M.Sc. in Computer Science
Department of Computer Science
Faculty of Applied Sciences
University of Sri Jayewardenepura

CSC 540 2.0 Software Engineering

Presented By:
Surani Tissera(PhD)
Department of Computer Science

System Modeling

Topics to be covered

- System Modeling
  - Interaction models
    - Use case diagram
    - Sequence diagram
  - Structural models
    - Class Diagram
  - Behavioral models
    - Activity Diagram
    - State Diagram

Recommended Reading

- Sommerville I. (2015), Software Engineering 10th Edition, Addison-Wesley

- Roger S. Pressman (2010), Software Engineering: A Practitioner’s Approach 7th Edition, McGraw-Hill

Objectives

You will:

- understand **how graphical models can be used to represent software systems** and **why several types of model are needed to fully represent a system**;
- understand the **fundamental system modeling perspectives of context, interaction, structure, and behavior**;
- understand the principle diagram types in the **Unified Modeling Language (UML)** and **how these diagrams may be used in system modeling**;

How the customer explained it

How the project leader understood it

How the analyst designed it

How the programmer wrote it

What the customer really needed

System modeling

- System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system.
- System modeling means **representing a system using some kind of graphical notation**, which is now almost always based on notations in the Unified Modeling Language (UML).
- System modelling helps the analyst to **understand the functionality of the system** and **models are used to communicate with customers**.

Existing and planned system models

- Models of the existing system are used during requirements engineering. They help clarify what the existing system does and can be used as a basis for discussing its strengths and weaknesses. These then lead to requirements for the new system.
- Models of the new system are used during requirements engineering to help explain the proposed requirements to other system stakeholders. Engineers use these models to discuss design proposals and to document the system for implementation.

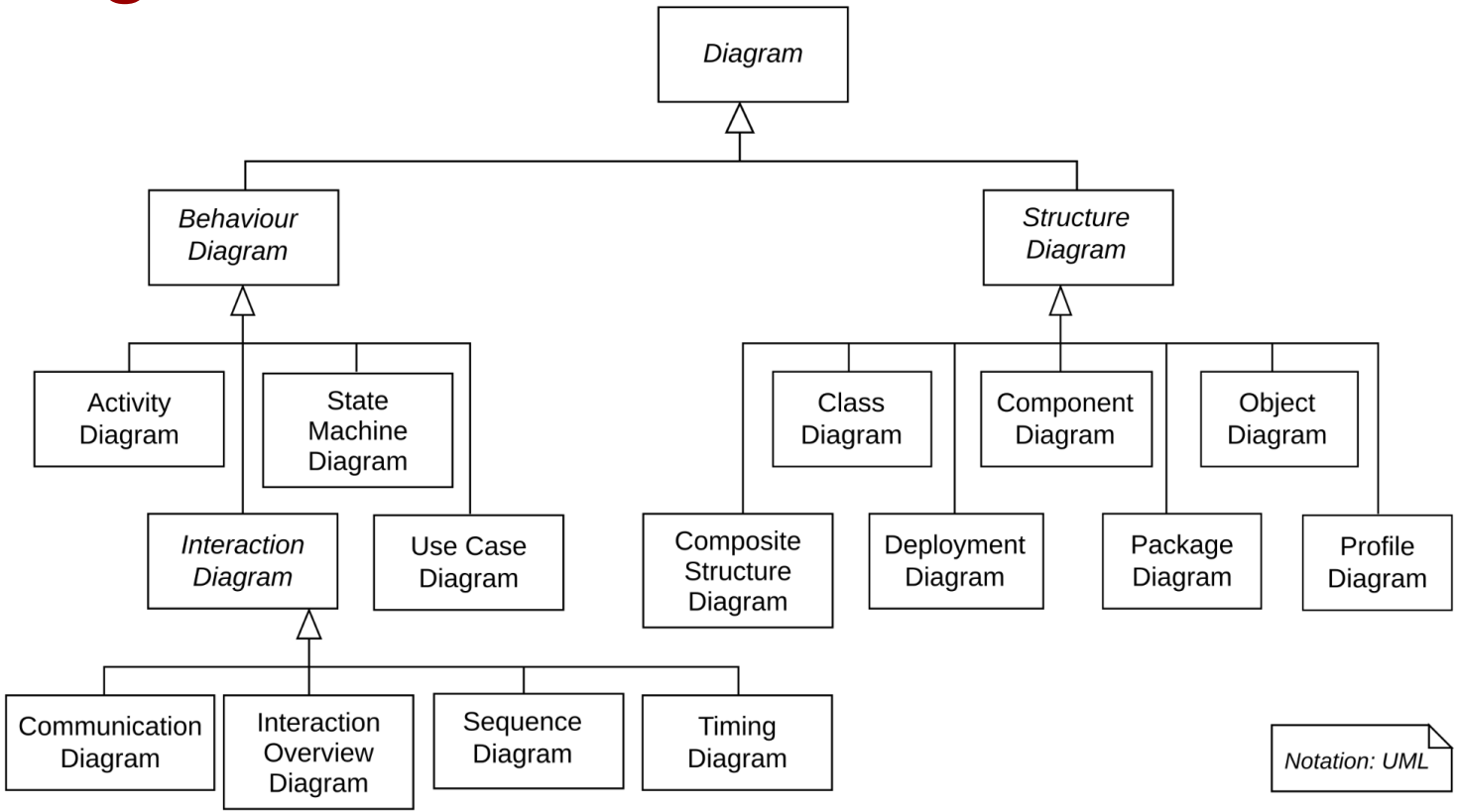
System perspectives

- An external perspective, where you model the context or environment of the system.
- An **interaction perspective**, where you model the interactions between a system and its environment, or between the components of a system.
- A **structural perspective**, where you model the organization of a system or the structure of the data that is processed by the system.
- A **behavioral perspective**, where you model the dynamic behavior of the system and how it responds to events.

UML diagram

- A UML diagram, which stands for "Unified Modeling Language" diagram.
- It is a visual representation of a software system's architecture, design, and implementation.
- It uses a standardized set of symbols and notations to depict the system's components, relationships, and interactions.
- It allows developers to better understand and document complex software systems before coding begins.

UML diagram



UML diagram types

- **Use case diagrams**, which show the interactions between a system and its environment.
- **Sequence diagram**s, which show interactions between actors and the system and between system components.
- **Class diagrams**, which show the object classes in the system and the associations between these classes.
- **Activity diagrams**, which show the activities involved in a process or in data processing .
- **State diagrams**, which show how the system reacts to internal and external events.

Use of graphical models

- As a means of facilitating discussion about an existing or proposed system
  - Incomplete and incorrect models are OK as their role is to support discussion.
- As a way of documenting an existing system
  - Models should be an accurate representation of the system but need not be complete.
- As a detailed system description that can be used to generate a system implementation
  - Models have to be both correct and complete.

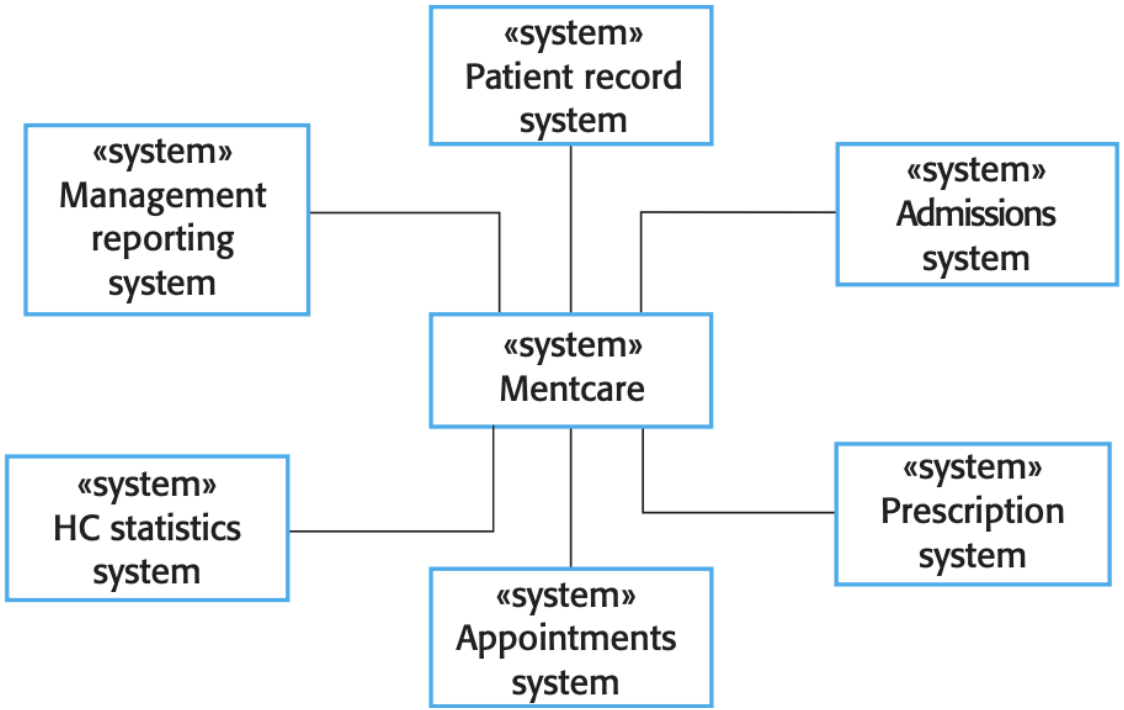
Context models

- Context models are used to illustrate the operational context of a system - they show what lies outside the system boundaries.
- Social and organisational concerns may affect the decision on where to position system boundaries.
- Architectural models show the system and its relationship with other systems.

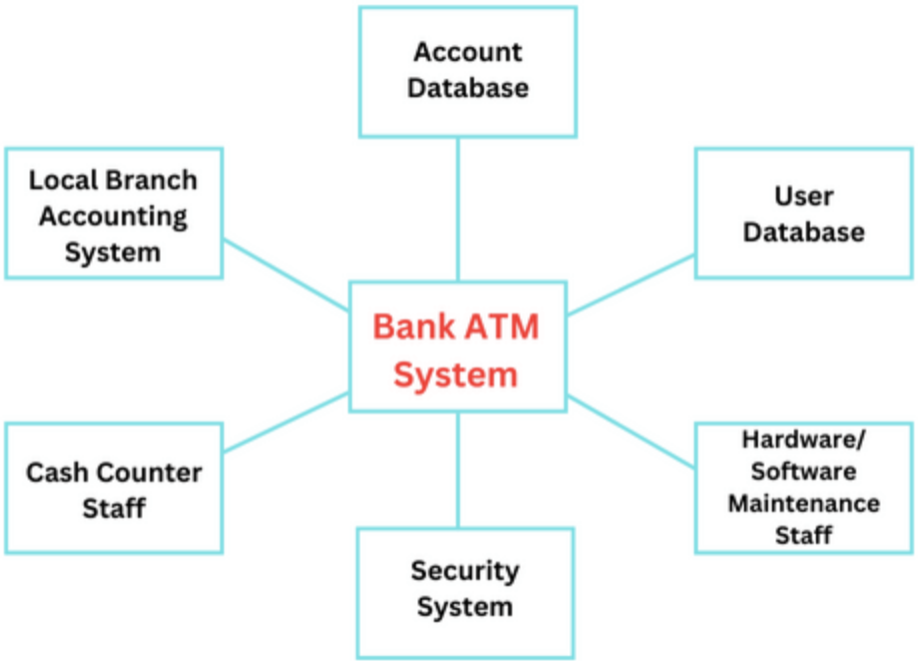
System boundaries

- System boundaries are established to define what is inside and what is outside the system.
  - They show other systems that are used or depend on the system being developed.
- The position of the system boundary has a profound effect on the system requirements.
- Defining a system boundary is a political judgment
  - There may be pressures to develop system boundaries that increase / decrease the influence or workload of different parts of an organization.

The context of the MHC-PMS



Example



Interaction models

- Modeling user interaction is important as it helps to identify user requirements.
- Modeling system-to-system interaction highlights the communication problems that may arise.
- Modeling component interaction helps us understand if a proposed system structure is likely to deliver the required system performance and dependability.

Interaction models

There are two kinds of diagrams that shows interaction models:

1. Use case diagrams
2. sequence diagrams

Use case modeling

- Use cases were developed originally to support requirements elicitation and now incorporated into the UML.
- Each use case represents a discrete task that involves external interaction with a system.
- Represented diagrammatically to provide an overview of the use case.

What is a **use case**?

- A requirements analysis concept
- A **case** of a **use** of the system/product
- Describes the system's actions from a the point of view of a user
- Specifies one aspect of the behavior of a system, without specifying the structure of the system

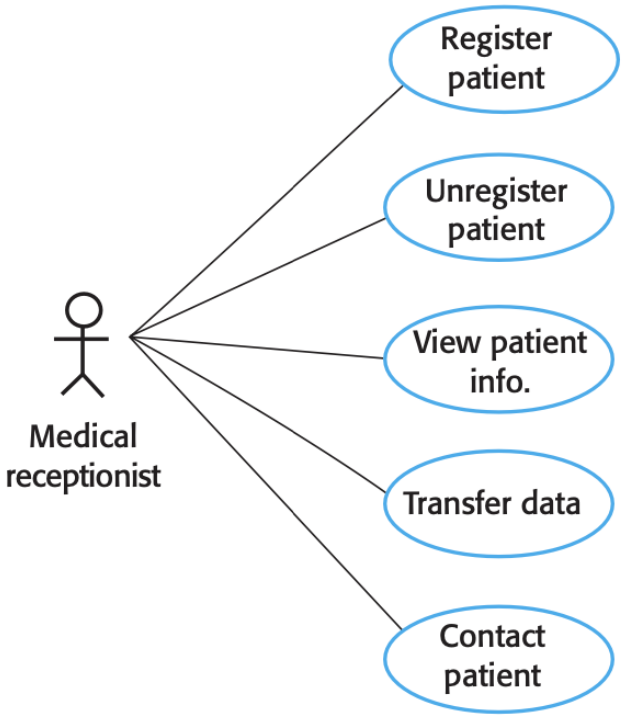
How do we describe use cases?

- Diagrams
- Textual or tabular descriptions
- User stories

Transfer-data use case

A use case in the MHC-PMS

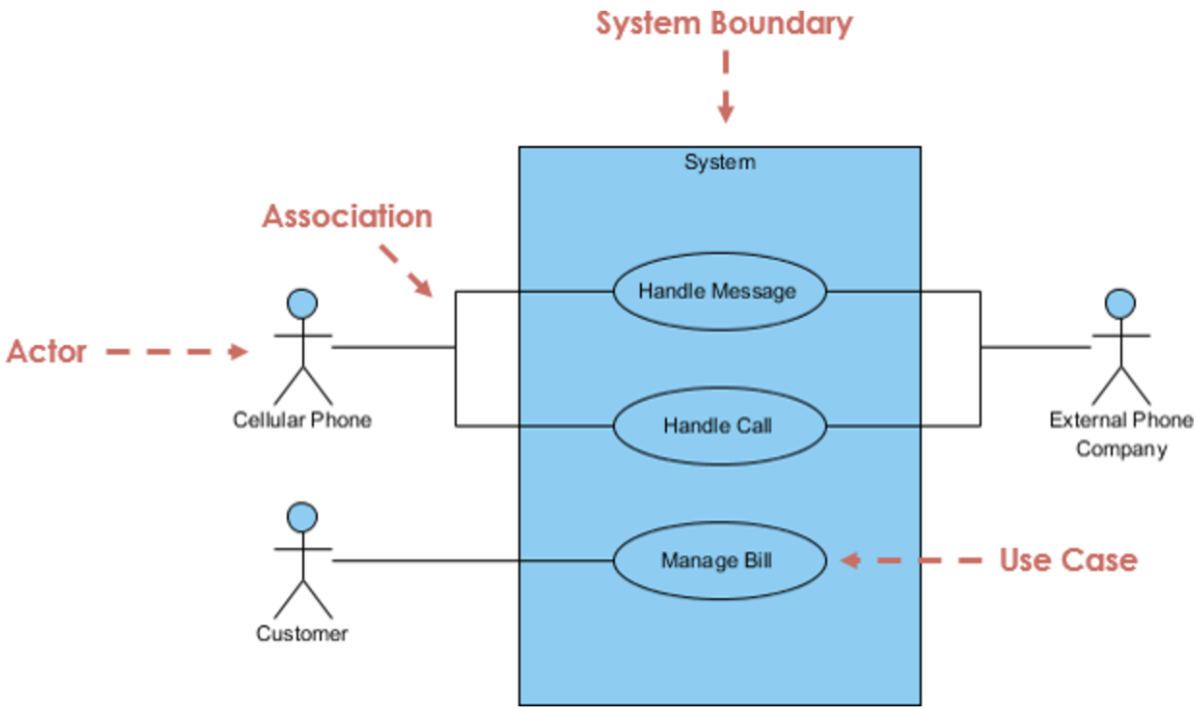
Use cases in the MHC-PMS involving the role ‘Medical Receptionist’



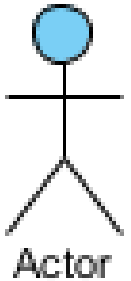
Tabular description of the ‘Transfer data’ use-case

Mentcare system: Transfer data	
Actors	Medical receptionist, Patient records system (PRS)
Description	A receptionist may transfer data from the Mentcare system to a general patient record database that is maintained by a health authority. The information transferred may either be updated personal information (address, phone number, etc.) or a summary of the patient’s diagnosis and treatment.
Data	Patient’s personal information, treatment summary
Stimulus	User command issued by medical receptionist
Response	Confirmation that PRS has been updated
Comments	The receptionist must have appropriate security permissions to access the patient information and the PRS.

Use Case Diagram at a Glance



Use Case Descriptions- Actor



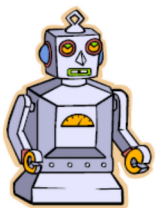
- Someone interacts with use case (system function).
- Named by noun.
- Actor plays a role in the business
- Similar to the concept of user, but a user can play different roles
- For example:
  - A prof. can be teacher and also researcher
  - plays 2 roles with two systems
- Actor triggers use case(s).
- Actor has a responsibility toward the system (inputs), and Actor has expectations from the system (outputs).

Finding Actors?

External objects that produce/consume data:

- Must serve as sources and destinations for data
- Must be external to the system

Humans



Machines

External systems

Organizational Units

Sensors

Finding Actors?

The following questions can help you identify the actors of your system

- Who uses the system?
- Who installs the system?
- Who starts up the system?
- Who maintains the system?
- Who shuts down the system?
- What other systems use this system?
- Who gets information from this system?
- Who provides information to the system?
- Does anything happen automatically at a present time?

Use Case Descriptions- Use case

- System function (process - automated or manual)
- Named by verb + Noun (or Noun Phrase).
  - i.e. Do something
- Each Actor must be linked to a use case, while some use cases may not be linked to actors.

USER/ACTOR	USER GOAL = Use Case
Order clerk	Look up item availability Create new order Update order
Shipping clerk	Record order fulfillment Record back order
Merchandising manager	Create special promotion Produce catalog activity report

How to Identify Use Cases?

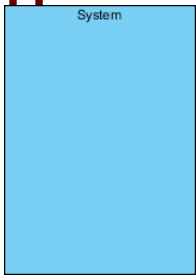
The following questions can be asked to identify use cases,

- What functions will the actor want from the system?
- Does the system store information? What actors will create, read, update or delete this information?
- Does the system need to notify an actor about changes in the internal state?
- Are there any external events the system must know about? What actor informs the system of those events?

Use Case Descriptions- Communication Link

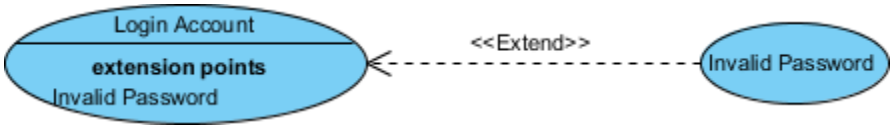
- The participation of an actor in a use case is shown by connecting an actor to a use case by a solid link.
- Actors may be connected to use cases by **associations**, indicating that the actor and the use case communicate with one another using messages.

Use Case Descriptions- Boundary of System



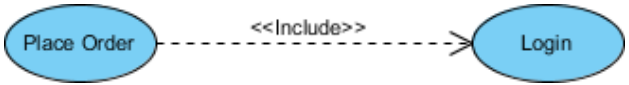
- The system boundary is potentially the entire system as defined in the requirements document.
- For large and complex systems, each module may be the system boundary.
- For example, for an ERP system for an organization, each of the modules such as personnel, payroll, accounting, etc. can form a system boundary for use cases specific to each of these business functions.
- The entire system can span all of these modules depicting the overall system boundary

Use Case Descriptions- Extends



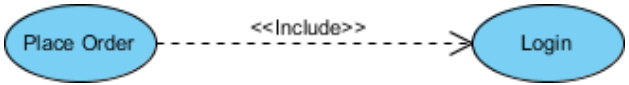
- Indicates that an "Invalid Password" use case may include (subject to specified in the extension) the behavior specified by base use case "Login Account".
- Depict with a directed arrow having a dotted line. The tip of arrowhead points to the base use case and the child use case is connected at the base of the arrow.
- The stereotype "<<extends>>" identifies as an extend relationship

Use Case Descriptions- Include



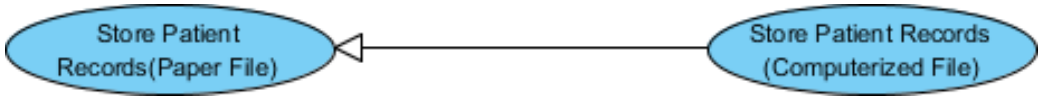
- When a use case is depicted as using the functionality of another use case, the relationship between the use cases is named as include or uses relationship.
- A use case includes the functionality described in another use case as a part of its business process flow.
- A uses relationship from base use case to child use case indicates that an instance of the base use case will include the behavior as specified in the child use case.

Use Case Descriptions- Include



- An include relationship is depicted with a directed arrow having a dotted line. The tip of arrowhead points to the child use case and the parent use case connected at the base of the arrow.
- The stereotype "<<include>>" identifies the relationship as an include relationship.

Use Case Descriptions- Generalization

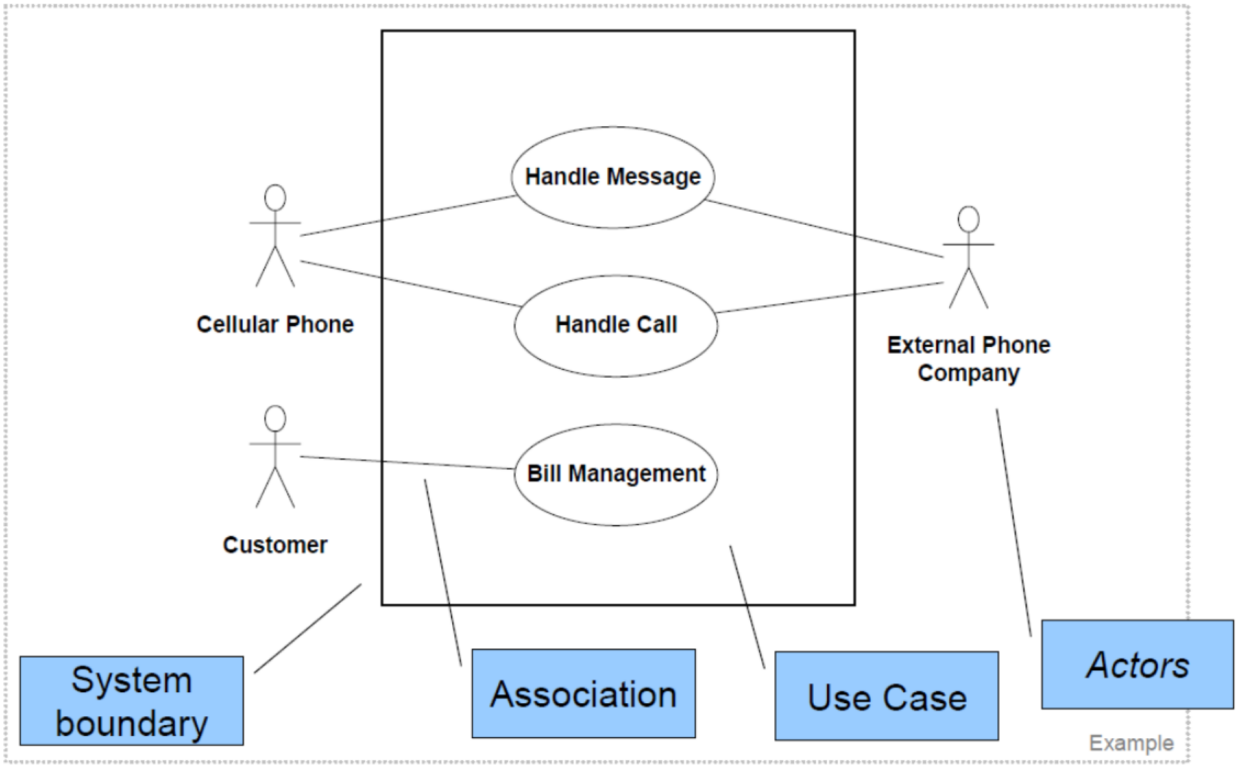


- A generalization relationship is a parent-child relationship between use cases.
- The child use case is an enhancement of the parent use case.
- Generalization is shown as a directed arrow with a triangle arrowhead.
- The child use case is connected at the base of the arrow. The tip of the arrow is connected to the parent use case.

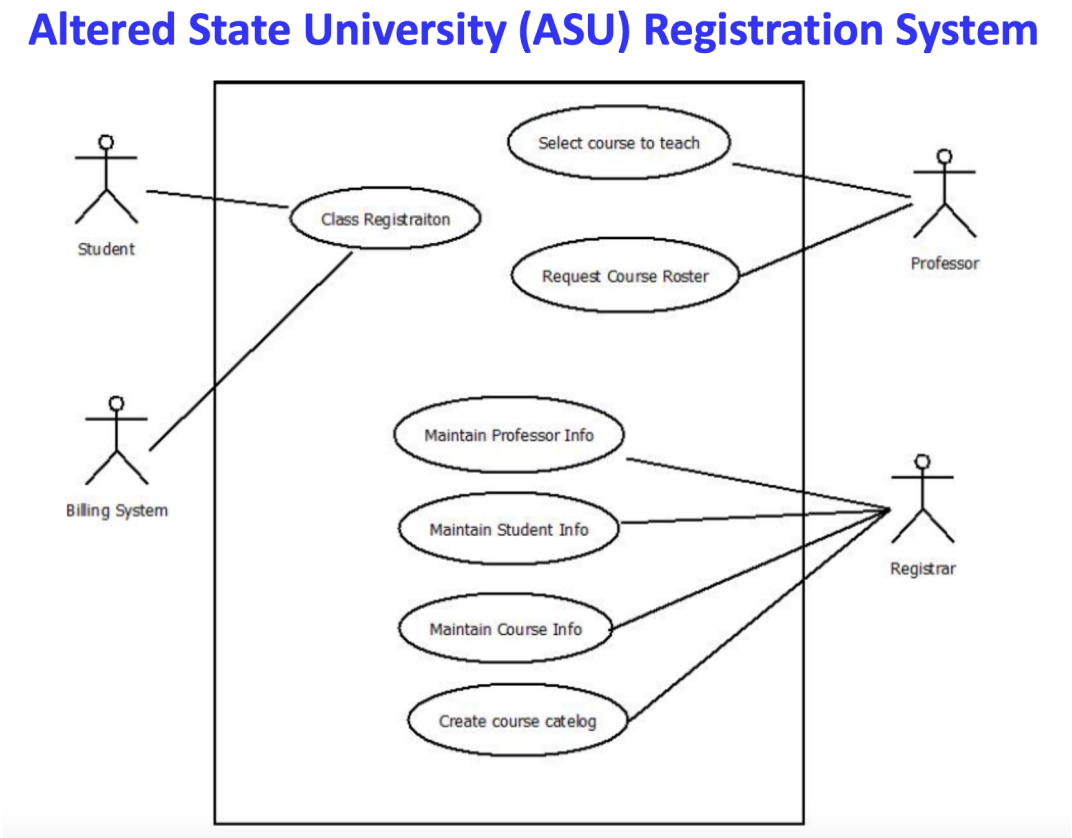
Use Case Diagram Tips

- Always structure and organize the use case diagram from the perspective of actors.
- Use cases should start off simple and at the highest view possible. Only then can they be refined and detailed further.
- Use case diagrams are based upon functionality and thus should focus on the "what" and not the "how".

Example: Use case Diagram



How to create use case diagram



How to create use case diagram

1. List main system functions (use cases) in a column:
   – think of business events demanding system's response
   – users' goals/needs to be accomplished via the system
   – Create, Read, Update, Delete (CRUD) data tasks
   – Naming use cases – user's needs usually can be translated in data tasks
2. Draw ovals around the function labels
3. Draw system boundary
4. Draw actors and connect them with use cases (if more intuitive, this can be done as step 2)
5. Specify include and extend relationships between use cases (yes, at the end - not before, as this may pull you into process thinking, which does not apply in UC diagramming).

Use-Case Diagram Example

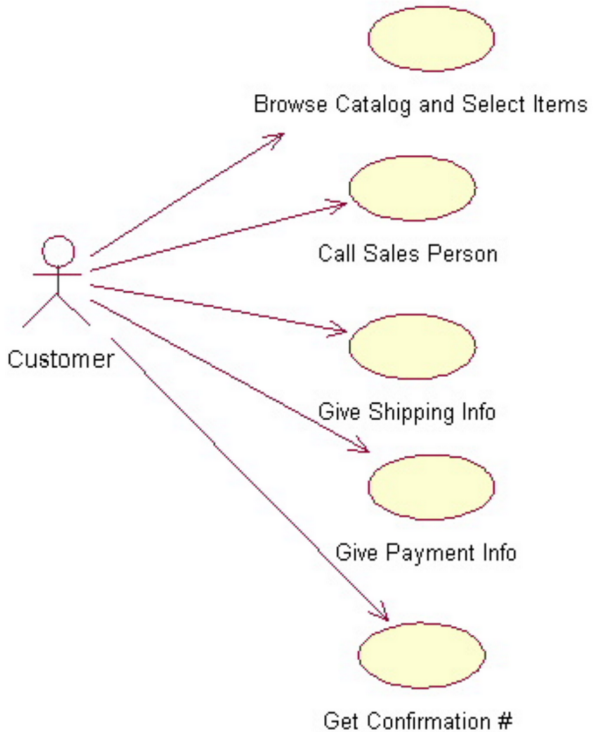
I. Begin with a Use Case

A user placing an order with a sales company might follow these steps :

1. Browse catalog and select items.
2. Call sales representative.
3. Supply shipping information.
4. Supply payment information.
5. Receive confirmation number from salesperson.

II. Then translate Use Case sequence into Diagram

Use-Case Diagram Example



Use-Case Diagram Case Study

Vending Machine

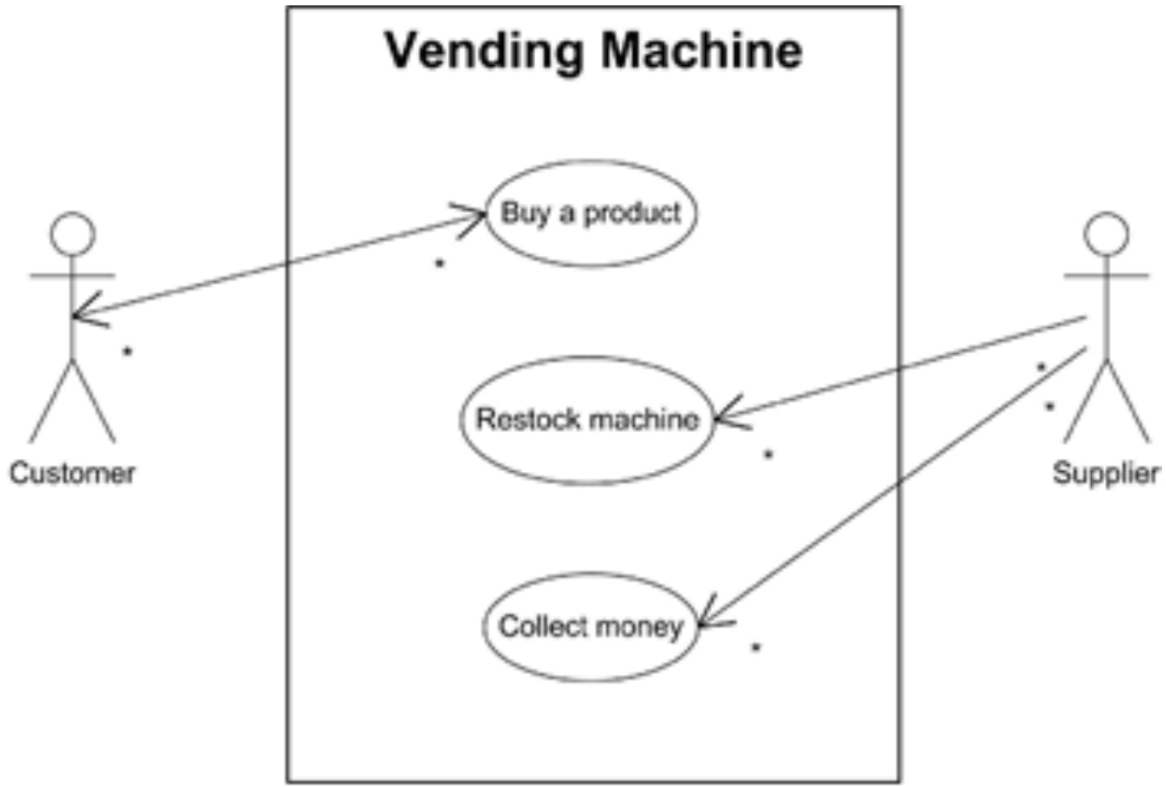
After client interview the following system scenarios were identified:

- A customer buys a product
- The supplier restocks the machine
- The supplier collects money from the machine

On the basis of these scenarios, the following three actors can be identified:

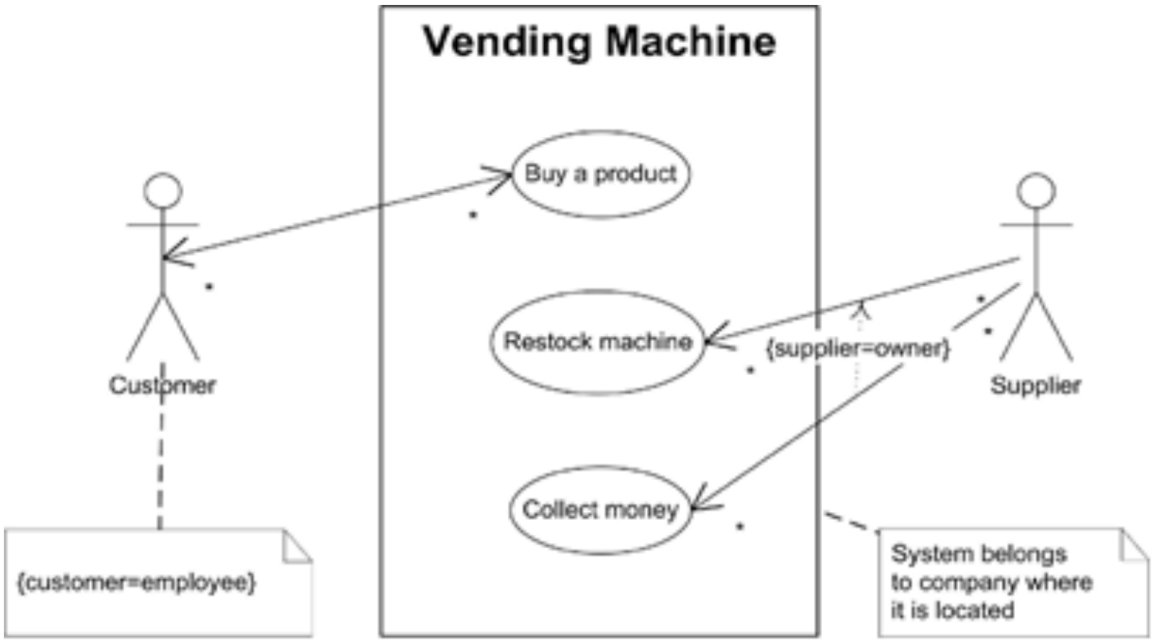
Customer; Supplier; Collector (in this case Collector=Supplier)

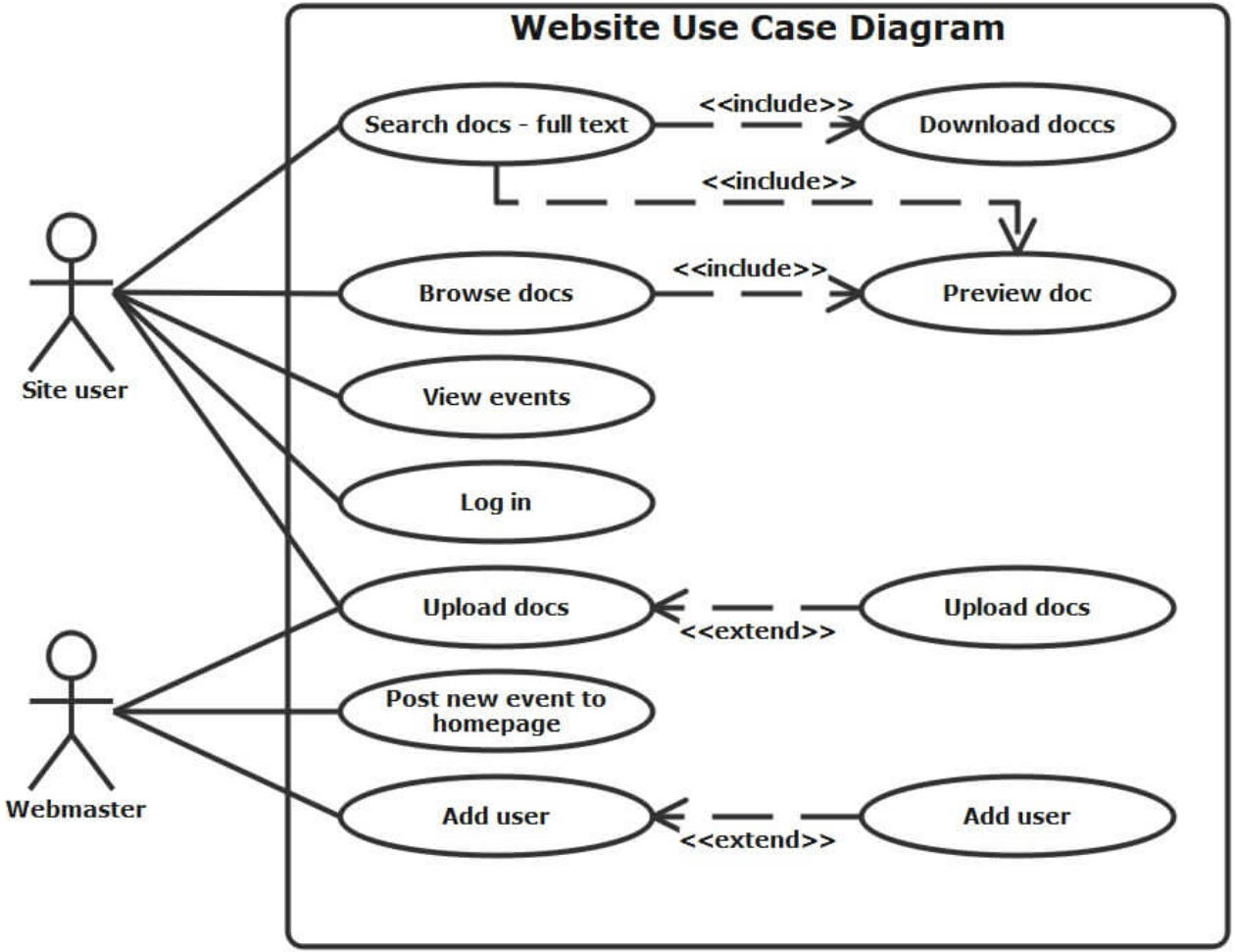
Use-Case Diagram Case Study

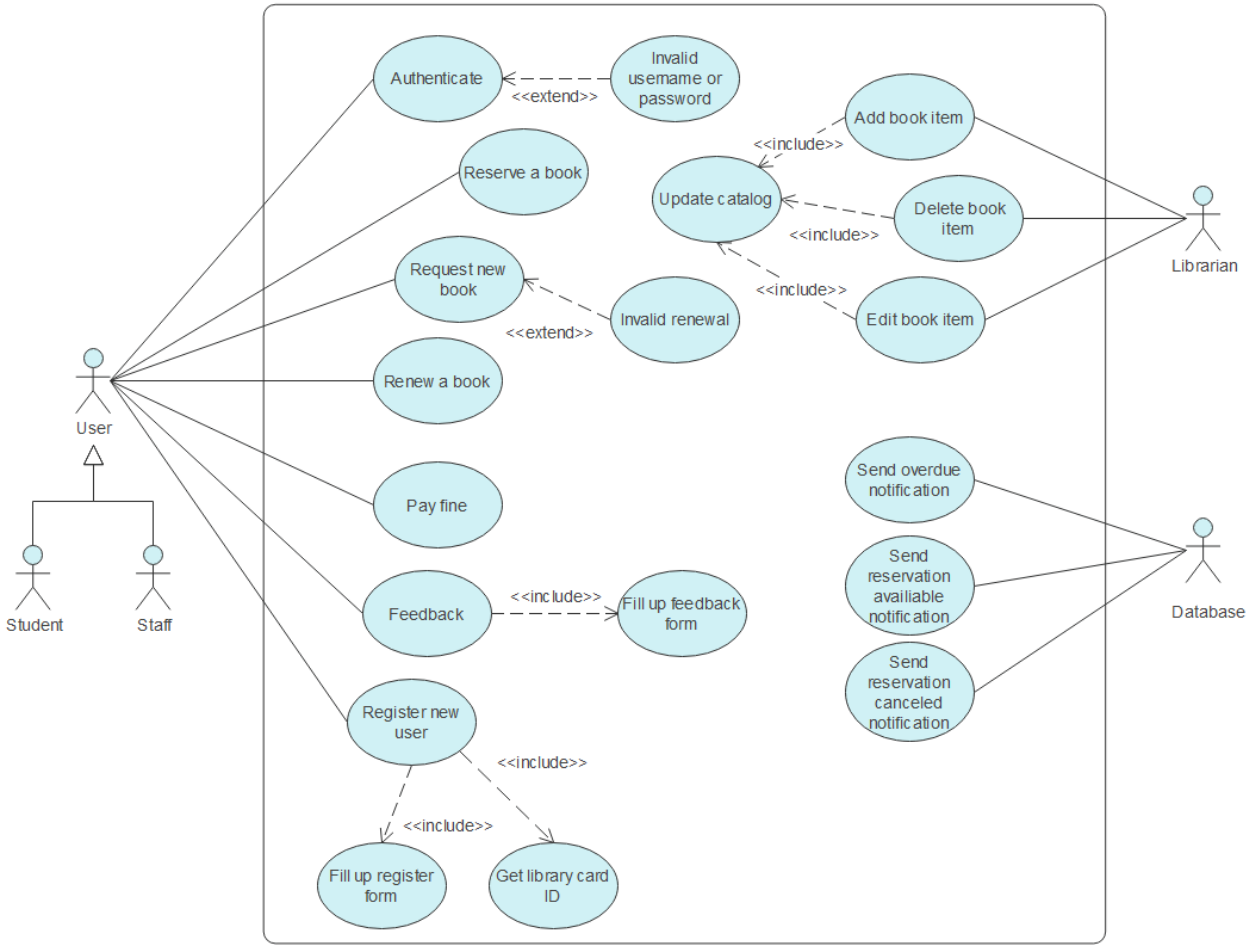


Use-Case Diagram Case Study

Introducing annotations (notes) and constraints.







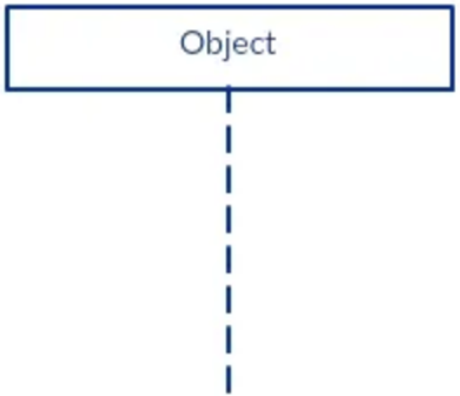
Sequence diagrams

- Sequence diagrams are part of the UML and are used to model the interactions between the actors and the objects within a system.
- A sequence diagram shows the sequence of interactions that take place during a particular use case or use case instance.
- The objects and actors involved are listed along the top of the diagram, with a dotted line drawn vertically from these.
- Interactions between objects are indicated by annotated arrows.

Why use Sequence Diagrams?

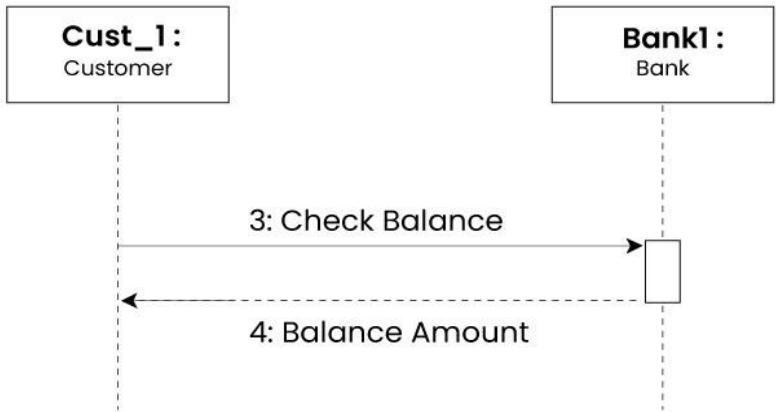
- Visualizing Dynamic Behavior
- Clear Communication
- Use Case Analysis
- Designing System Architecture
- Documenting System Behavior
- Debugging and Troubleshooting

Notation- Lifeline Notation



- A sequence diagram is made up of several of these lifeline notations.
- It should be arranged horizontally across the top of the diagram.
- No two lifeline notations should overlap each other.
- They represent the different objects or parts that interact with each other in the system during the sequence.
- Naming a lifeline: Instance Name : Class Name

Notation- Lifeline Notation



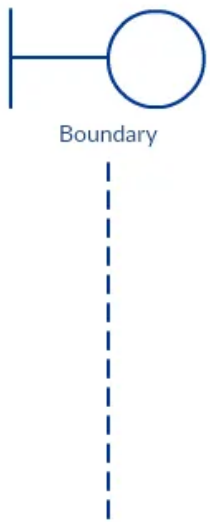
Lifeline Notation

lifeline with an actor element symbol

lifeline with an entity element symbol

Lifeline Notation

lifeline with a boundary element symbol

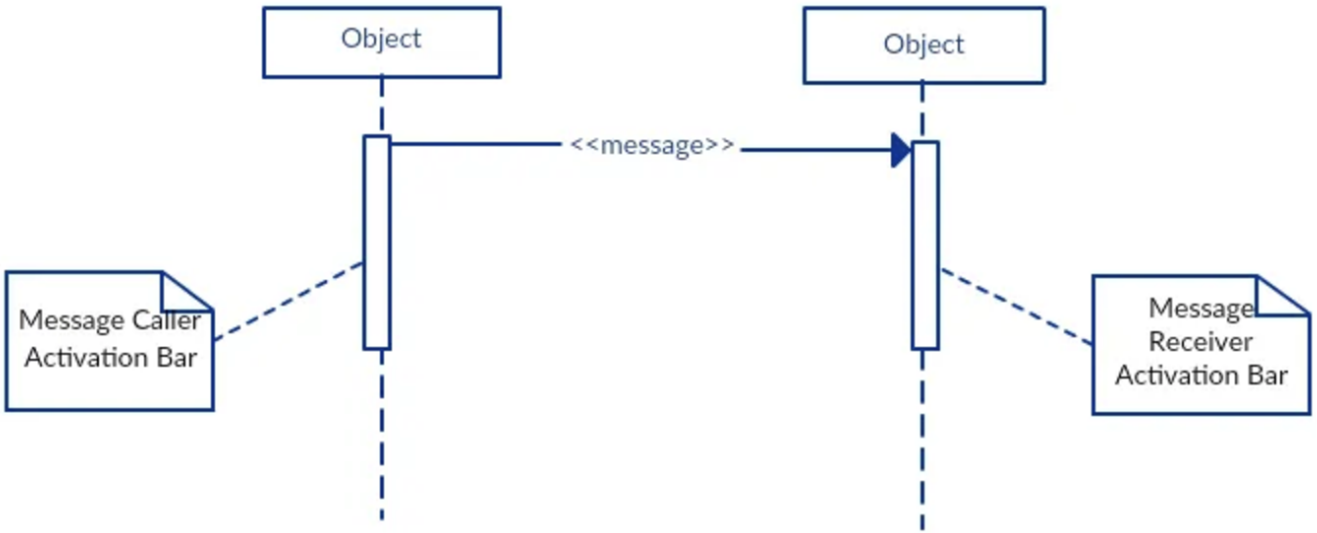


lifeline with a control element symbol

Notation- Activation Bars

- The activation bar is the box placed on the lifeline.
- It is used to indicate that an object is active (or instantiated) during an interaction between two objects.
- The length of the rectangle indicates the duration of the objects staying active.

Notation- Activation Bars



Notation- Message Arrows

An arrow from the Message Caller to the Message Receiver specifies a message in a sequence diagram.

The format for this message signature is :

attribute = message_name (arguments): return_type

All parts except the message_name are optional.

Notation- Message Arrows

1. Synchronous message

A synchronous message is used when the sender waits for the receiver to process the message and return before carrying on with another message.

The arrowhead used to indicate this type of message is a solid one, like the one below.

Notation- Message Arrows

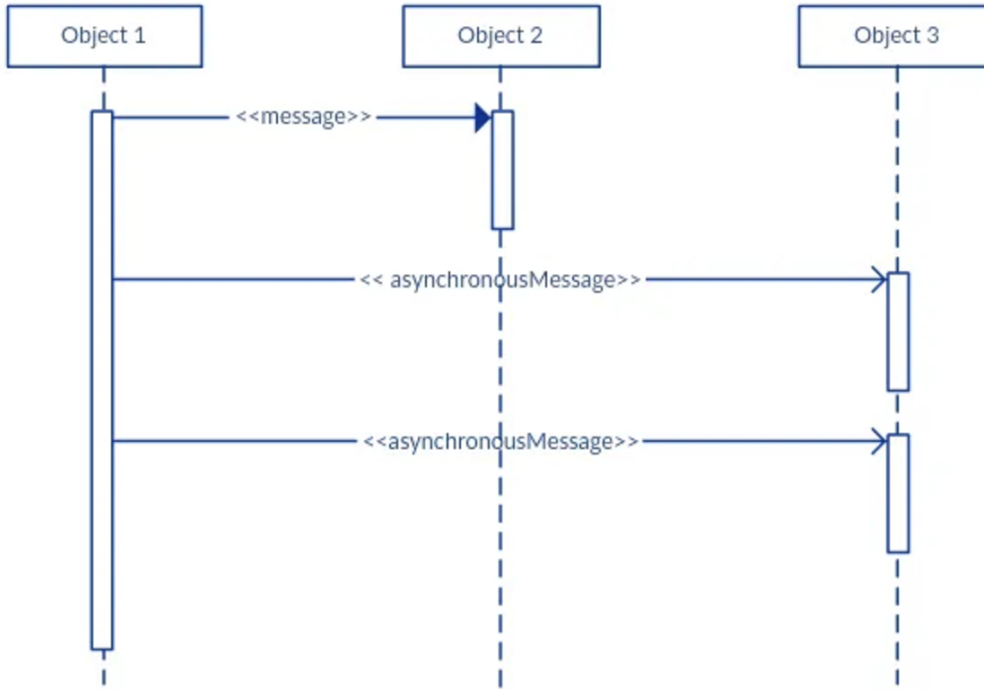
2. Asynchronous message

An asynchronous message is used when the message caller does not wait for the receiver to process the message and return before sending other messages to other objects within the system.

The arrowhead used to show this type of message is a line arrow as shown in the example below.

Notation- Message Arrows

2. Asynchronous message



Notation- Message Arrows

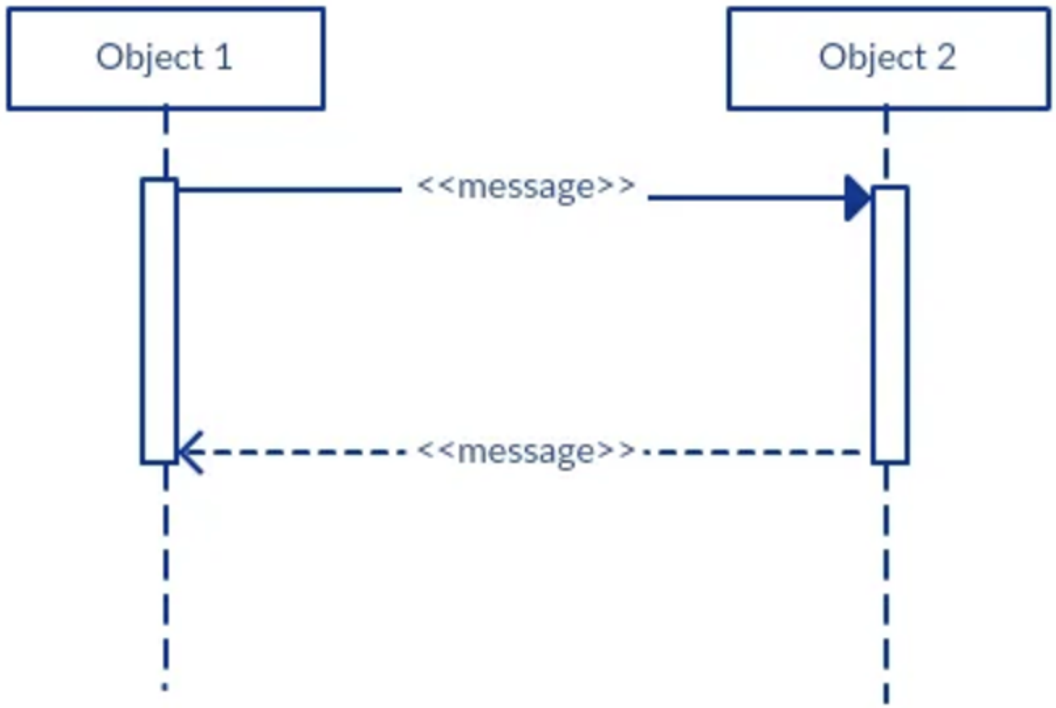
3. Return message

A return message is used to indicate that the message receiver is done processing the message and is returning control over to the message caller.

Return messages are optional notation pieces, for an activation bar that is triggered by a synchronous message always implies a return message.

Notation- Message Arrows

3. Return message



Notation- Message Arrows

4. Participation creation message

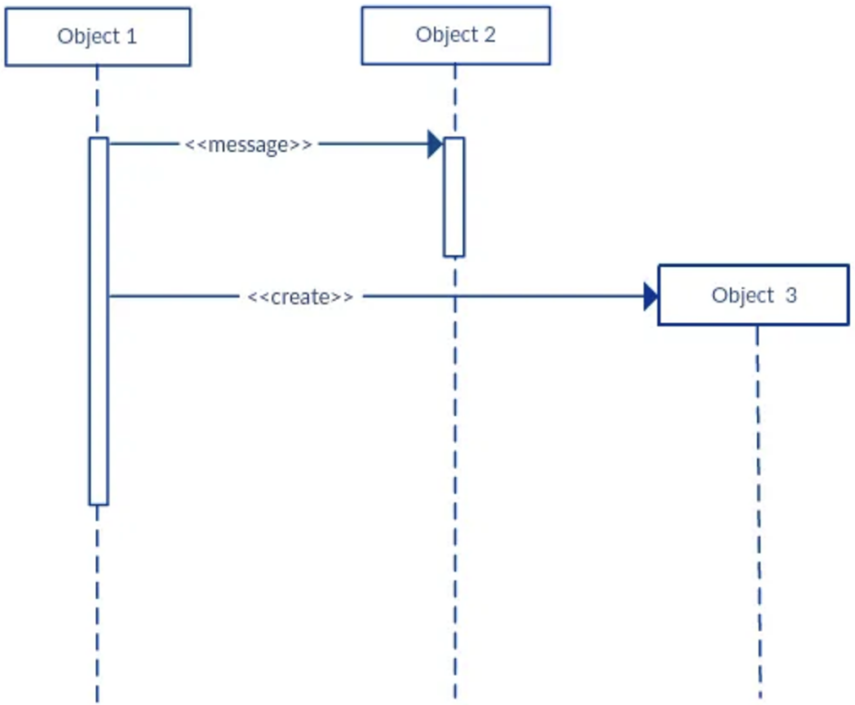
Objects or participants can be created according to the message that is being sent.

The dropped participant box notation can be used when you need to show that the particular participant did not exist until the create call was sent.

If the created participant does something immediately after its creation, you should add an activation box right below the participant box.

Notation- Message Arrows

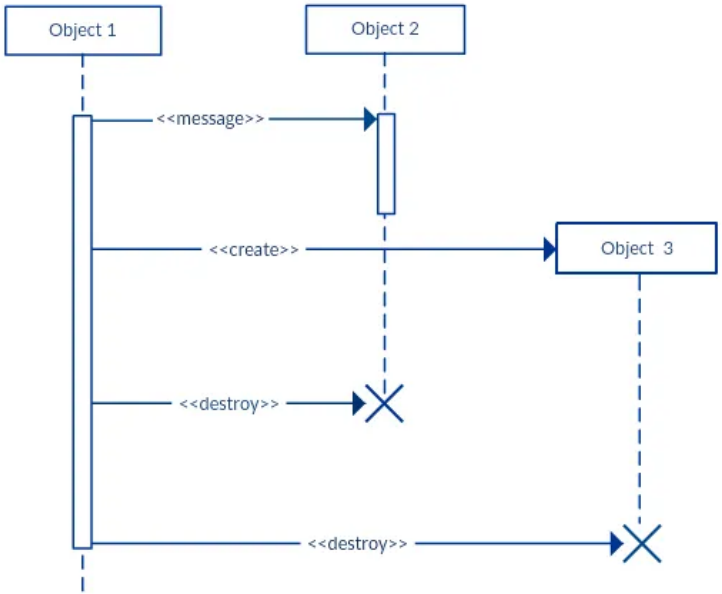
4. Participation creation message



Notation- Message Arrows

5. Participation destruction message

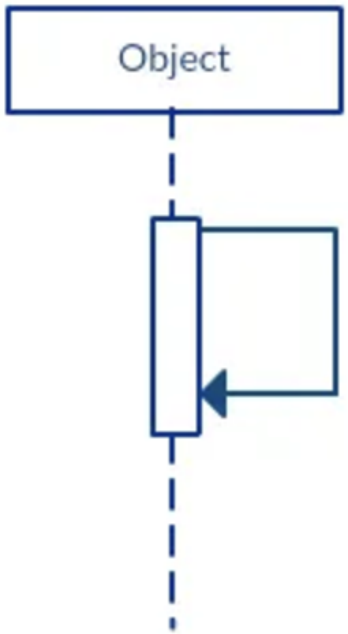
Participants when no longer needed can also be deleted from a sequence diagram. This is done by adding an ‘X’ at the end of the lifeline of the said participant.



Notation- Message Arrows

6. Reflexive message

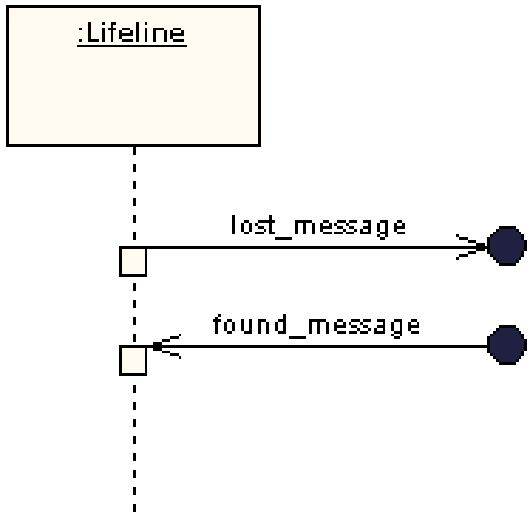
When an object sends a message to itself, it is called a reflexive message. It is indicated with a message arrow that starts and ends at the same lifeline as shown in the example below.



Notation- Message Arrows

6. Lost

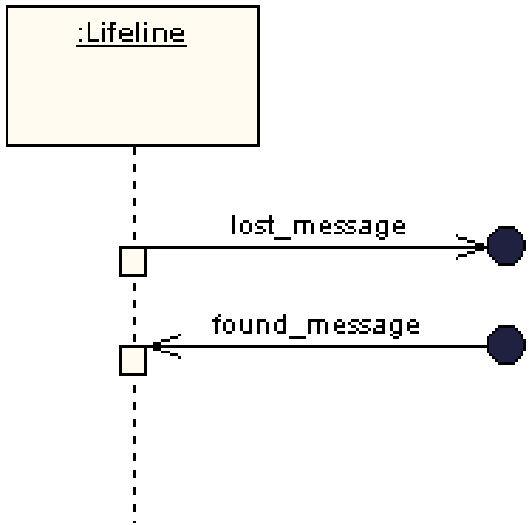
Lost messages are those that are either sent but do not arrive at the intended recipient, or which go to a recipient not shown on the current diagram.



Notation- Message Arrows

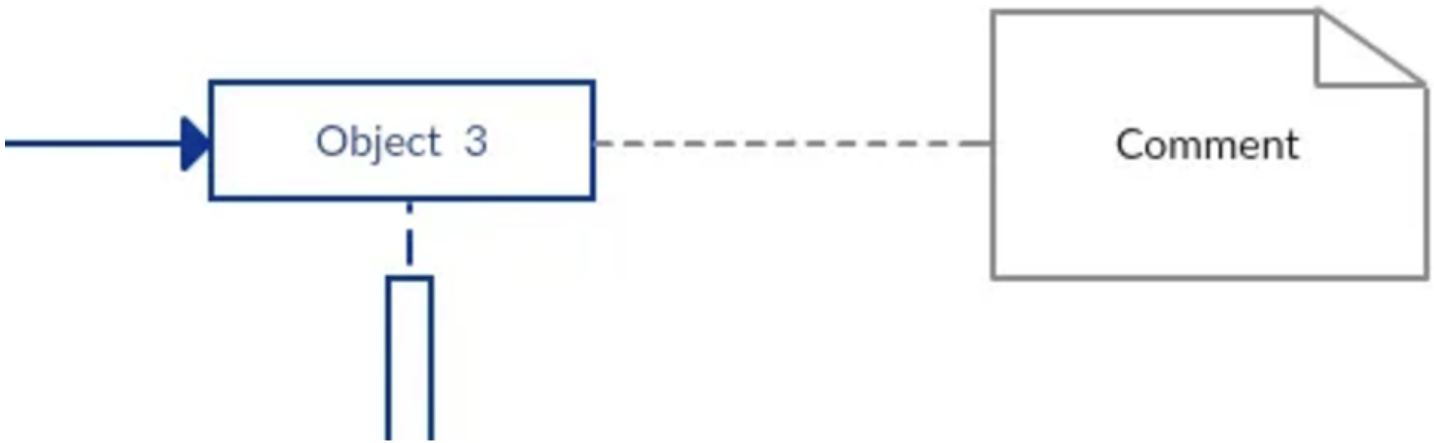
7. Found

Found messages are those that arrive from an unknown sender, or from a sender not shown on the current diagram. They are denoted going to or coming from an endpoint element.



Notation- Comment

The comment object is a rectangle with a folded-over corner as shown below. The comment can be linked to the related object with a dashed line.



Notation- Sequence Fragments

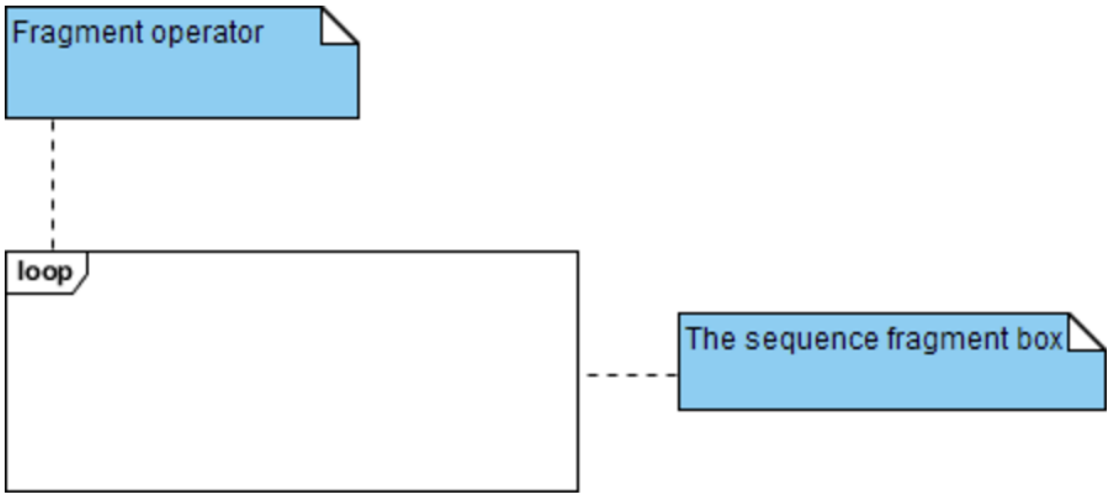
UML 2.0 introduces sequence (or interaction) fragments. Sequence fragments make it easier to create and maintain accurate sequence diagrams

A sequence fragment is represented as a box, called a combined fragment, which encloses a portion of the interactions within a sequence diagram

The fragment operator (in the top left cornet) indicates the type of fragment.

Fragment types: ref, assert, loop, break, alt, opt, neg

Notation- Sequence Fragments



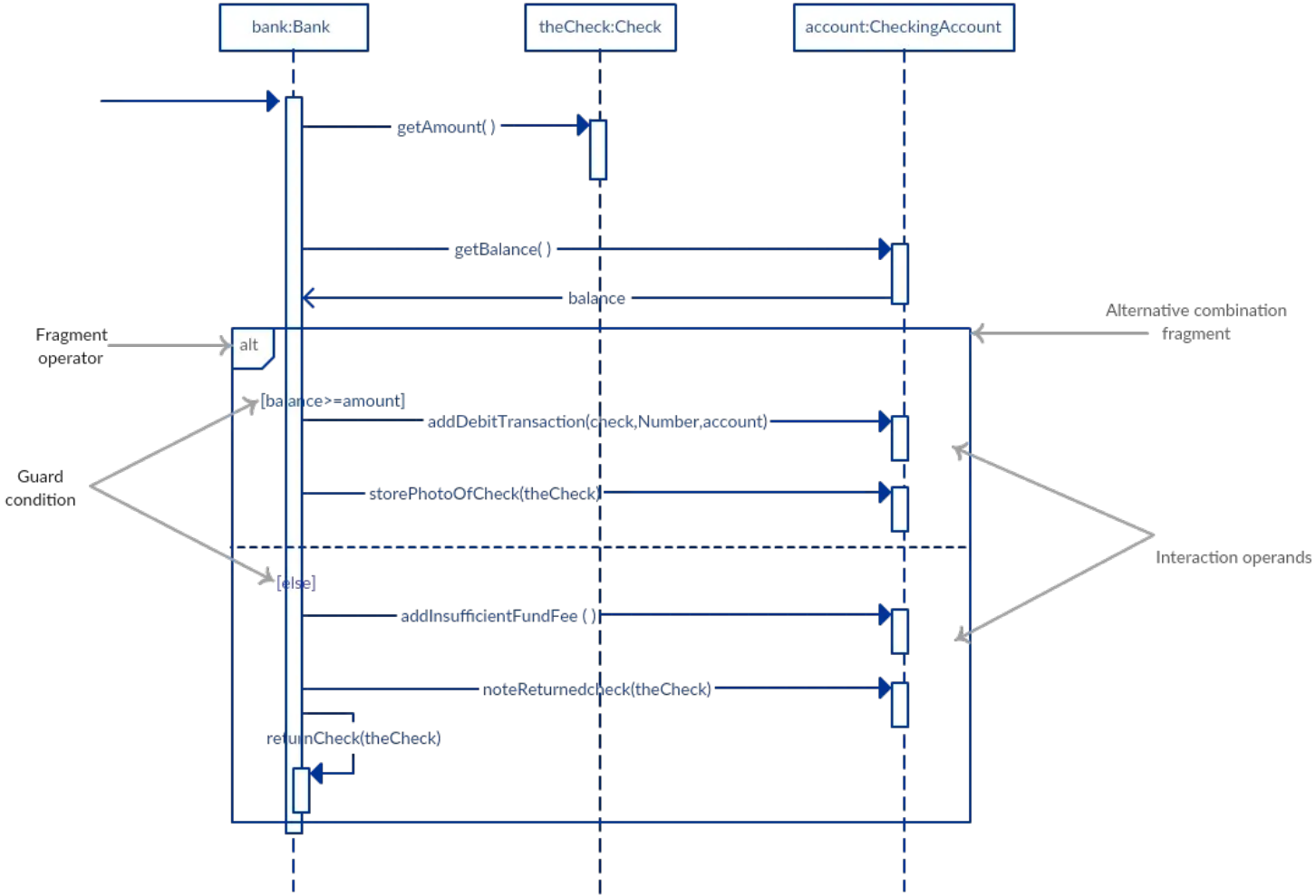
Notation- Sequence Fragments

Operator	Fragment Type
alt	Alternative multiple fragments: only the one whose condition is true will execute.
opt	Optional: the fragment executes only if the supplied condition is true. Equivalent to an alt only with one trace.
par	Parallel: each fragment is run in parallel.
loop	Loop: the fragment may execute multiple times, and the guard indicates the basis of iteration.
region	Critical region: the fragment can have only one thread executing it at once.
neg	Negative: the fragment shows an invalid interaction.
ref	Reference: refers to an interaction defined on another diagram. The frame is drawn to cover the lifelines involved in the interaction. You can define parameters and a return value.
sd	Sequence diagram: used to surround an entire sequence diagram.

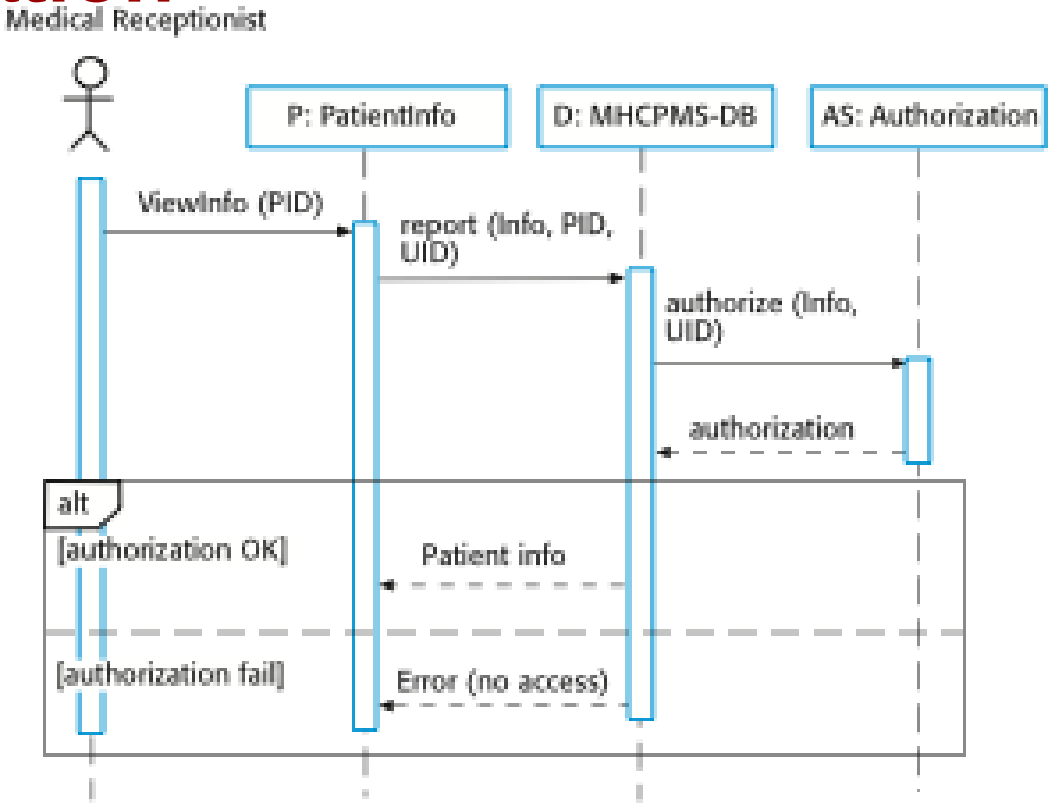
Notation- Sequence Fragments:

Alternatives

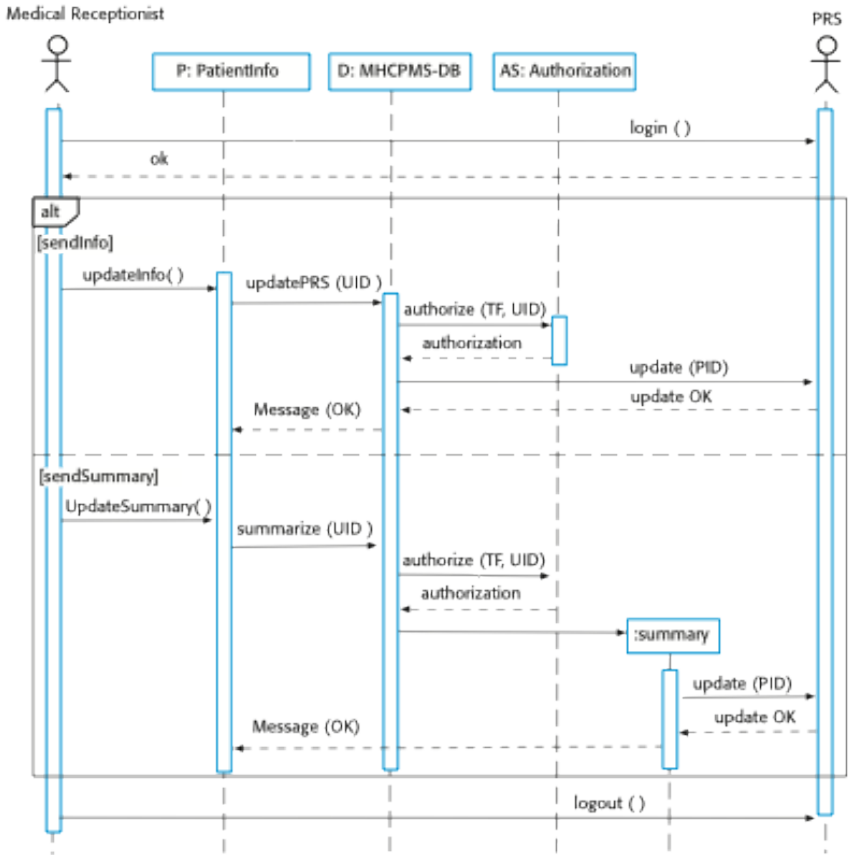
The alternative combination fragment is used when a choice needs to be made between two or more message sequences. It models the “if then else” logic.



Sequence diagram for View patient information



Sequence diagram for Transfer Data



Next Lesson

System Modelling (Cnt)

Q & A