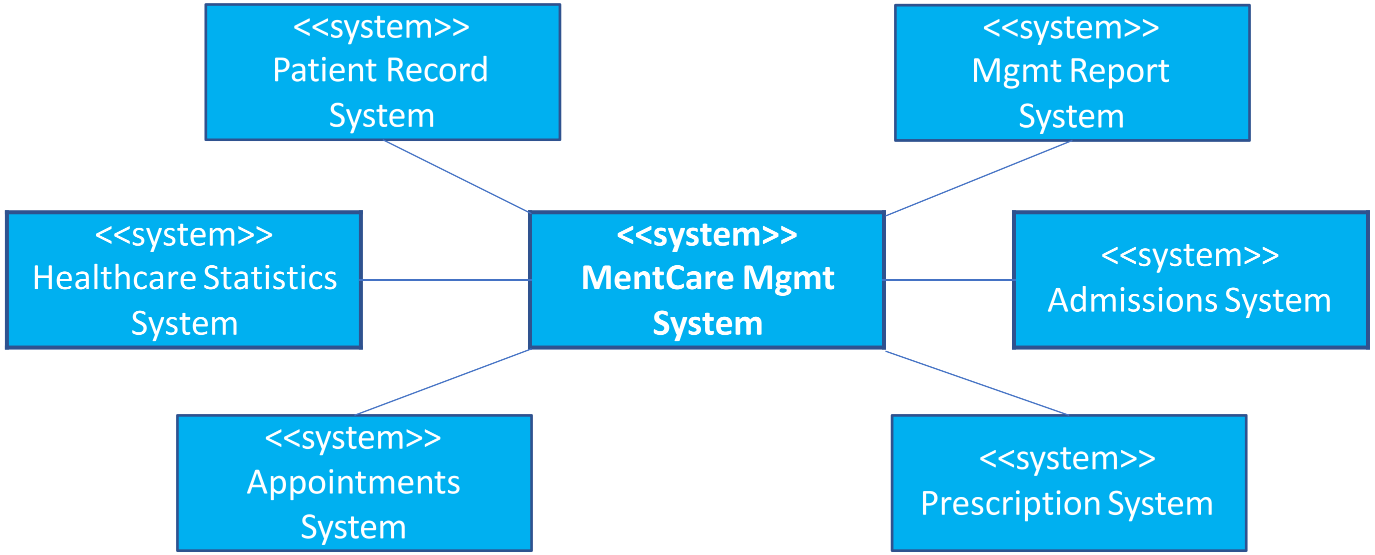
System or external system

Connection between the system and external systems

Project Description:
The Mid-Scotland regional health authority wishes to procure an information system to help manage the care of patients suffering from mental health problems. The overall goals of the system are twofold:

1. To provide better management information about mental healthcare in the region.
2. To provide an improved records system for clinical staff involved in diagnosis and treatment.

Source: *Sommerville, I., 2011. Software engineering 9th Edition*



<<system>> Patient Record System

<<system>> Mgmt Report System

<<system>> Healthcare Statistics System

<<system>> MentCare Mgmt System

<<system>> Admissions System

<<system>> Appointments System

<<system>> Prescription System

Exercise: Context Model

City Transport Service (CTS)

Project Description:
The city tourist information office wants to have an online city transport service for foreign visitors.

An online city transport service is an information system that allows people to plan journeys, book travel tickets for city rail and bus services.

Journey Planning

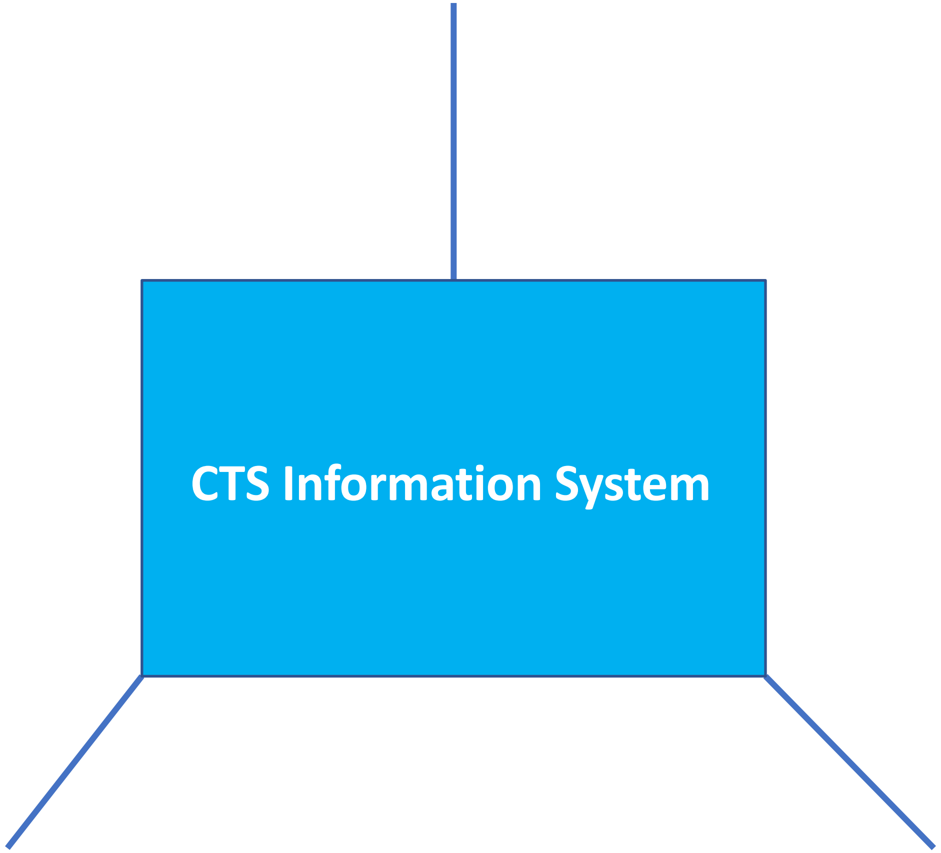
Ticket Booking

Authentication & Registration Service

ID Verification Service

Online Payment Service

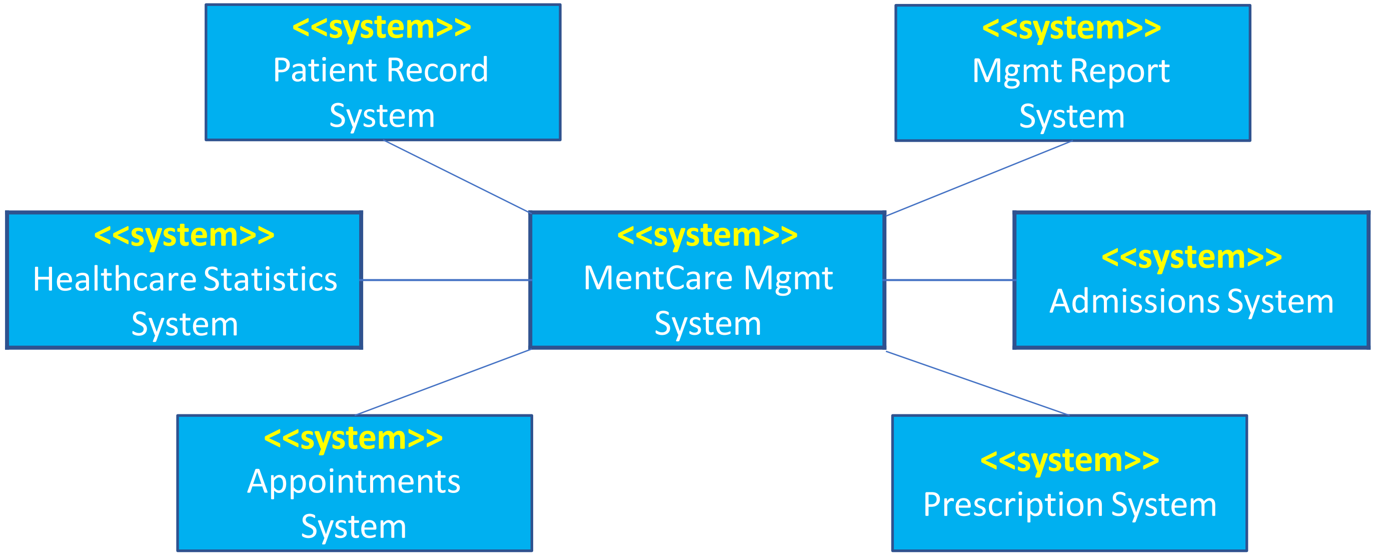
Department of Transport



Terminology: Stereotype

> *"A stereotype is UML's way of attaching extra classifications to model items; it's one of the ways that UML is made extensible."*
>
> *--Stevens, P. and Pooley, R.J., 2006. Using UML: software engineering with objects and components. Pearson Education.*

- It is used to describe a model element
- It is placed close to the element in the diagram,
- It uses a pair of << >> to enclose a *type*, e.g., *<<use>>, <<include>>, <<import>>, <<system>>, etc.*

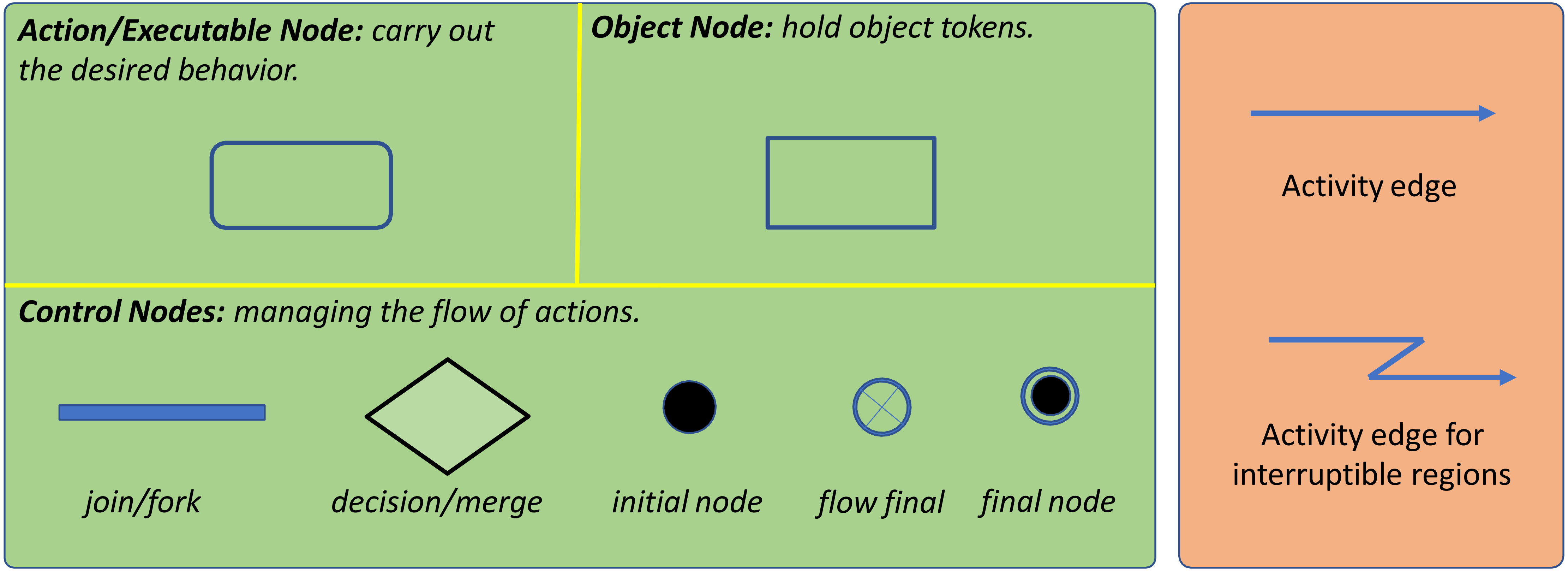


<<system>> Patient Record System
<<system>> Mgmt Report System
<<system>> Healthcare Statistics System
<<system>> MentCare Mgmt System
<<system>> Admissions System
<<system>> Appointments System
<<system>> Prescription System

Elements of Activity Diagram

> *"An Activity is a kind of Behavior that is specified as a graph of nodes interconnected by edges. The flow of execution is modeled as ActivityNodes connected by ActivityEdges"*
>
> *--OMG, O., 2017. OMG Unified Modeling Language (OMG UML) Version 2.5. 1.*

Elements of activity diagram



Action/Executable Node: *carry out the desired behavior.*

Object Node: *hold object tokens.*

Control Nodes: *managing the flow of actions.*

join/fork

decision/merge

initial node

flow final

final node

Activity edge

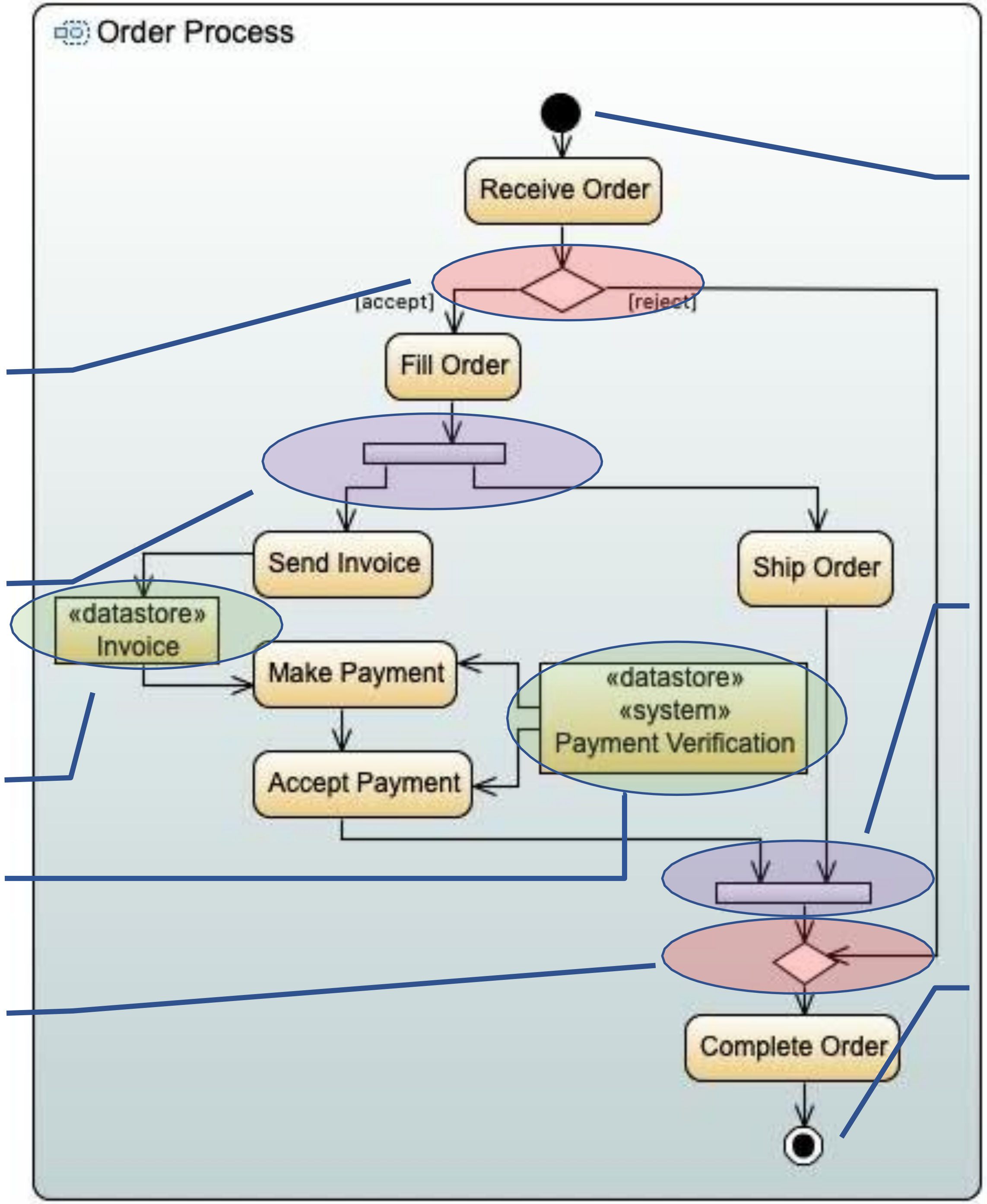
Activity edge for interruptible regions

ActivityNodes

EdgeNodes

NOTE: There are many other advanced notations defined but not presented here.

Business Process Modelling using Activity Diagrams



An ***Initial Node*** is a ***Control Node*** that represents a starting point for executing an ***Activity***.

A ***Decision Node*** is a ***Control Node*** that chooses between outgoing flows.

A ***Fork Node*** is a ***Control Node*** that splits a flow into multiple *concurrent* flows.

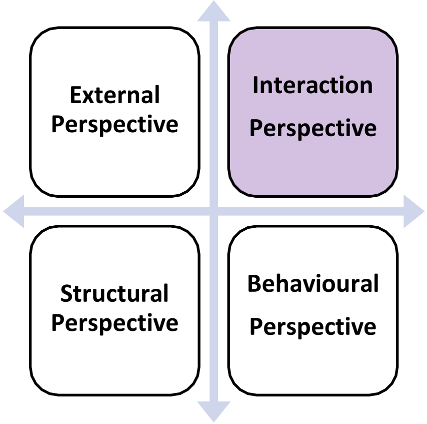
A ***Join Node*** is a ***Control Node*** that synchronizes multiple flows.

An ***Object Node*** holds object tokens, such as, artifacts, systems, data or strings, etc.

A ***Merge Node*** is a ***Control Node*** that brings together multiple flows without synchronization.

A ***Final Node*** is a ***Control Node*** at which the behavior in an ***Activity*** stops.

System Modelling – Interaction Perspective



- **User interaction**, which involves user inputs and outputs;
  - Helps to identify user requirements.

- **System interaction**, interactions between the software system being developed and the systems in its environment;
  - Highlights the communication problems that may arise.

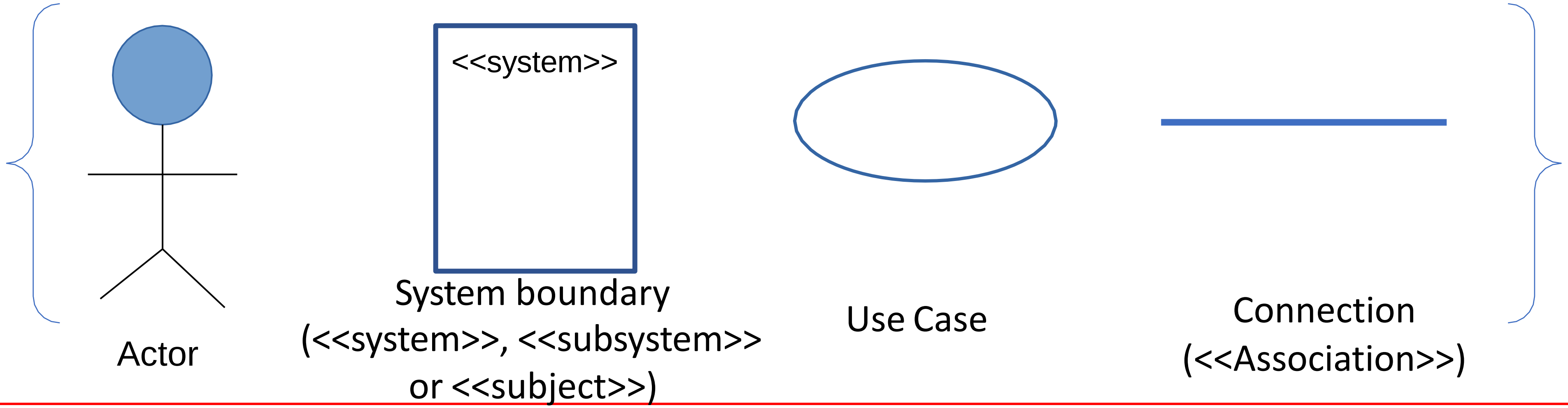
- **Component interaction**, interactions between components of a software system.
  - Help to understand if a proposed system structure is likely to deliver the required system performance and dependability.

Use Cases

User Story Index Card Management System: Update User Story	
Actors	Product Owner
Description	A product owner may modify an existing user story in the system. By modifying a user story, the newly updated information must be validated, for example, Sprint Point must be a numerical value. At the same time, all mandatory fields must be completed, and then saved on persistent storage. The updated user story must be visible to the user immediately.
Data	An existing user story in the system
Stimulus	User command issued by product owner
Response	Confirmation that the user story has been validated, updated and saved
Comments	The person who wants to make changes to the user story must first login as a Product Owner role; the project file must already be loaded in the system.

Use Case Modeling Using UML

The elements of Use Case diagram:



Actor

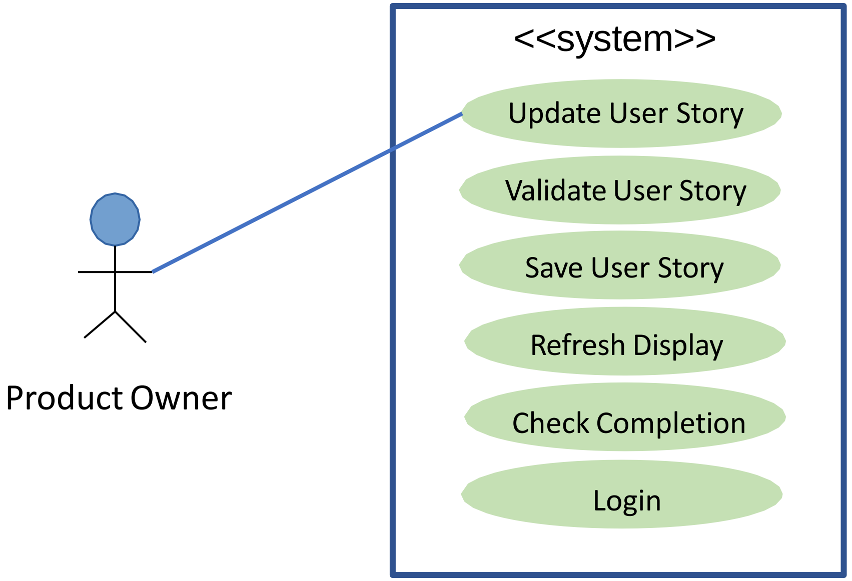
<<system>>

System boundary (<<system>>, <<subsystem>> or <<subject>>)

Use Case

Connection (<<Association>>)

User Story Index Card Management System: Update User Story	
Actors	Product Owner
Description	A product owner may modify an existing user story in the system. By modifying a user story, the newly updated information must be validated, for example, Sprint Point must be a numerical value. At the same time, all mandatory fields must be completed, and then saved on persistent storage. The updated user story must be visible to the user immediately.
Data	An existing user story in the system
Stimulus	User command issued by product owner
Response	Confirmation that the user story has been validated, updated and saved
Comments	The person who wants to make changes to the user story must first login as a Product Owner role; the project file must already be loaded in the system.



Product Owner

<<system>>

Update User Story

Validate User Story

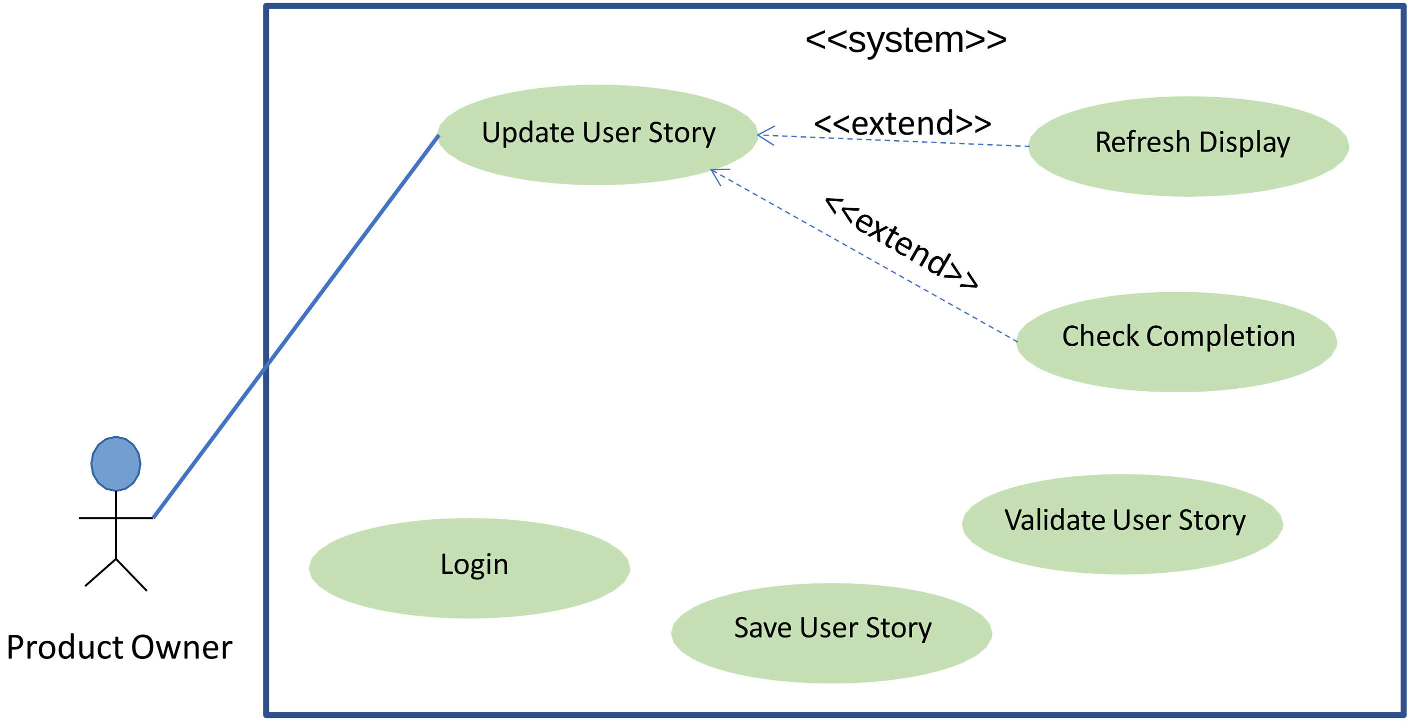
Save User Story

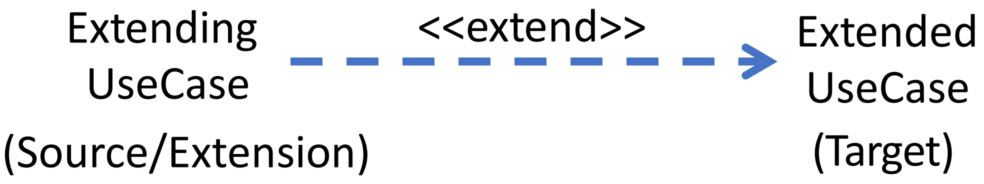
Refresh Display

Check Completion

Login

Use Case Dependency -- Extends

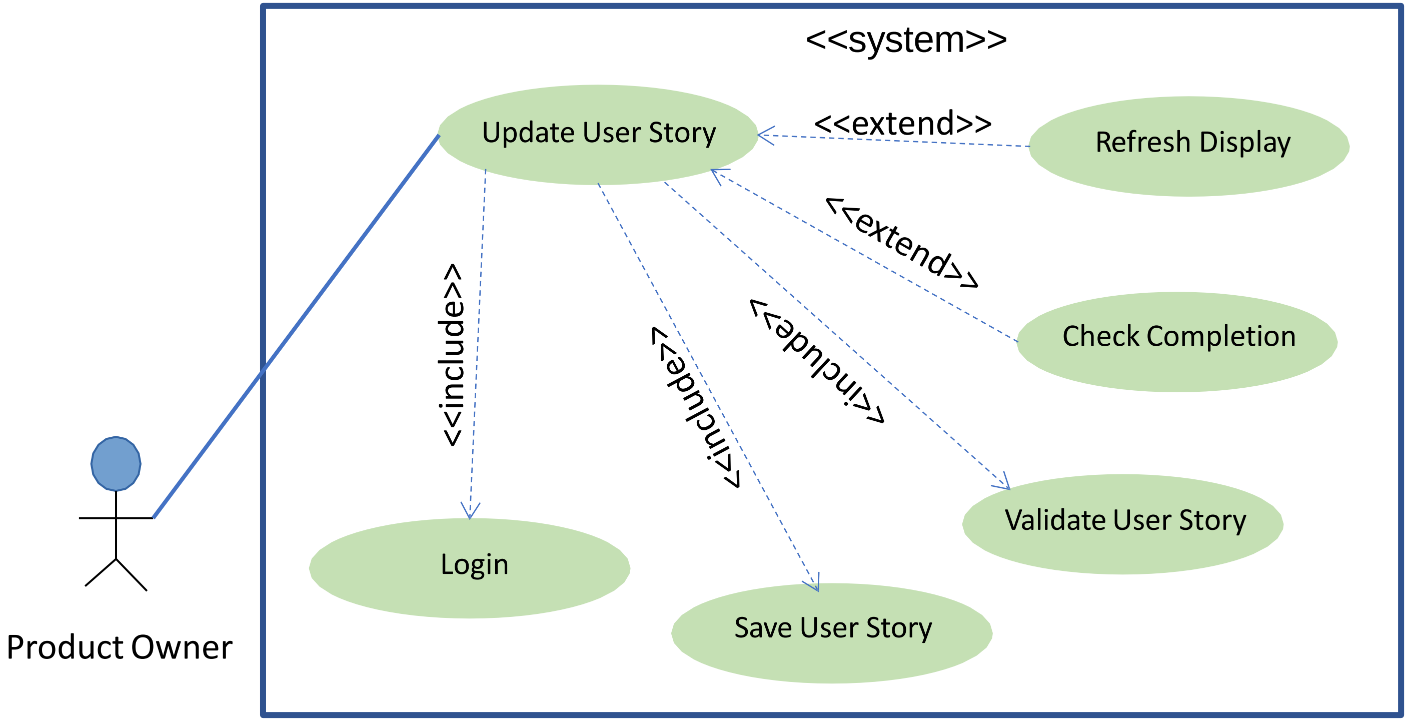



Extending UseCase (Source/Extension)
<<extend>>
Extended UseCase (Target)

Extend is intended to be used:

1. If there are some additional behaviors that should be added (maybe conditionally)
2. *Extended UseCase* is defined independently of the extending UseCase
3. Extending UseCase may not be meaningful by itself

Use Case Dependency -- Include



<<system>>

Update User Story

Refresh Display

Check Completion

Validate User Story

Login

Save User Story

<<extend>>

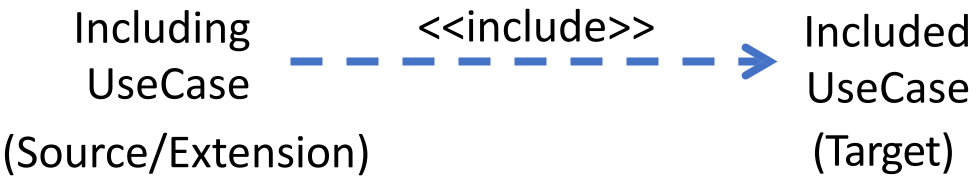
<<extend>>

<<include>>

<<include>>

<<include>>

Product Owner



Including UseCase

(Source/Extension)

<<include>>

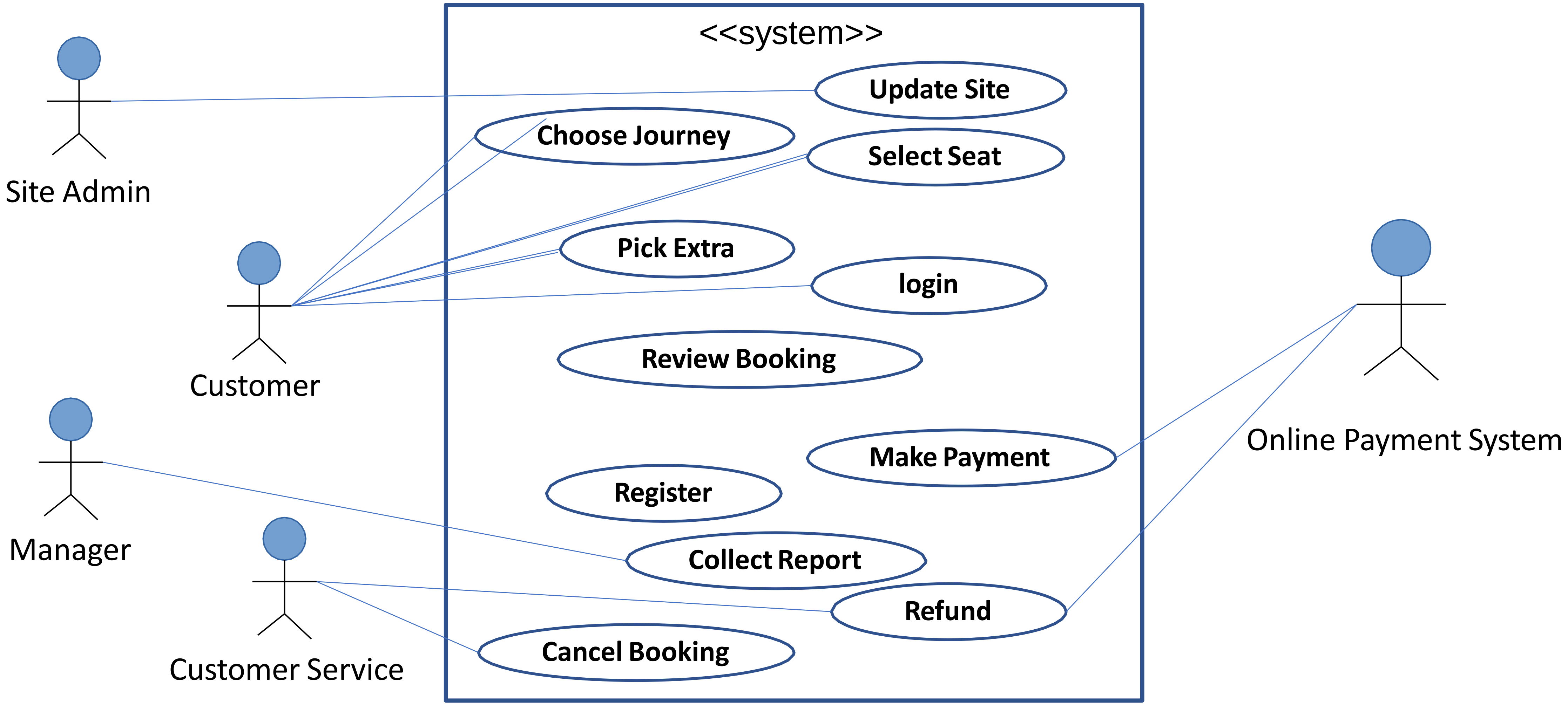
Included UseCase

(Target)

Include is intended to be used:

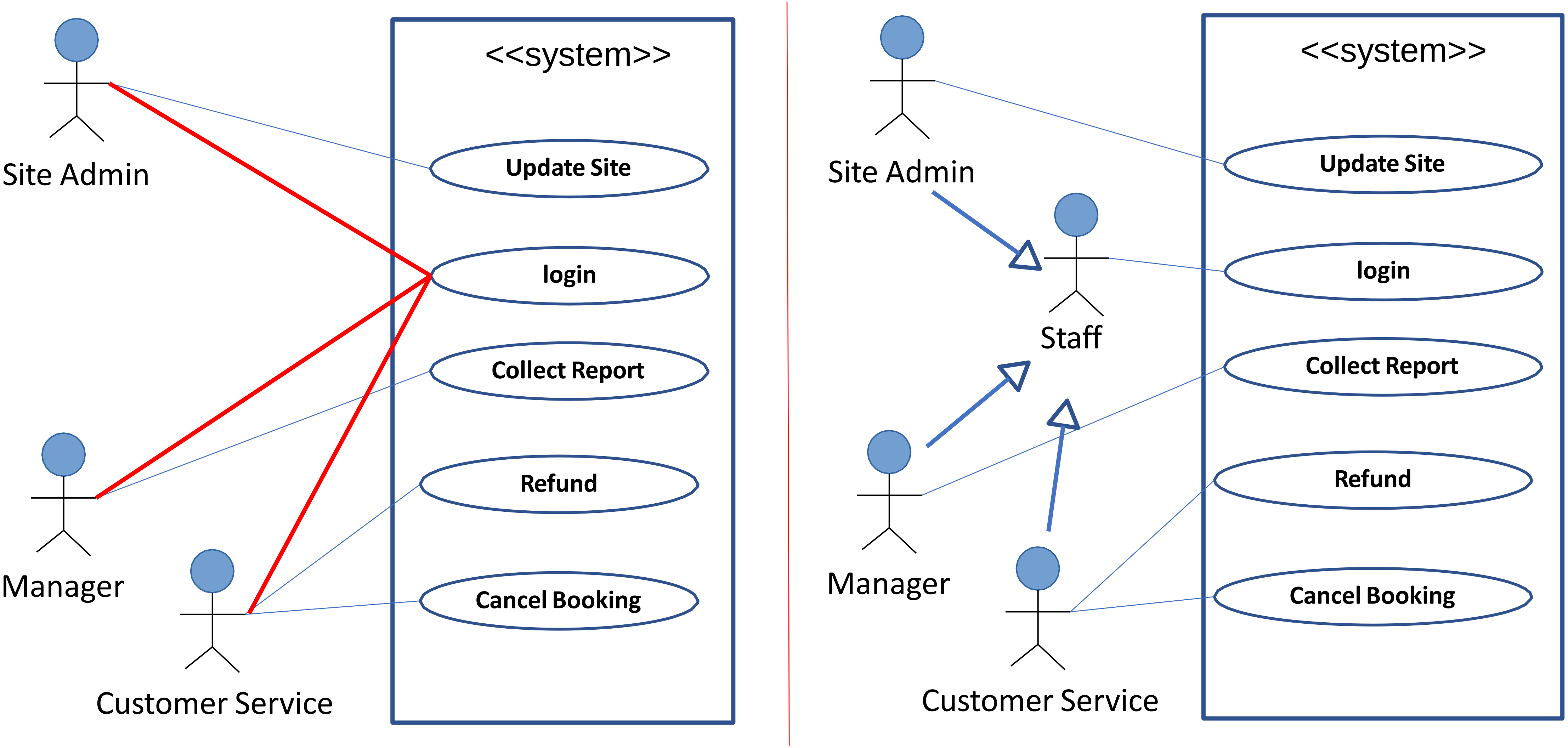
1. The including use case may depend on the changes produce by executing the included use cases(s).
2. The included use case(s) must be available for the behaviors of the including use case to be completed.
3. It can also be used when there are common parts of the behaviors of the two or more use case(s).

Use Case Brainstorming – Train Ticket Online Booking System

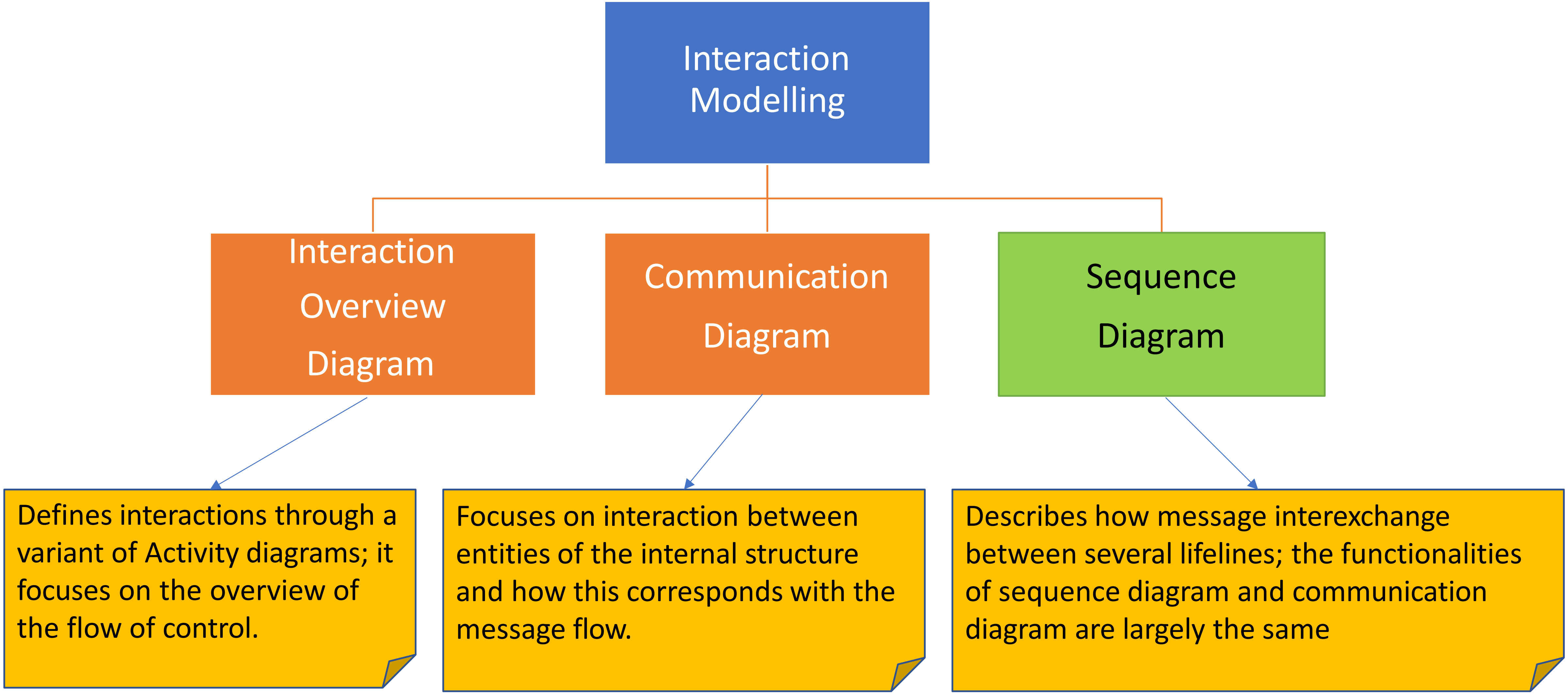


Use Case Dependency -- Generalization

Generalization ≈ *"is a"* →



Interaction Modelling



Interaction Modelling

Interaction Overview Diagram

Communication Diagram

Sequence Diagram

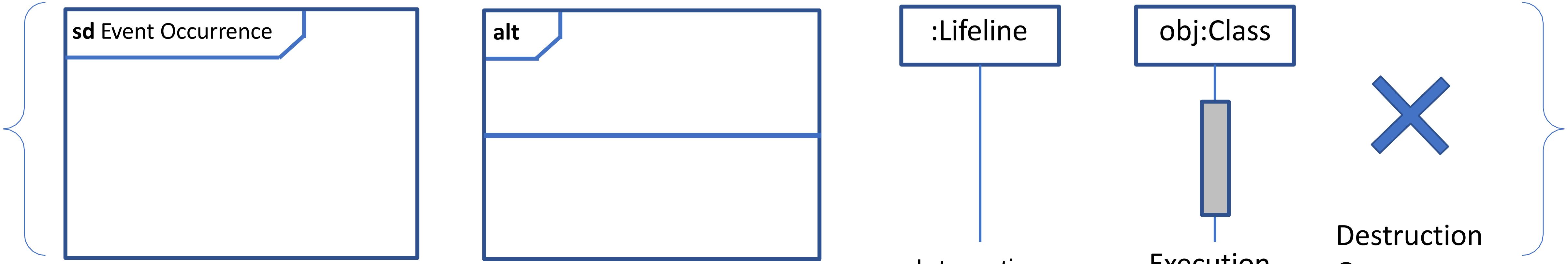
Defines interactions through a variant of Activity diagrams; it focuses on the overview of the flow of control.

Focuses on interaction between entities of the internal structure and how this corresponds with the message flow.

Describes how message interexchange between several lifelines; the functionalities of sequence diagram and communication diagram are largely the same

Sequence Diagrams

Graphic Nodes



Frame for Interaction

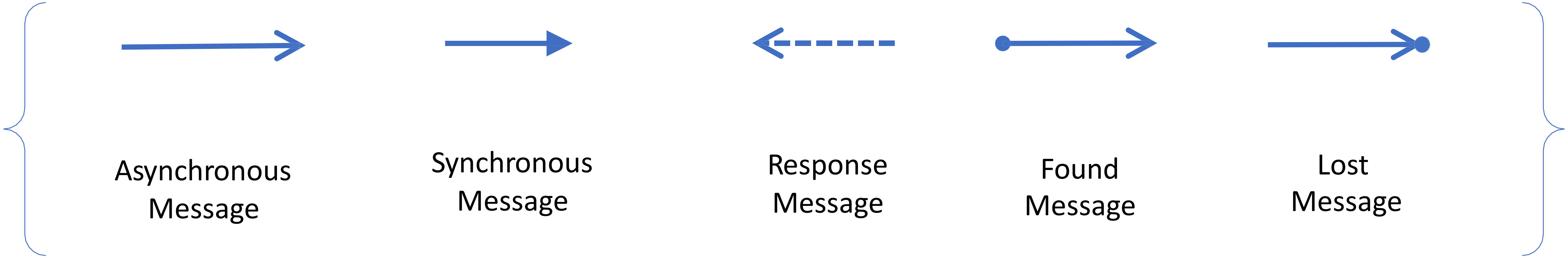
Alternative Combined Fragment

Interaction Lifeline

Execution Specification

Destruction Occurrence Specification

Graphic Paths



Asynchronous Message

Synchronous Message

Response Message

Found Message

Lost Message

Example: User Story Index Card Management System -- Login

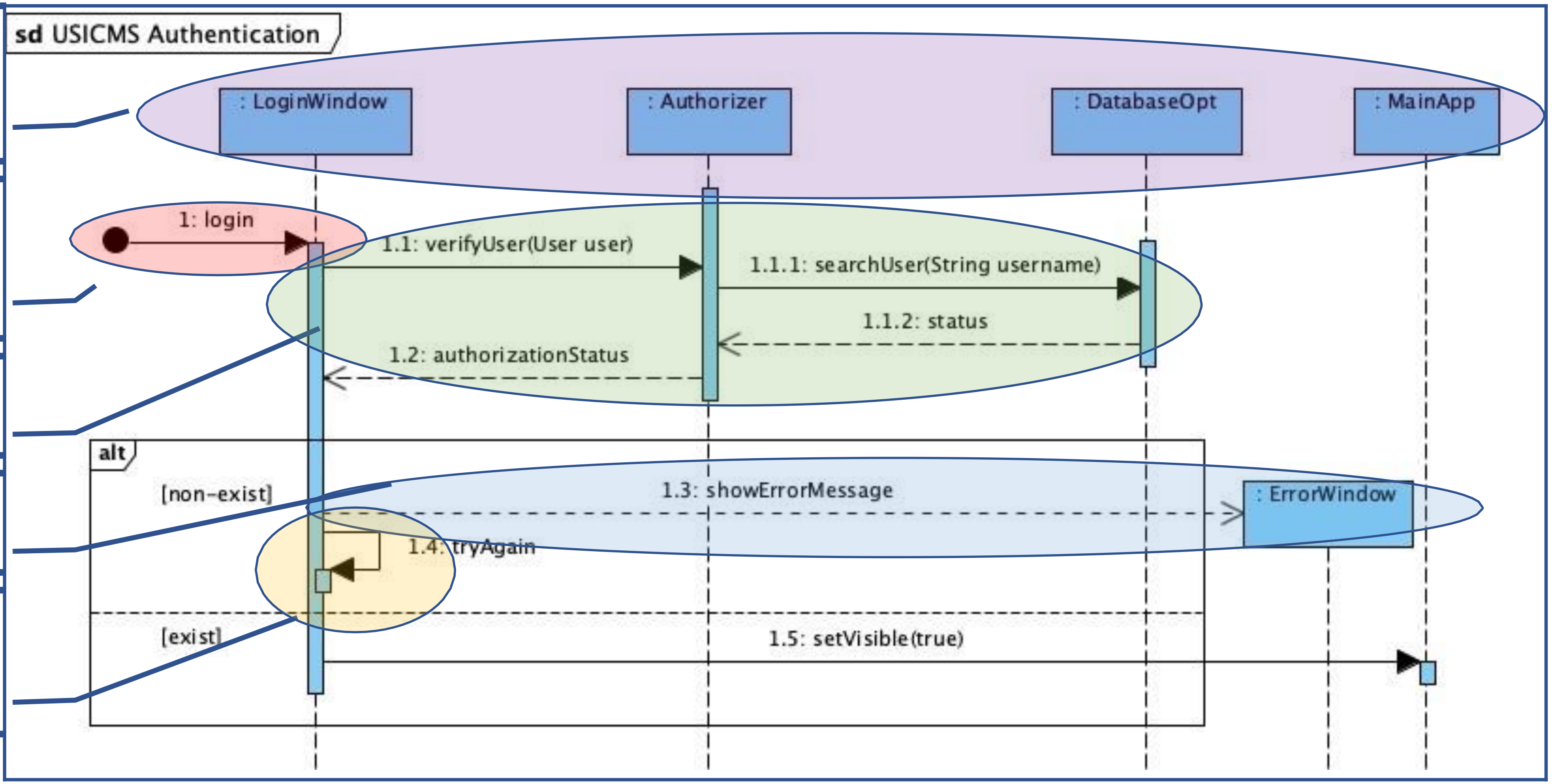
A ***lifeline*** represents an individual participant or objects in the interaction.

A ***Found Message*** is a message with known receiver, but the sender is not described within the specification.

Synchronous messages (or calls) with ***replies.***

A ***Create message*** represents the instantiation of a ***lifeline.***

A ***Recursive Message*** represents the invocation of message of the same ***lifeline.***



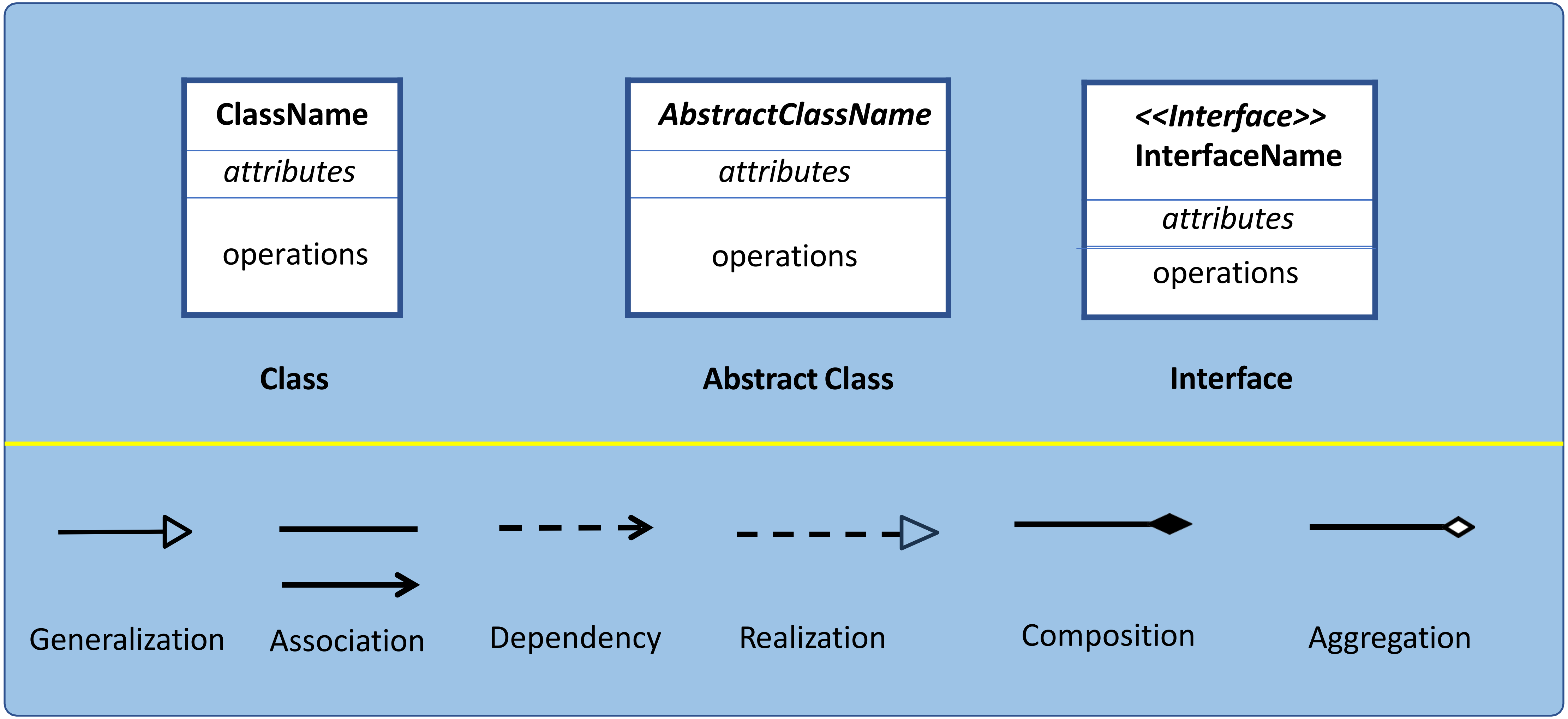
System Modelling – Structural Perspective

- Model the organization of a system in terms of functional components and their relationships
  - The organization of the system design (static structure)
  - The organization of the system when it is executing (dynamic structure)

- Used for discussing and designing the system architecture

- Class diagrams are used for modeling the static structure of a system

Class Diagram



ClassName

attributes

operations

Class

AbstractClassName

attributes

operations

Abstract Class

<<Interface>>
InterfaceName

attributes

operations

Interface

Generalization

Association

Dependency

Realization

Composition

Aggregation

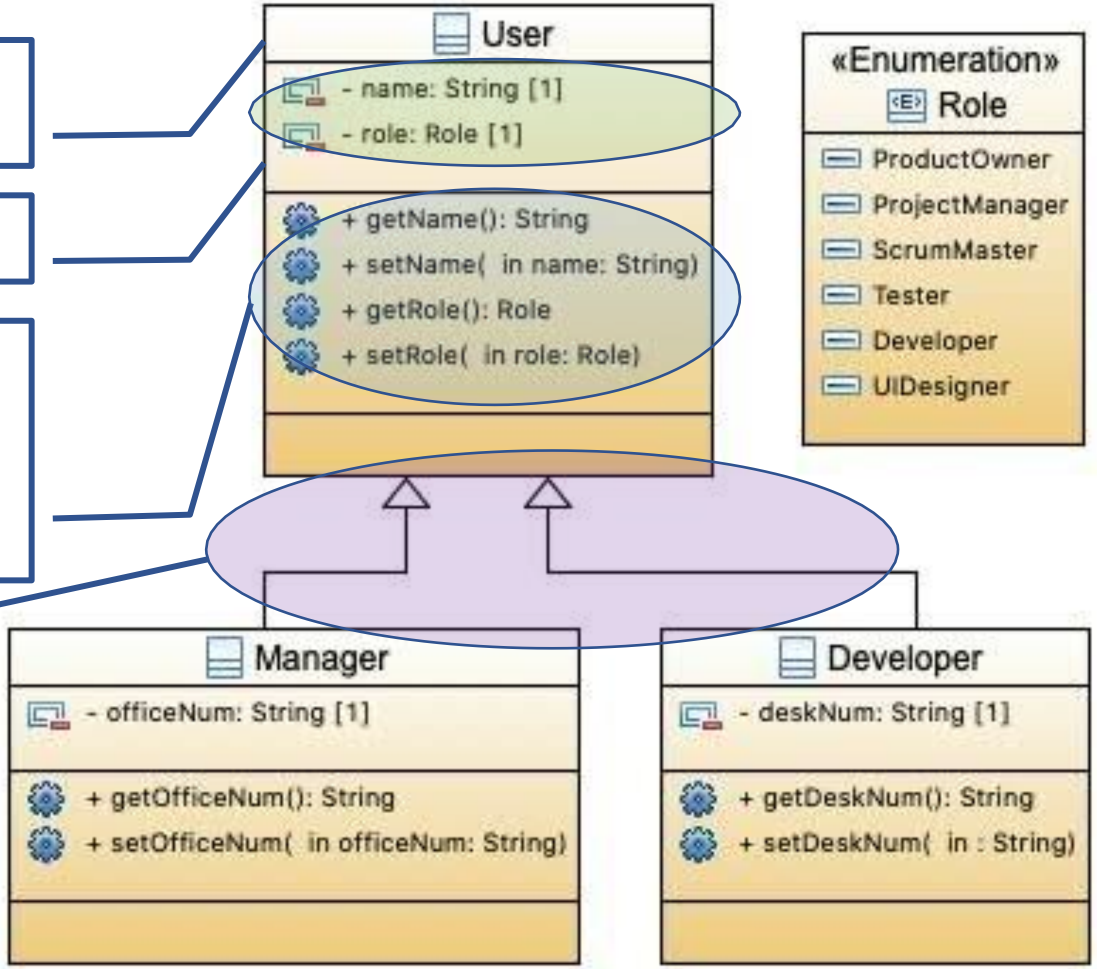
Generalization

A ***class*** may contain several ***attributes*** (properties) and ***operations*** (methods)

An attribute can be ***private***, ***protected***, ***public*** or ***default***

A class can have zero or more ***operations*** (also known as methods). In order to allow other class to access *private* attributes, a class often contain '***getters'*** and '***setters'*** operations for manipulating them.

Generalization represents an "***is-a***" relationship. In this example, it can be read as a Manager is a type of User; a Developer is a type of User. The children class (Manager and Developer) automatically inherits the properties of their parent class. For example, A ***Manager*** also has ***name*** and ***role***.

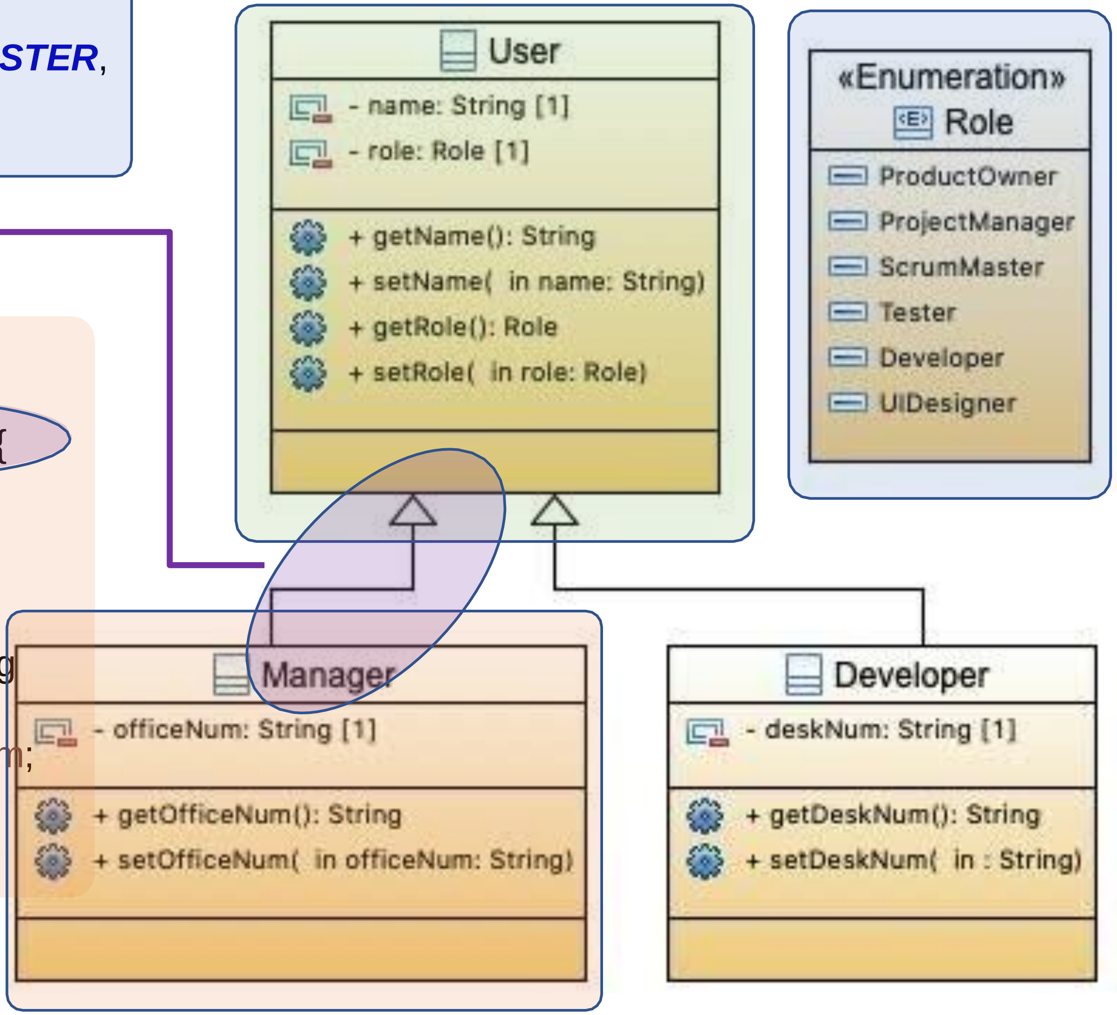


Generalization -- UML to Code

```
package edu.dapeng.usicms.model;
public enum Role {
   DEVELOPER, PRODUCT_OWNER, PROJECT_MANAGER,   SCRUM_MASTER,
TESTER, UIDESIGNER;
}
```

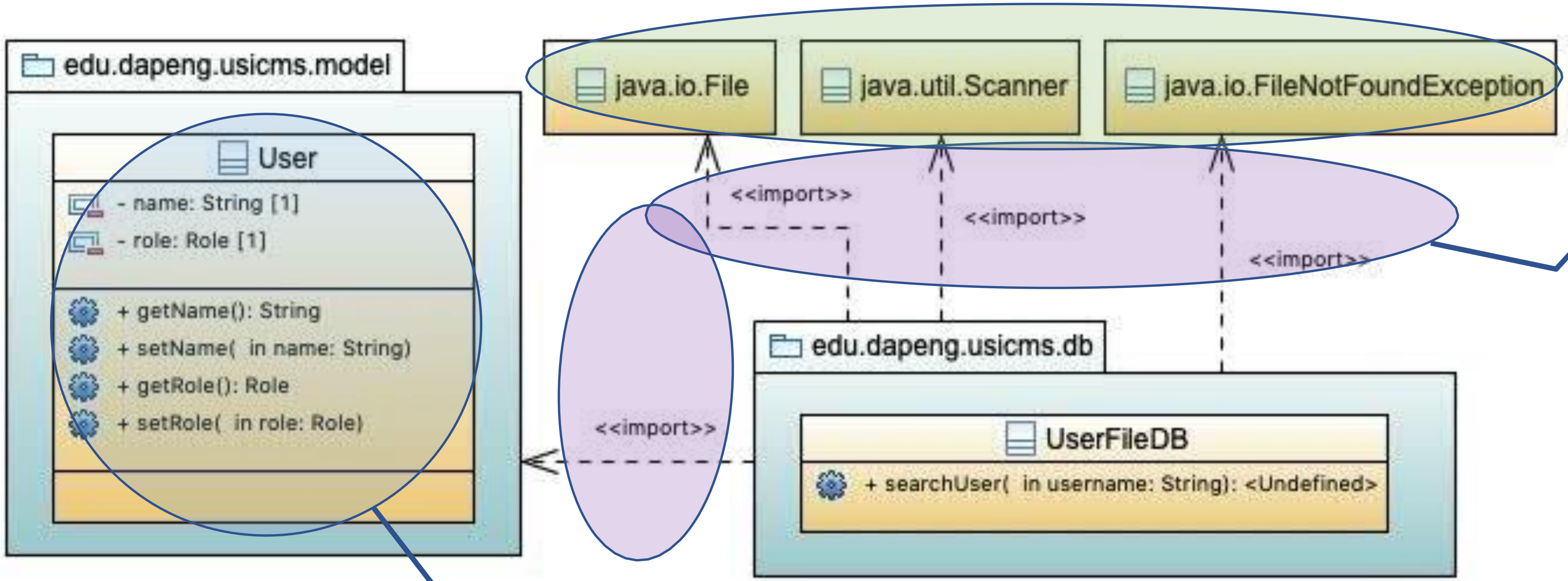
```
package edu.dapeng.usicms.model;
public class User {
        private String name;
        private Role role;
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public Role getRole() {
                return role;
        }
        public void setRole(Role role) {
                this.role = role;
        }
}
```

```
package edu.dapeng.usicms.model;
public class Manager extends User{
        private String  officeNum;
        public String getOfficeNum() {
                return officeNum;
        }
        public void setOfficeNum(String
        officeNum) {
                this.officeNum = officeNum;
        }
}
```



Dependency

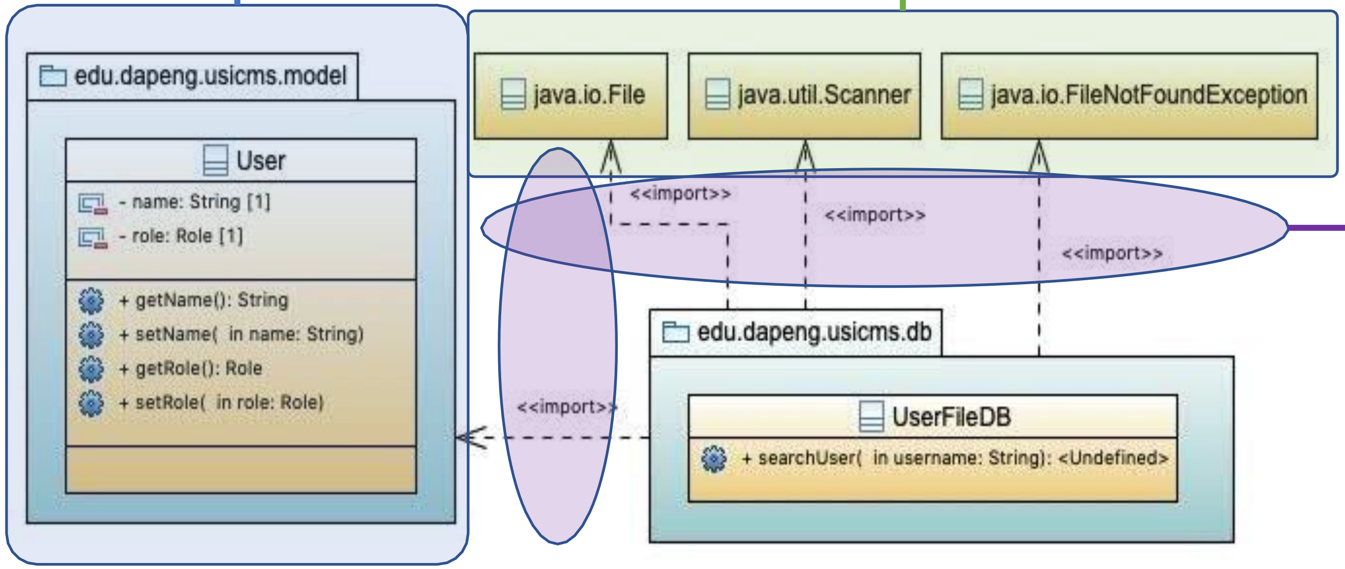
This can be <<import>> of standard Java libraries



Dependency represents the relationship between entities in which if the changes to the definition of one entity may cause changes to the other entities. This can be *<<use>>*, *<<import>>*, *<<depend>>*, *<<refine>>*, *<<extend>>*, *<<include>>*, *<<access>>*, *<<instanceOf>>*, *<<bind>>*, *<<instantiate>>*, etc.

This can be an <<import>> of a class from a different package of the same program.

Dependency – UML to Code



```
package edu.dapeng.usicms.db;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

import edu.dapeng.usicms.model.Role;
import edu.dapeng.usicms.model.User;

public class UserFileDB {
        public User searchUser(String username) {
                User user = new User();
                try {
                        File userFile = new File("User.usesp");
                        Scanner fileReader = new Scanner(userFile);
                        while (fileReader.hasNextLine()) {
                                String data = fileReader.nextLine();
                                ….
                                fileReader.close();
                                return user;
                        }
                } catch (FileNotFoundException e) {
                        …
                }
        }
}
```

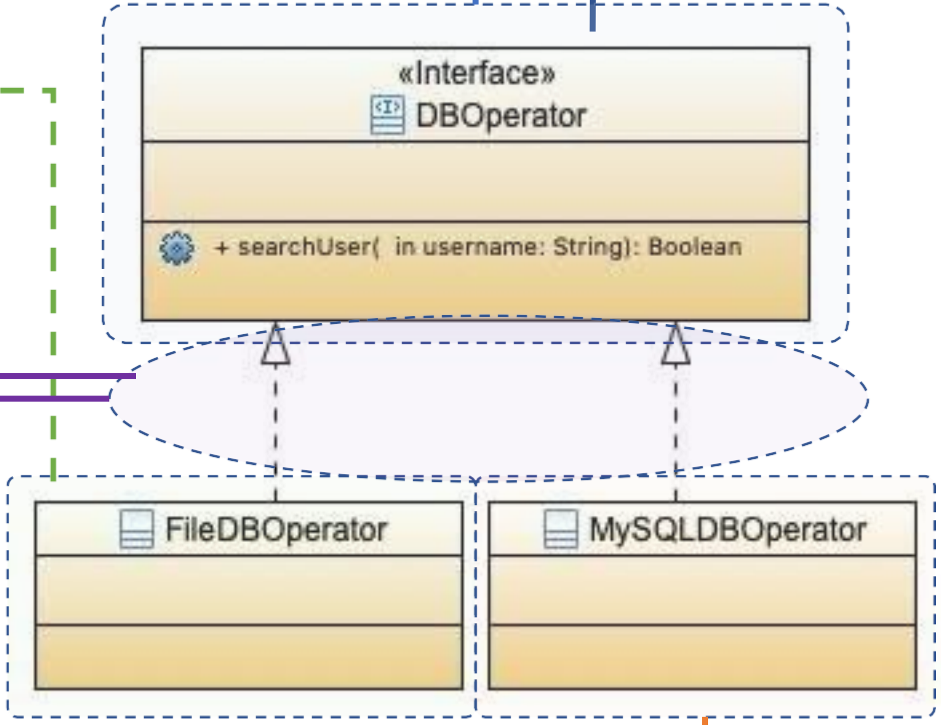
Realization

In a realization relationship, an <<interface>> stereotype is used to create an interface and to realize the particular interface.

```
package edu.dapeng.usicms.db;
public interface DBOperator {
        public boolean searchUser(String username);
}
```

```
package edu.dapeng.usicms.db;
……
public class FileDBOperator implements DBOperator{
        @Override
        public boolean searchUser(String username) {
                try {
                        File userFile = new File("User.usesp");
                        Scanner fileReader = new Scanner(userFile);
                        ….
        }
}
```

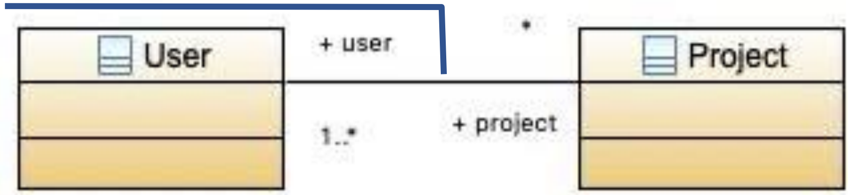
```
package edu.dapeng.usicms.db;
……
public class MySQLDBOperator implements DBOperator {
        @Override
        public boolean searchUser(String username) {
                try {
                        Class.forName("com.mysql.jdbc.Driver");
                        Connection con =
                        DriverManager.getConnection("jdbc:mysql://localhost:3306/usicms",
                        "dapeng", "123456");
                        …..
        }
}
```



Association, Aggregation and Composition

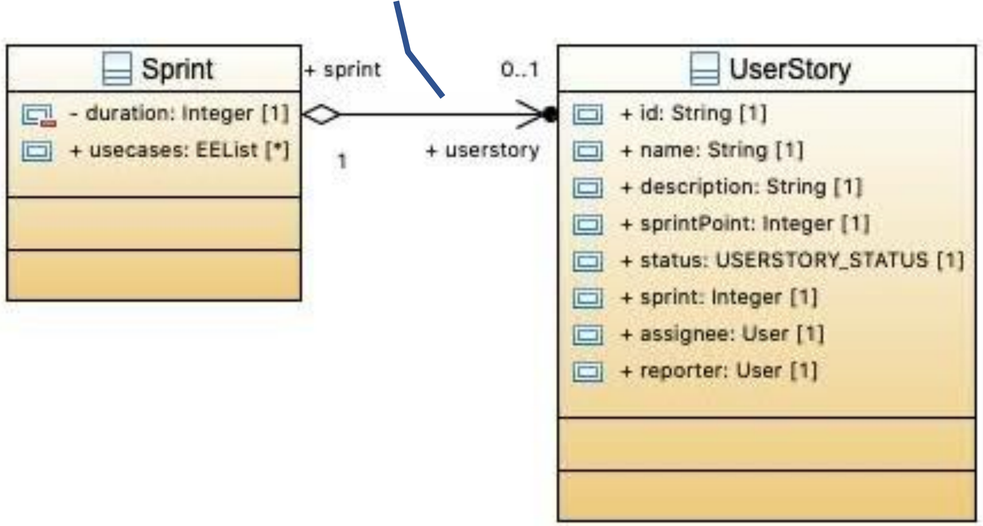
$$\boldsymbol{Composition} \subset \boldsymbol{Aggregation} \subset \boldsymbol{Association}$$

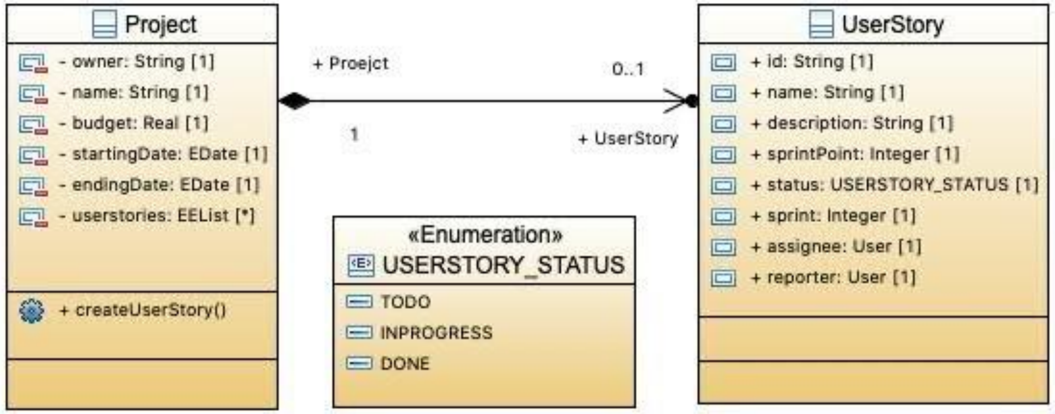
An ***Association*** relationship represents connections or associations between object in the system.



A ***Composition*** is a subtype of an ***aggregation*** relationship. It represents a '***whole/part***' relationship. If a composite is removed, all its associated parts will also be removed with it.

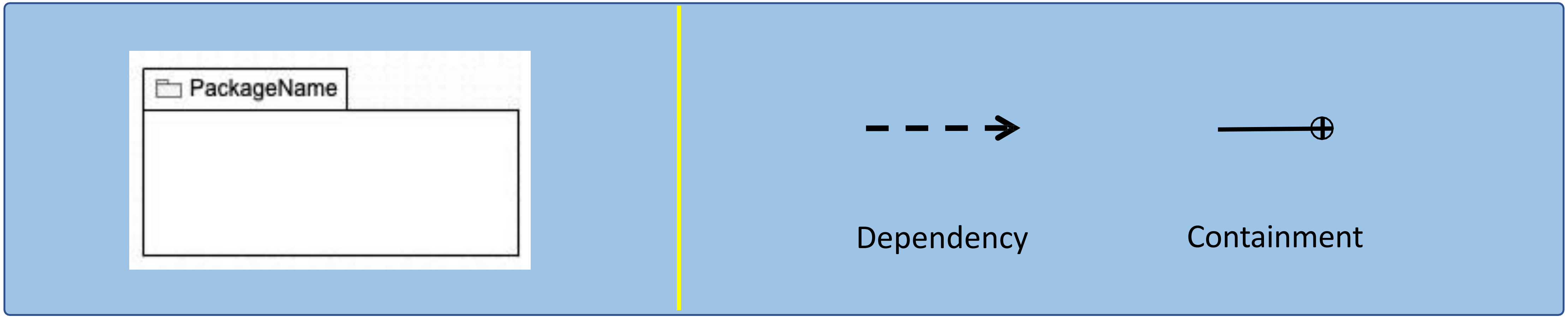
An ***Aggregation*** is a subtype of an ***association*** relationship. It can be described as a '***part of***' relationship. In Aggregation relationship, objects have separate lifetimes.





Package Diagrams

- A Package is considered as a namespace for its members.
- When performing analysis, package diagrams are used to organize the artifacts of the development
  - Provides encapsulation and containment and supports modularity
  - Provides clarity and neat organization in a complex systems development
  - Support version control

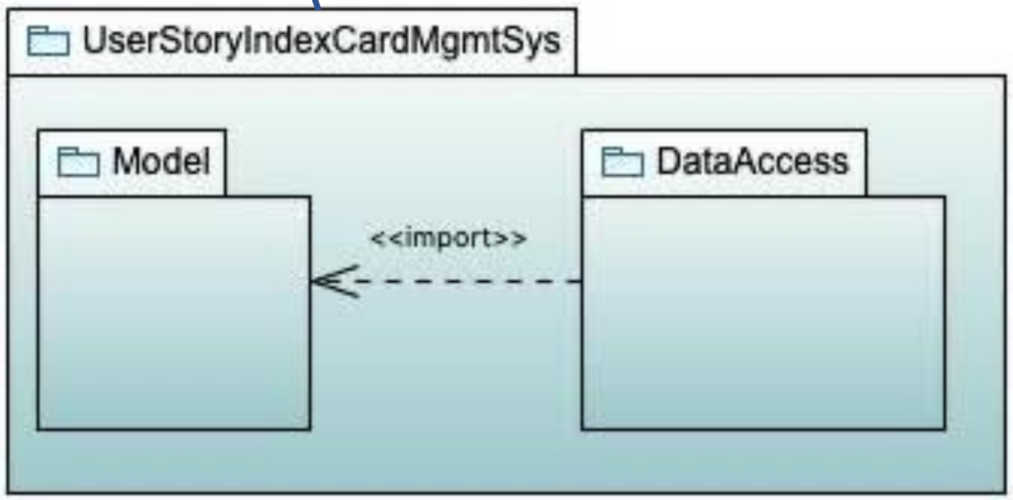


Dependency

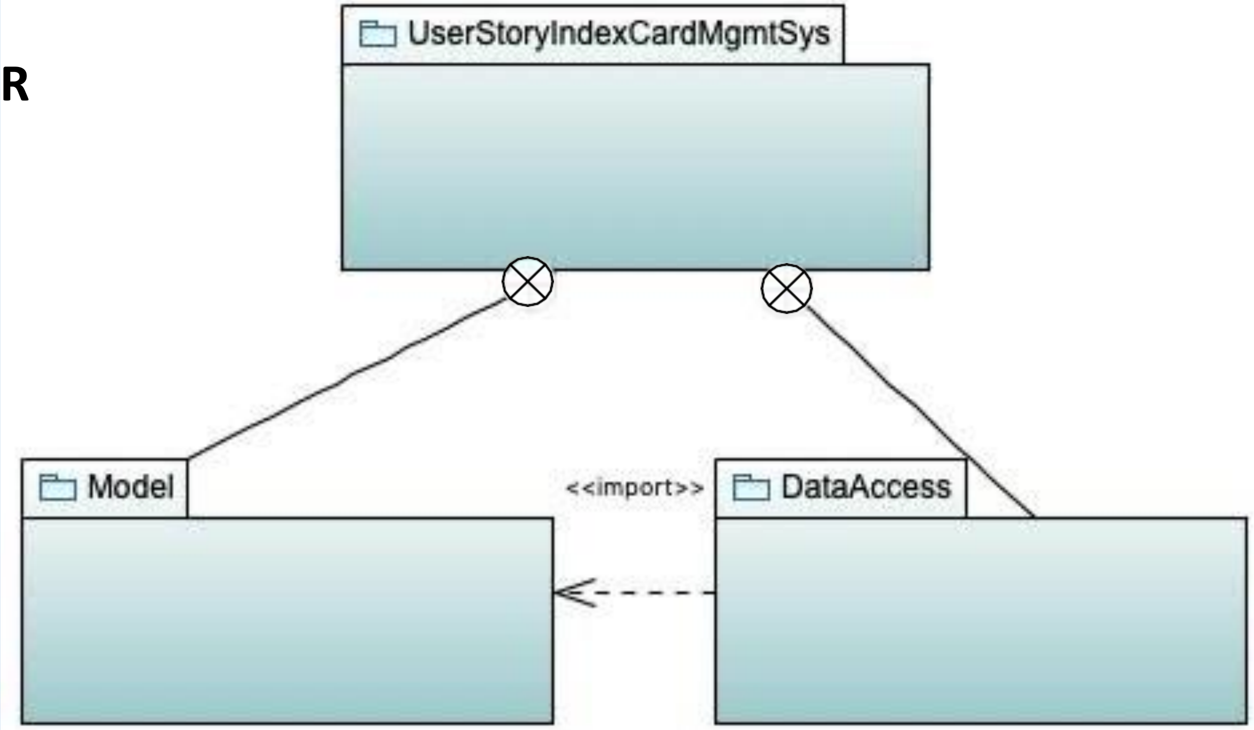
Containment

Package Diagram Examples

A ***Package*** is a namespace for its members, which comprise those elements that are owned or contained and those imported.

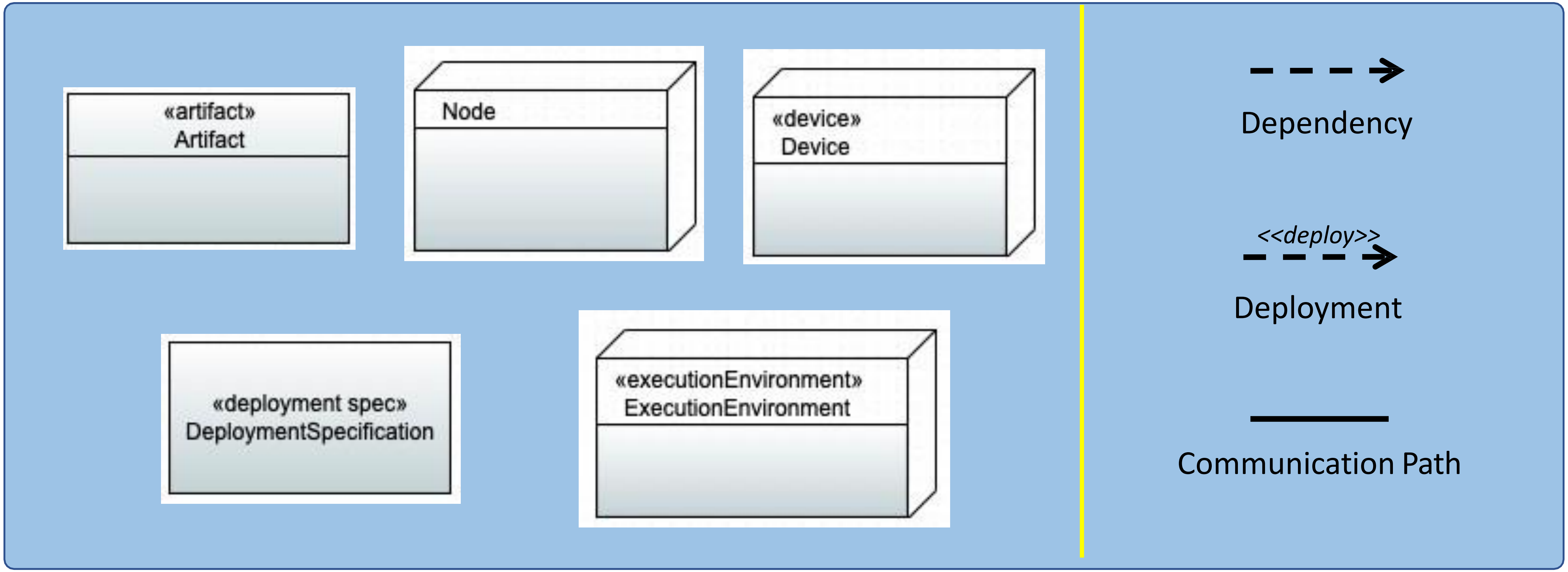


OR

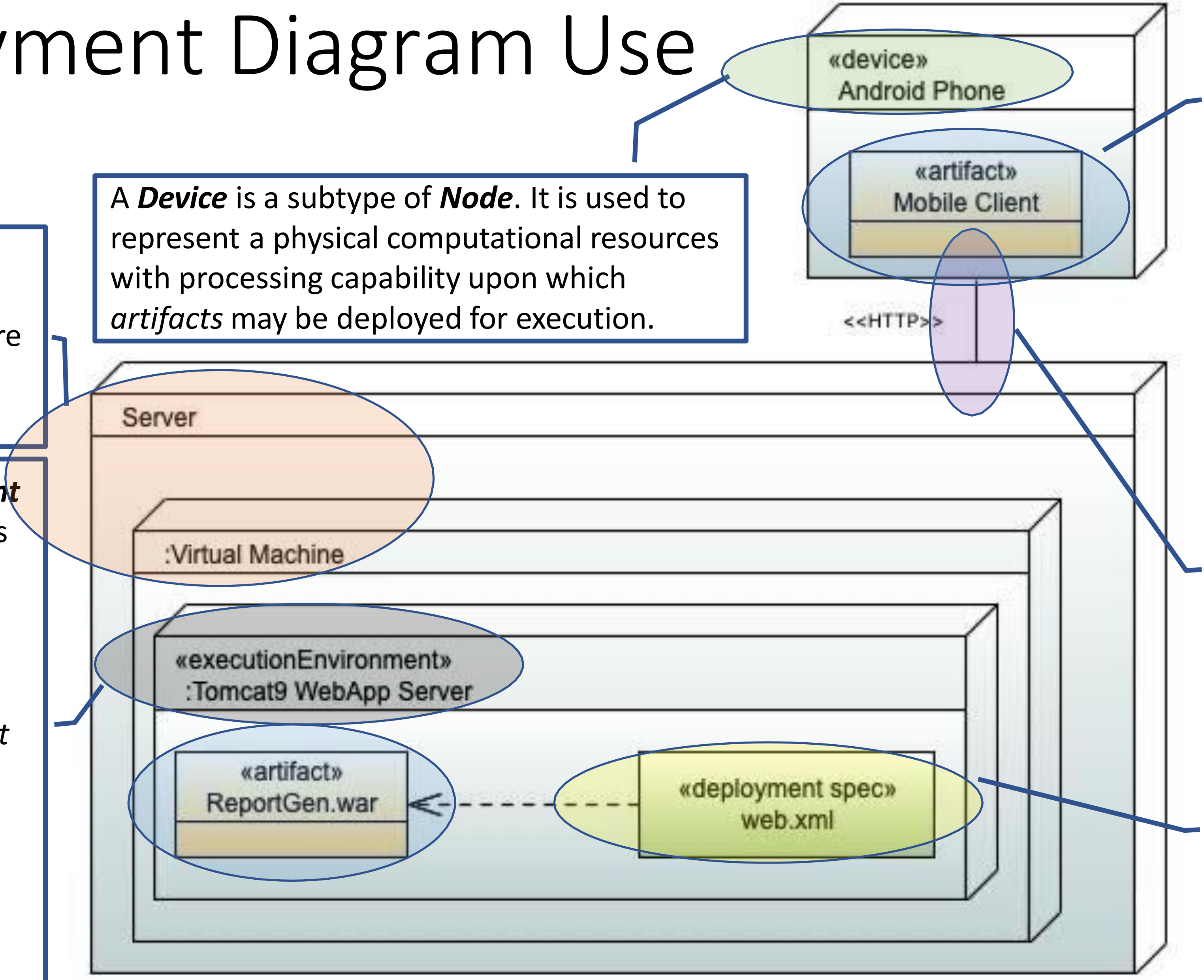


Deployment Diagrams

- Deployment diagram show the relationships between logical and/or physical elements of systems and assets assigned to them.



Deployment Diagram Use Case



A ***Node*** is a generic element that can represent either hardware devices or software execution environment.

An ***ExecutionEnvironment*** is a subtype of ***Node***. It is used to represent some environment (mostly software) for supporting the execution of *artifact*. An *ExecutionEnvironment* is often assigned to a *device* or a *node*. For example, an Application Server, an Operating System, or a database, etc.

A ***Device*** is a subtype of ***Node***. It is used to represent a physical computational resources with processing capability upon which *artifacts* may be deployed for execution.

An ***Artifact*** represents some concrete elements in the physical world that is used or produced by a software development process or by operation of a system. E.g., executable files, source files, database tables, a document, or messages, etc.

A ***CommunicationPath*** is a type of Association between two deployment targets, through which they can exchange information.

A ***Deployment Specification*** specifies a set of properties of an artifact deployed on a node.

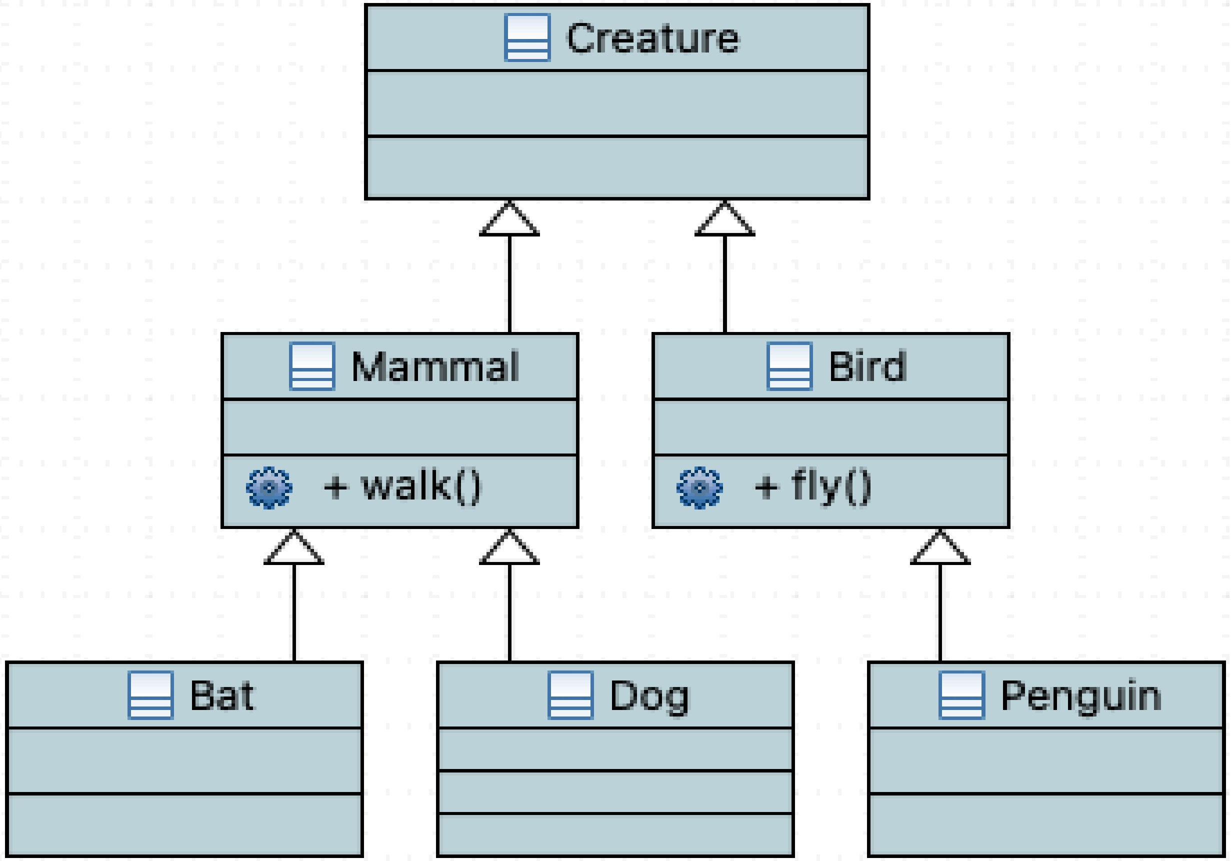
HTTP: Hypertext Transfer Protocol

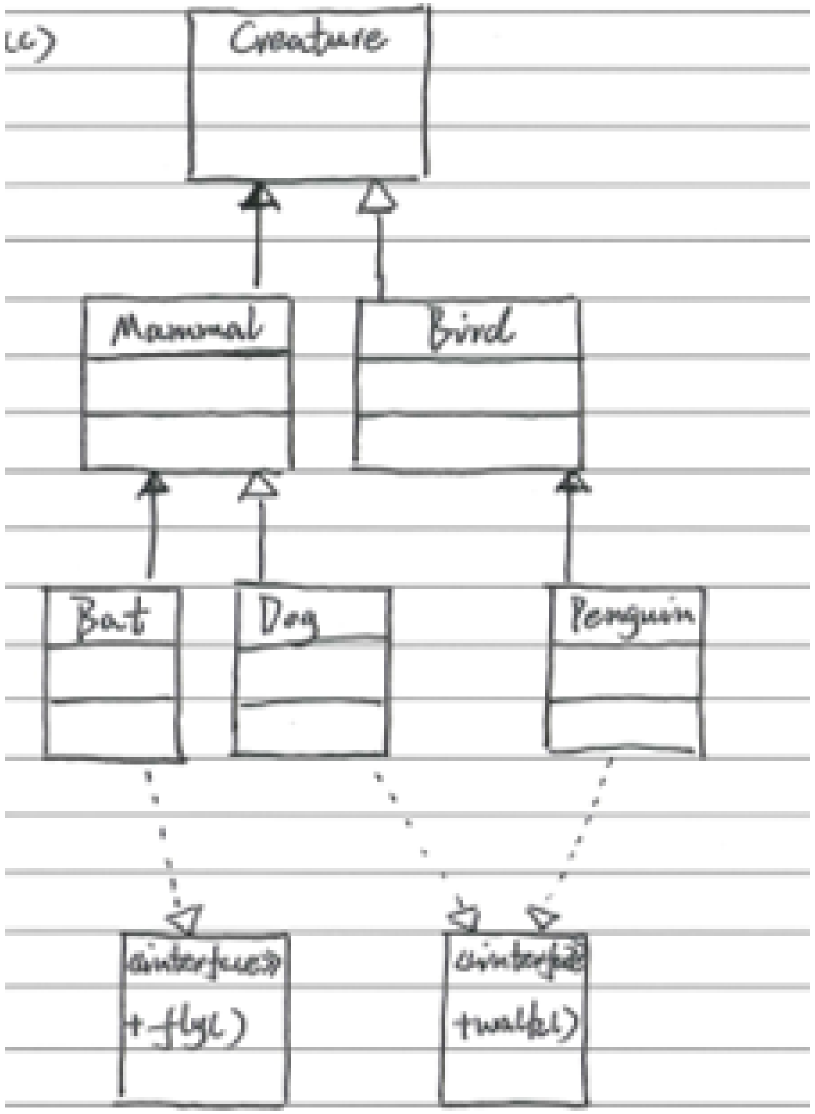
Summary

- A model is an abstract view of a system that deliberately ignores some system details
- A context model shows how a system is positioned in the operational environment. It helps define the boundaries of the system.
- Use case diagrams are used to describe the interactions between external actors and the system to be developed
- Sequence diagrams are used to show the interactions between system objects
- Class diagrams are used to defined static structure of classes of a system and their relationships
- Package diagrams are used to organize the artifacts of the development process
- Deployment diagrams are used to show the allocation of components to physical nodes.

Practice:

1. In Object-Oriented design, modelling complex relationships between entities is hard. The following class diagram models creatures using inheritance. It is technically correct, but logically incorrect, as a bat is a mammal but can fly, and a penguin is a bird, but cannot fly. Redesign the class diagram using class composition so that it is logically and technically correct.




(c)
Creature
Mammal
Bird
Bat
Dog
Penguin
«interface»
+fly()
«interface»
+walk()

2. According to the following code fragments, draw a class diagram:

```
          •   package cn.fzu.miec.doc;
          •   public class Report {
              •   private String title;
•
              •   public Report(String title) {
                  •   this.title = title;
              •   }
•
              •   public String getTitle() {
                  •   return title;
              •   }
          •   }
•
•
          •   package cn.fzu.miec.device;import cn.fzu.miec.doc; public class Printer {
                  •   public void print(Report report) { System.out.println(report.getTitle());
              •   }
          •   }
•
•
          •   package cn.fzu.miec.device;
          •   import cn.fzu.miec.doc;
          •   public class HPPrinter extends Printer {
•
•     }
```

- 3. *Develop a sequence diagram showing the interactions involved when a student registers for a course at a university. Courses may have limited enrolment, so the registration process must include checks that places are available. Assume the student accesses an electronic course catalogue to learn about available courses.