TB141Ic – ICT System Engineering and Rapid Prototyping

System modeling with UML - Use Case and Activity diagrams

Jacopo De Stefani

Delft University of Technology, The Netherlands

22/02/2023

Licensing information

Learning objectives and related literature

Learning objectives

- Distinguish different modeling approaches
- Explain the concept of model-driven-engineering
- Model using Use Case Diagrams and Activity Diagrams
  - Recall the basic components
  - Read an existing diagram
  - Develop a novel diagram for a specific purpose

Related literature

- Chapter 5 [1]
- Online references - https://www.uml-diagrams.org/

[1] I. Sommerville (2011). "Software engineering 9th Edition". In: *ISBN-10* 137035152, p. 18

How do we design a software-based system? [2]

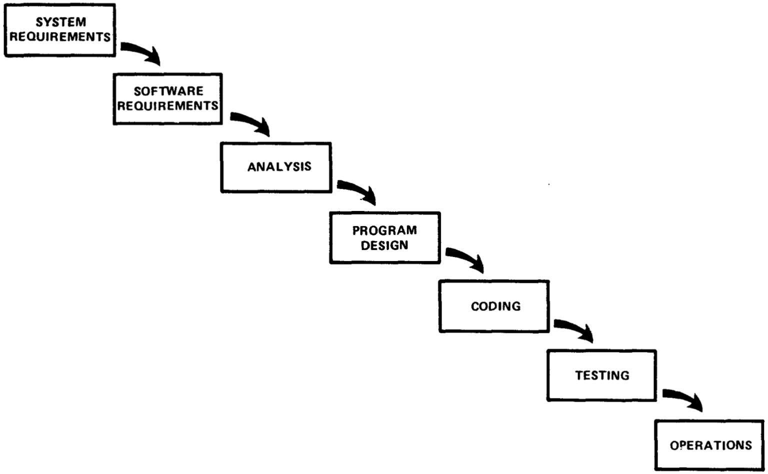
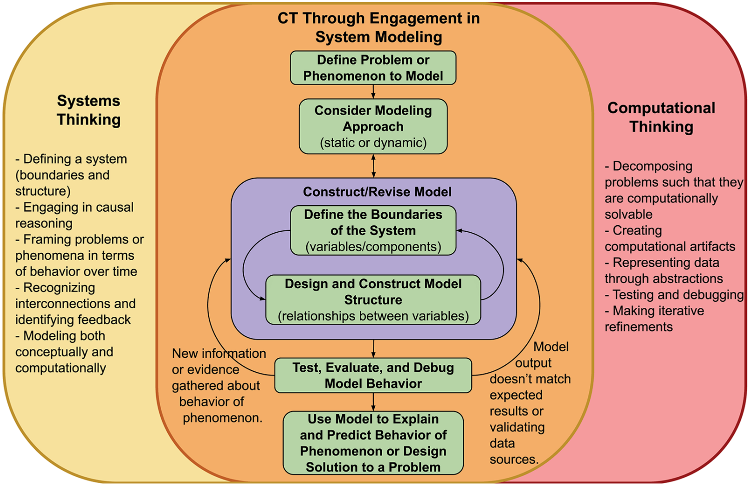


Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

[2] W. W. Royce (1987). "Managing the development of large software systems: concepts and techniques". In: *Proceedings of the 9th international conference on Software Engineering*, pp. 328–338

System modeling


CT Through Engagement in System Modeling
Define Problem or Phenomenon to Model
Consider Modeling Approach (static or dynamic)
Construct/Revise Model
Define the Boundaries of the System (variables/components)
Design and Construct Model Structure (relationships between variables)
Test, Evaluate, and Debug Model Behavior
Use Model to Explain and Predict Behavior of Phenomenon or Design Solution to a Problem
New information or evidence gathered about behavior of phenomenon.
Model output doesn't match expected results or validating data sources.
Systems Thinking
- Defining a system (boundaries and structure)
- Engaging in causal reasoning
- Framing problems or phenomena in terms of behavior over time
- Recognizing interconnections and identifying feedback
- Modeling both conceptually and computationally
Computational Thinking
- Decomposing problems such that they are computationally solvable
- Creating computational artifacts
- Representing data through abstractions
- Testing and debugging
- Making iterative refinements

Source:

`https://concord.org/newsletter/2019-fall/engaging-in-computational-thinking-through-system-modeling/`

System modeling

Working definition

Simplified, abstract representation of reality, made with a specific perspective, for a specific purpose.

- **Specific perspective**: Modeling choices are made to represent only the aspects of reality that are of interest in the model
- **Specific purpose**: Modeling choices are made to fulfill a well-defined goal

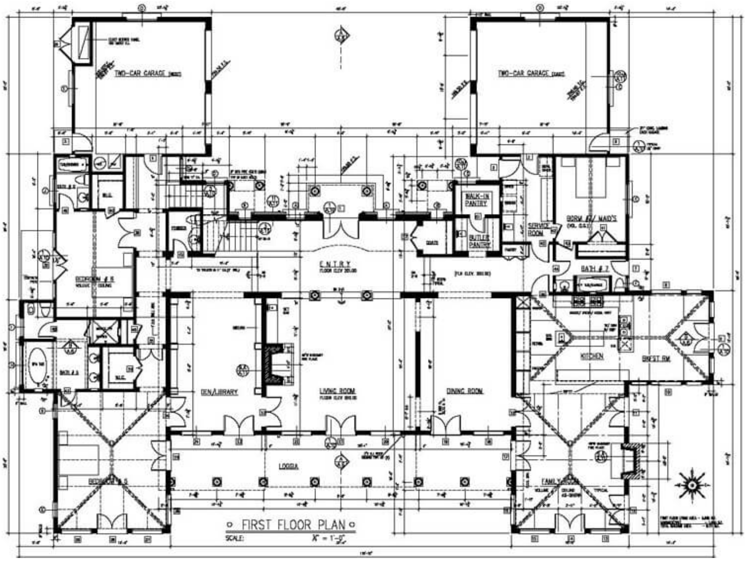
System modeling

In ICT:

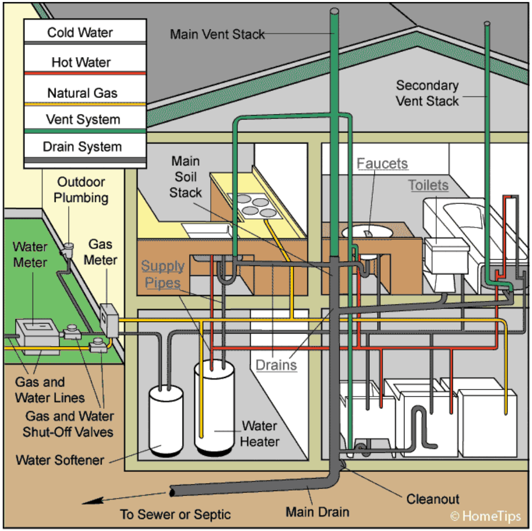
- System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system.
- System modeling has now come to mean representing a system using some kind of graphical notation, which is now almost always based on notations in the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

System modeling - What is a house? - For you

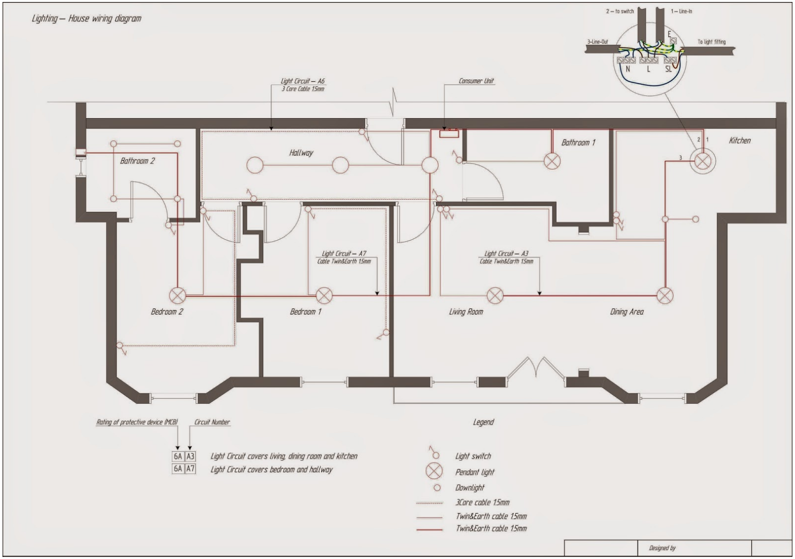
System modeling - What is a house? - For an architect



System modeling - What is a house? - For a plumber



System modeling - What is a house? - For an electrician



Existing and planned system models

Descriptive modeling

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.

Prescriptive modeling

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.

- In a model-driven engineering process, it is possible to generate a complete or partial system implementation from the system model.

Use of graphical models

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

System perspectives

- **External/Context perspective:** where you model the context or environment of the system.
- **Interaction perspective:** where you model the interactions between a system and its environment, or between the components of a system.
- **Structural perspective:** where you model the organization of a system or the structure of the data that is processed by the system.
- **Behavioral perspective:** where you model the dynamic behavior of the system and how it responds to events.

Content Plan

- **Lecture 4:** Context and interaction perspective
- **Lecture 5:** Strucutral and behavioral perspective

Context models

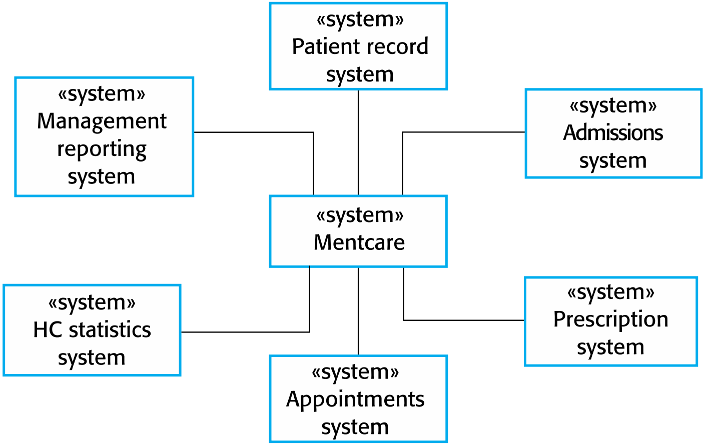
Context models

- 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.

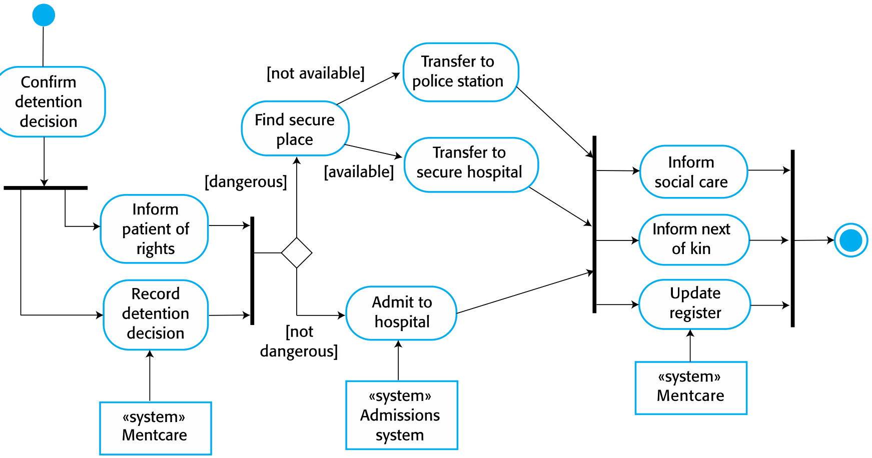
The context of the Mentcare system



Process perspective

- Context models simply show the other systems in the environment, not how the system being developed is used in that environment.
- Process models reveal how the system being developed is used in broader business processes.
- UML activity diagrams may be used to define business process models.

Process model of involuntary detention



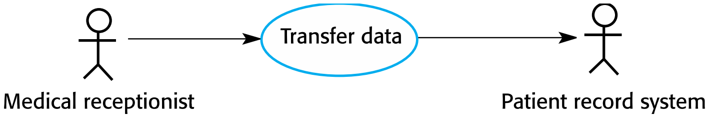
Interaction models

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.
- Use case diagrams and sequence diagrams may be used for interaction modeling.

Transfer-data use case

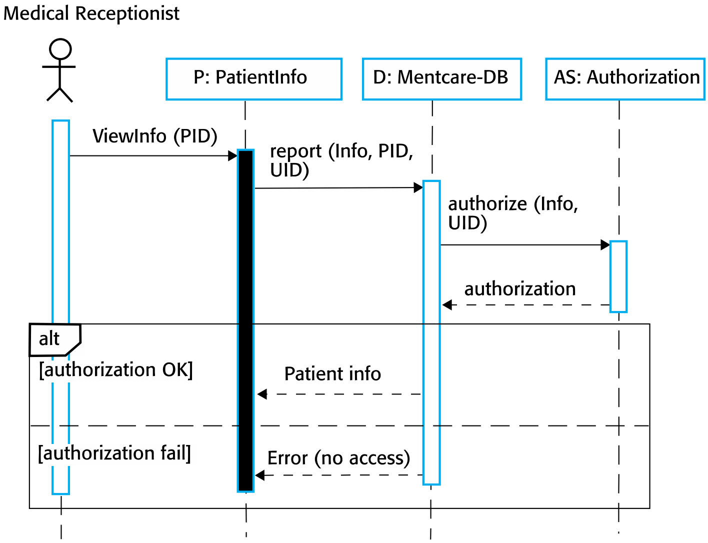
- A use case in the Mentcare system



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.

Sequence diagram for View patient information



Structural models

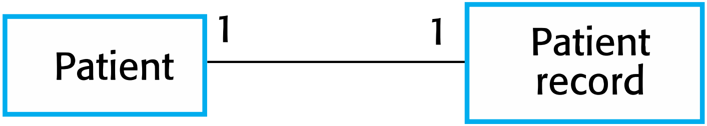
Structural models

- Structural models of software display the organization of a system in terms of the components that make up that system and their relationships.
- Structural models may be static models, which show the structure of the system design, or dynamic models, which show the organization of the system when it is executing.
- You create structural models of a system when you are discussing and designing the system architecture.

Class diagrams

- Class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations between these classes.
- An object class can be thought of as a general definition of one kind of system object.
- An association is a link between classes that indicates that there is some relationship between these classes.
- When you are developing models during the early stages of the software engineering process, objects represent something in the real world, such as a patient, a prescription, doctor, etc.

UML classes and association



Behavioral models

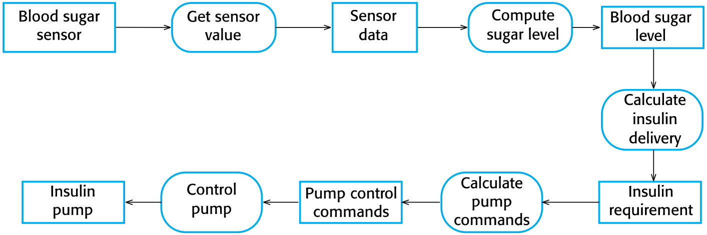
Behavioral models

- Behavioral models are models of the dynamic behavior of a system as it is executing. They show what happens or what is supposed to happen when a system responds to a stimulus from its environment.
- You can think of these stimuli as being of two types:
  - **Data:** Some data arrives that has to be processed by the system.
  - **Events:** Some event happens that triggers system processing. Events may have associated data, although this is not always the case.

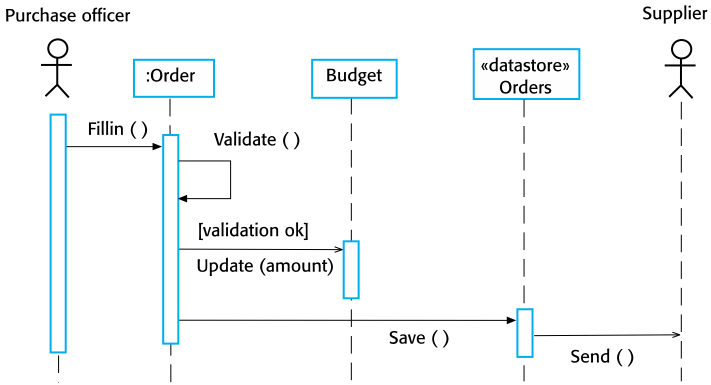
Data-driven modeling

- Many business systems are data-processing systems that are primarily driven by data. They are controlled by the data input to the system, with relatively little external event processing.
- Data-driven models show the sequence of actions involved in processing input data and generating an associated output.
- They are particularly useful during the analysis of requirements as they can be used to show end-to-end processing in a system.

An activity model of the insulin pump's operation



Order processing



Event-driven modeling

- Real-time systems are often event-driven, with minimal data processing. For example, a landline phone switching system responds to events such as ‘receiver off hook’ by generating a dial tone.
- Event-driven modeling shows how a system responds to external and internal events.
- It is based on the assumption that a system has a finite number of states and that events (stimuli) may cause a transition from one state to another.

Model-driven engineering

Model-driven engineering

- Model-driven engineering (MDE) is an approach to software development where models rather than programs are the principal outputs of the development process.
- The programs that execute on a hardware/software platform are then generated automatically from the models.
- Proponents of MDE argue that this raises the level of abstraction in software engineering so that engineers no longer have to be concerned with programming language details or the specifics of execution platforms.

Agile methods and MDE

- The developers of MDE claim that it is intended to support an iterative approach to development and so can be used within agile methods.
- The notion of extensive up-front modeling contradicts the fundamental ideas in the agile manifesto and I suspect that few agile developers feel comfortable with model-driven engineering.
- If transformations can be completely automated and a complete program generated from a Platform Independent Model, then, in principle, MDE could be used in an agile development process as no separate coding would be required.

In this course

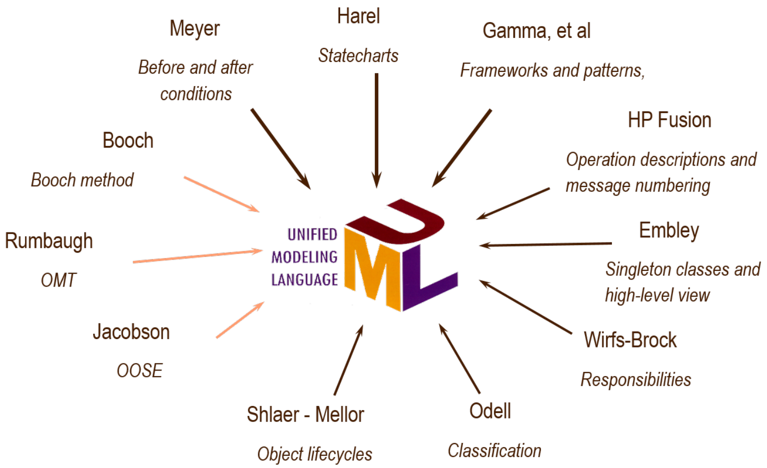
Mendix is an example of MDE approach, in which we will construct a working prototype from models of the view (UI), the model (the data) and the controller (the application logic).

UML modeling

The tower of babel

- In 1994, more than 50 object-oriented modeling methods:
  - Fusion, Shlaer-Mellor, ROOM, Class-Relation, Wirfs-Brock, Coad-Yourdon, MOSES, Syntropy, BOOM, OOSD, OSA, BON, Catalysis, COMMA, HOOD, Ooram, DOORS ...
  - «Meta-models» quite identical
  - Graphical notations differ
  - The process differs or remains vague
- But: Industry needs standards!

Unified Modeling Language



UML Standard

- Designed by Booch, Rumbaugh, Jacobson (3 amigos)
  - Started in 1994; version 1.0 finished in 1997
  - Version 1.5 (1.4.2) since July 2004: current
  - Version 2.0 in beta since 2004 - final late 2005
- They put aside their own methods and notations to end the object-oriented method wars
- Lack of standardization
- Has become the de facto standard

General goals of UML

- Model systems using Object-Oriented concepts
- Establish an explicit coupling to conceptual as well as executable artifacts
- To create a modeling language usable by both humans and machines
- Models different types of systems (information systems, technical systems, embedded systems, real-time systems, distributed systems, system software, business systems, UML itself, ...)
- Two main concepts:
  - Views
  - Diagrams

UML Views - Diagrams mapping

Use case view

Shows the functionality of the system as perceived by external actors.

Diagrams	Used by
Use case	Customers
Activity	Designers
	Developers
	Testers

Component View

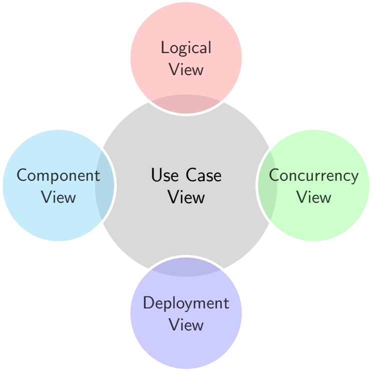
Shows the organization of the code components and their dependencies.

Diagrams	Used by
Component	Developers

Deployment View

Shows the deployment of the system into the physical architecture.

Diagrams	Used by
Deployment	Developers
	Testers
	System
	Integrators



Logical view

Shows how the functionality of the system is designed / provided.

Diagrams	Used by
Class	Developers
State	Designers
Sequence	
Collaboration	
Activity	

Concurrency view

Addresses the problems with communication and synchronization for a concurrent system.

Diagrams	Used by
State	Developers
Sequence	System
Collaboration	Integrators
Activity	Testers
Deployment	
Components	

UML Views

- Each view is a projection of the complete system.
- Each view highlights particular aspects of the system.
  - View ~ Perspective
- Views are described by a number of diagrams.
  - No strict separation, so a diagram can be part of more than one view.

UML Diagrams

- Use-Case diagram
- Class diagram
- Object diagram
- State diagram
- Sequence diagram
- Collaboration diagram
- Activity diagram
- Component diagram
- Deployment diagram

UML Diagrams

- **Use-Case diagram**
- **Class diagram**
- Object diagram
- State diagram
- **Sequence diagram**
- Collaboration diagram
- **Activity diagram**
- Component diagram
- Deployment diagram

UML Diagrams

- Use-Case diagram
- Class diagram
- Object diagram
- State diagram
- Sequence diagram
- Collaboration diagram
- Activity diagram
- Component diagram
- Deployment diagram

My approach to UML

Focus on the (Unified Modeling) **Language** aspect:

- Notation ≡ Grammar
- UML Understanding ≡ Reading
- UML Modeling ≡ Writing/Speaking

Use Case Diagrams

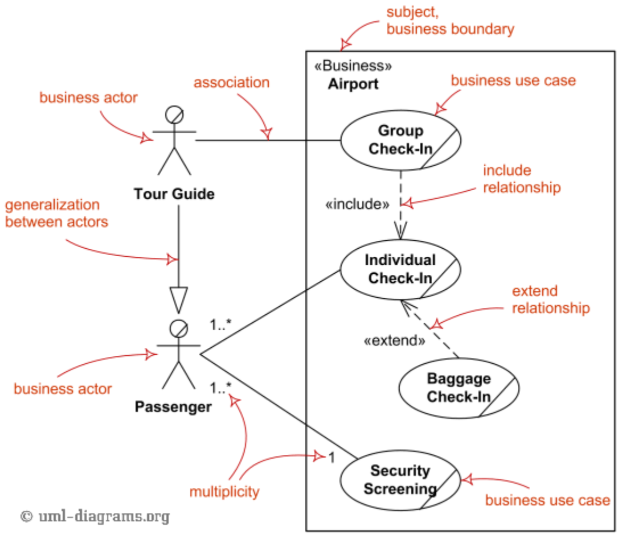
Use case modeling

- 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.
- Actors in a use case may be people or other systems.
- Represented diagramatically to provide an overview of the use case and in a more detailed textual form.

UML Use Cases

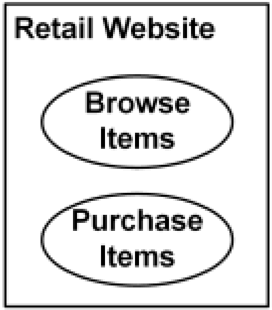
- Can be used to discover requirements and write them down
- Focus on interaction
- Central to UML
- Link users and system builders
- Tend to be better to find out functional user requirements
  - not well suited to find out about constraints, high-level business and non-functional requirements, or domain requirements

Use case diagram - Overview



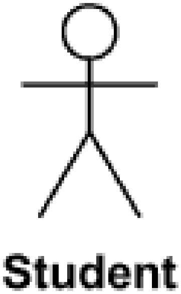
+ Specification

Use case diagram - Elements



Use case + System boundary

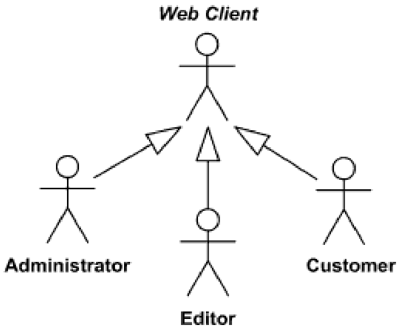
Every use case must have a name. Use case is shown as an ellipse containing the name of the use case. Use cases visually located inside the system boundaries are the use cases applicable to the subject (but not necessarily owned by the subject).



Actor

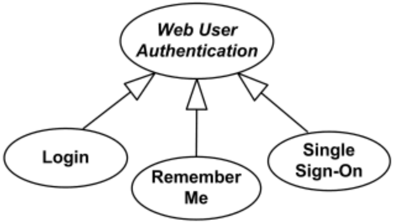
Standard UML icon for actor is "stick man" icon with the name of the actor above or below the icon. Actor names should follow the capitalization and punctuation guidelines for classes. The names of abstract actors should be shown in italics. All actors must have names.

Use case diagram - Relationships - Generalization



Generalization (Actor)

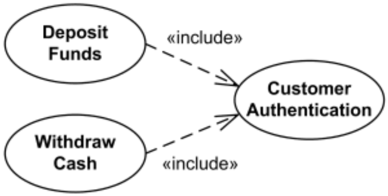
Generalization between actors is rendered as a solid directed line with a large arrowhead (the same as for generalization between classes).



Generalization (Use Case)

Generalization between use cases is similar to generalization between classes – child use case inherits properties and behavior of the parent use case and may override the behavior of the parent. It is rendered as a solid directed line with a large arrowhead, the same as for generalization between classifiers.

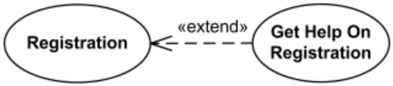
Use case diagram - Relationships - Include/Extends



Include (Use Case)

An include relationship is a directed relationship between two use cases when required, not optional behavior of the included use case is inserted into the behavior of the including (base) use case.
The include relationship is analogous to a subroutine call or macro and could be used: when there are common parts of the behavior of two or more use cases,to simplify large use case by splitting it into several use cases.

An include relationship between use cases is shown by a dashed arrow with an open arrowhead from the base use case to the included use case. The arrow is labeled with the keyword ≪include≫.



Extend (Use Case)

Extend is a directed relationship that specifies how and when the behavior defined in usually supplementary (optional) extending use case can be inserted into the behavior defined in the extended use case.
Extended use case is meaningful on its own, it is independent of the extending use case. Extending use case typically defines optional behavior that is not necessarily meaningful by itself.

Extend relationship between use cases is shown by a dashed arrow with an open arrowhead from the extending use case to the extended (base) use case. The arrow is labeled with the keyword ≪extend≫.

Use Case - Description

- Flows of events specify the behaviour of a use case clearly enough for an outsider to understand it what a system does, not how!
- Several ways:
  - structured text (without/with pre-post conditions)
  - pseudo-code
  - sequence diagrams
  - state diagrams
- A description should include:
  - how and when the use case starts and ends
  - when it interacts with actors and what objects are exchanged
  - the basic flow and alternative flows of the behaviour

Use Case - Components of the specification

- **Actors**
- **Pre-conditions**
- **Post-conditions**
- **Basic Flow**
- **Alternative Flows**
- **Special Requirements**

Tabular description of the ‘Transfer data’ use-case

MHC-PMS: Transfer data	
Actors	Medical receptionist, patient records system (PRS)
Description	A receptionist may transfer data from the Mentcase 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 - Components of the specification

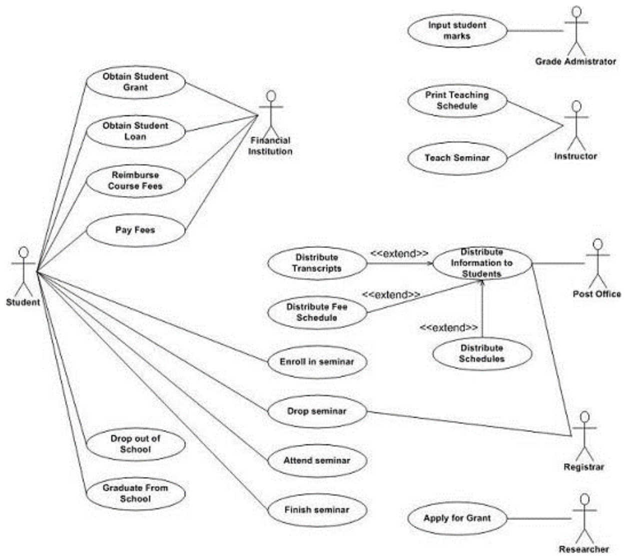
Main flow of events: The use case starts when the system prompts the Customer for a PIN Number. The Customer can now enter a PIN number via the keypad. The Customer commits the entry by pressing the Enter button. The system then checks this PIN number to see if it is valid. If the PIN number is valid, the system acknowledges the entry, thus ending the use case.

Exceptional flow of events: The Customer can cancel a transaction at any time by pressing the Cancel button, thus restarting the use case. No changes are made to the Customer's account.

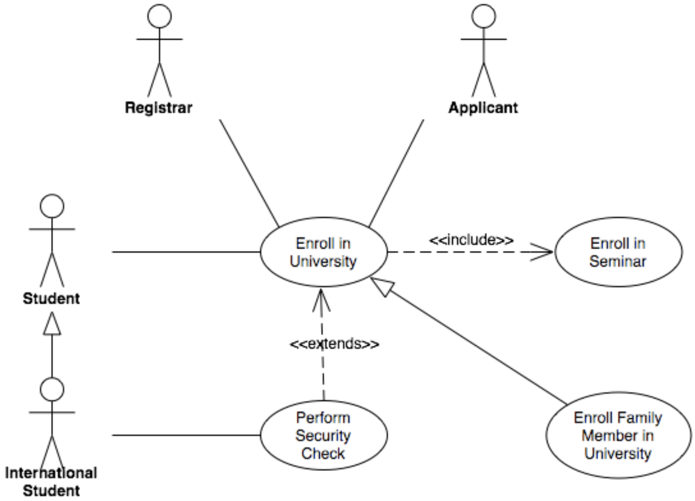
Exceptional flow of events: The Customer can clear a PIN number anytime before committing it and reenter a new PIN number.

Exceptional flow of events: If the Customer enters an invalid PIN number, the use case restarts. If this happens three times in a row, the system cancels the entire transaction, preventing the Customer from interacting with the ATM for 60 seconds.

Use case diagram - Example - Reading - 1



Use case - Example - Reading - 2



Use case - Example - Writing

Exercise 1

Provide use cases for a web customer making purchases online. Focus on the different activities that the user will need to do in its shopping process.

Instructions

- Read the case (2-3 minutes)
- Participative design (7-10 minutes)
- Wrap-up (3-5 minutes)

Purpose: Experiment with modeling
Outcome: UML Use Case specification

Use case - Example - Writing

Exercise 2

Describe use cases that an automated teller machine (ATM) or the automatic banking machine (ABM) provides to the bank customers. Consider also the point of view of a technician that might need to maintain the ATM.

Instructions

- Please do try it at home!

Purpose: Experiment with modeling
Outcome: UML Use Case specification

How to solve the exercise?

1. Read text
2. Identify functionalities → What does the system do? (Verbs)
    - If a document containing the requirements is available → Requirement ≡ Use Case
3. For each functionality:
    - **Actors**
    - **Pre-conditions**
    - **Post-conditions**
    - **Basic Flow**
    - **Alternative Flows**
    - **Special Requirements**

How to solve the exercise?

1. Read text
2. Identify functionalities → What does the system do? (Verbs)
   - If a document containing the requirements is available → Requirement ≡ Use Case
3. For each functionality:
   - **Actors** → Who is doing the action? (Subject)
   - **Pre-conditions**
   - **Post-conditions**
   - **Basic Flow**
   - **Alternative Flows**
   - **Special Requirements**

How to solve the exercise?

1. Read text
2. Identify functionalities → What does the system do? (Verbs)
   - If a document containing the requirements is available → Requirement ≡ Use Case
3. For each functionality:
   - **Actors** → Who is doing the action? (Subject)
   - **Pre-conditions** → What should be valid before doing the action?
   - **Post-conditions**
   - **Basic Flow**
   - **Alternative Flows**
   - **Special Requirements**

How to solve the exercise?

1. Read text
2. Identify functionalities → What does the system do? (Verbs)
    - If a document containing the requirements is available → Requirement ≡ Use Case
3. For each functionality:
    - **Actors** → Who is doing the action? (Subject)
    - **Pre-conditions** → What should be valid before doing the action?
    - **Post-conditions** → What should be valid after doing the action?
    - **Basic Flow**
    - **Alternative Flows**
    - **Special Requirements**

How to solve the exercise?

1. Read text
2. Identify functionalities → What does the system do? (Verbs)
   - If a document containing the requirements is available → Requirement ≡ Use Case
3. For each functionality:
   - **Actors** → Who is doing the action? (Subject)
   - **Pre-conditions** → What should be valid before doing the action?
   - **Post-conditions** → What should be valid after doing the action?
   - **Basic Flow** → Which is the regular flow of the action?
   - **Alternative Flows**
   - **Special Requirements**

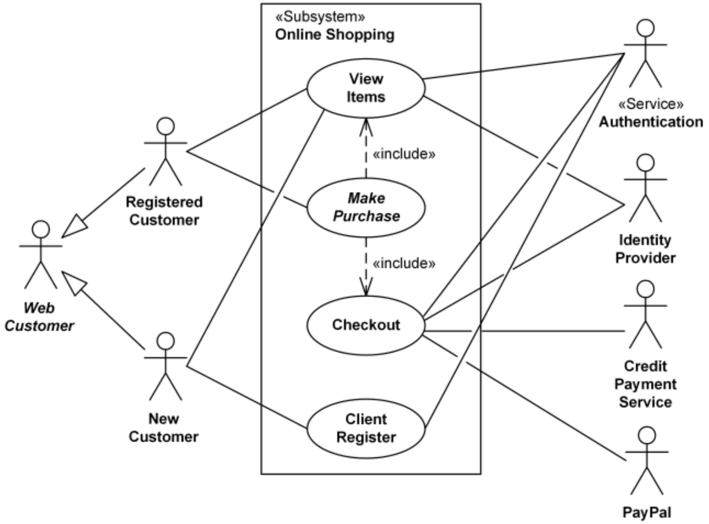
How to solve the exercise?

1. Read text
2. Identify functionalities → What does the system do? (Verbs)
   - If a document containing the requirements is available → Requirement ≡ Use Case
3. For each functionality:
   - **Actors** → Who is doing the action? (Subject)
   - **Pre-conditions** → What should be valid before doing the action?
   - **Post-conditions** → What should be valid after doing the action?
   - **Basic Flow** → Which is the regular flow of the action?
   - **Alternative Flows** → What are the exceptional flows of the action?
   - **Special Requirements**

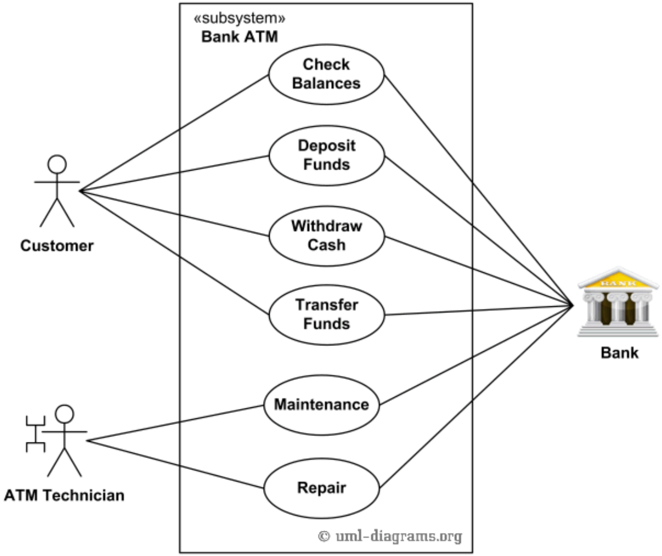
How to solve the exercise?

1. Read text
2. Identify functionalities → What does the system do? (Verbs)
    - If a document containing the requirements is available → Requirement ≡ Use Case
3. For each functionality:
    - **Actors** → Who is doing the action? (Subject)
    - **Pre-conditions** → What should be valid before doing the action?
    - **Post-conditions** → What should be valid after doing the action?
    - **Basic Flow** → Which is the regular flow of the action?
    - **Alternative Flows** → What are the exceptional flows of the action?
    - **Special Requirements** → Is there any additional requirement worth mentioning?

Use case - Example 1 - Writing - Possible solution



Use case - Example 2 - Writing - Possible solution



Tips and tricks - Modeling

A well-structured use case:

- factors common behaviour by using inclusion
- factors variants by using extension
- describes the flow of events clearly enough for an outsider to easily understand it
- uses a minimal set of descriptions that specify the normal and variant semantics of the use case

Tips and tricks - Drawing

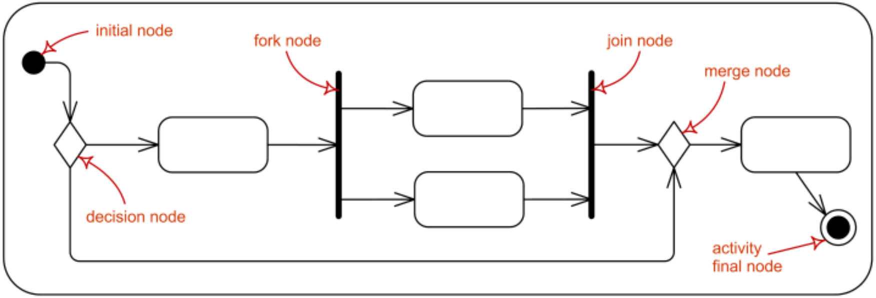
- **When drawing a use case:**
  - show only those use case that are important to understand the behaviour
  - show only the actors relating to these use cases
- **When drawing a use case diagram:**
  - give it a name communicating its purpose
  - minimize lines that cross
  - semantically close elements should be physically close
  - Use different diagrams when needed

Activity Diagrams

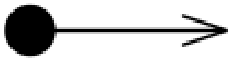
Activity Diagrams

- Can be used to support interaction modeling
- Focus on flow of events/objects/activities
- Allows to specify sequence/parallelism
- Support the visualization of activities across components
- Usually employed for business process modeling

Activity diagram - Overview

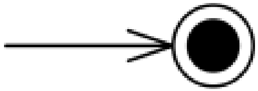


Activity diagram - Details



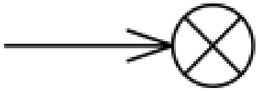
Flow initial

Initial node is a control node at which flow starts when the activity is invoked. Activity may have more than one initial node. Initial nodes are shown as a small solid circle.



Activity final

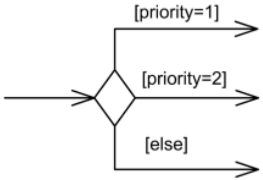
Activity final node is a control final node that stops all flows in an activity. Activity final is new in UML 2.0. Activity final nodes are shown as a solid circle with a hollow circle inside. It can be thought of as a goal notated as "bull's eye," or target.



Flow final

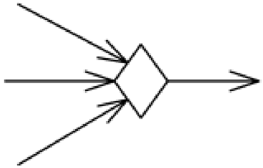
Flow final node is a control final node that terminates a flow. The notation for flow final node is small circle with X inside.

Activity diagram - Details



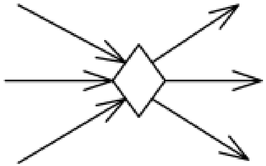
Decision node

Decision node is a control node that accepts tokens on one or two incoming edges and selects one outgoing edge from one or more outgoing flows. The notation for a decision node is a diamond-shaped symbol.



Merge node

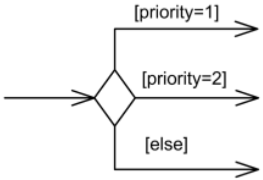
Merge node is a control node that brings together multiple incoming alternate flows to accept single outgoing flow. There is no joining of tokens. Merge should not be used to synchronize concurrent flows. The notation for a merge node is a diamond-shaped symbol with two or more edges entering it and a single activity edge leaving it.



Merge and decision node

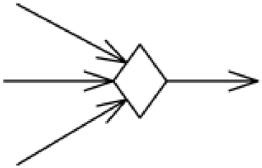
The functionality of merge node and decision node can be combined by using the same node symbol.

Activity diagram - Details



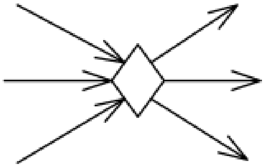
Decision node

Decision node is a control node that accepts tokens on one or two incoming edges and selects one outgoing edge from one or more outgoing flows. The notation for a decision node is a diamond-shaped symbol.



Merge node

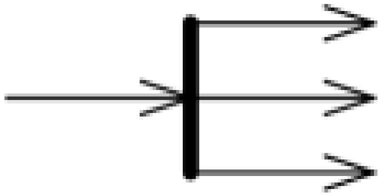
Merge node is a control node that brings together multiple incoming alternate flows to accept single outgoing flow. There is no joining of tokens. Merge should not be used to synchronize concurrent flows. The notation for a merge node is a diamond-shaped symbol with two or more edges entering it and a single activity edge leaving it.



Merge and decision node

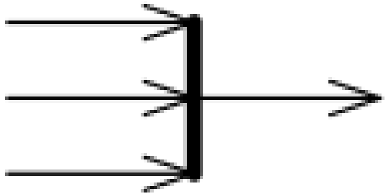
The functionality of merge node and decision node can be combined by using the same node symbol.

Activity diagram - Details



Fork node

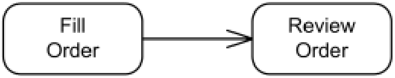
Fork node is a control node that has one incoming edge and multiple outgoing edges and is used to split incoming flow into multiple concurrent flows. The notation for a fork node is a line segment with a single activity edge entering it, and two or more edges leaving it.



Join node

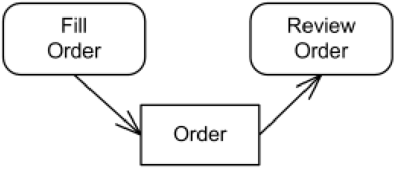
Join node is a control node that has multiple incoming edges and one outgoing edge and is used to synchronize incoming concurrent flows. The notation for a join node is a line segment with several activity edges entering it, and only one edge leaving it.

Activity diagram - Details



Activity edge

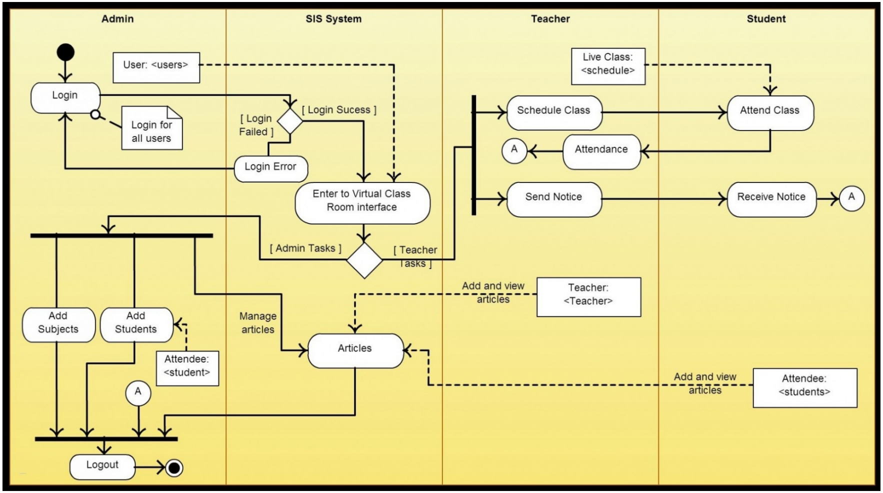
Activity edge could be control edge or data flow edge (aka object flow edge). Both are notated by an open arrowhead line connecting activity nodes.



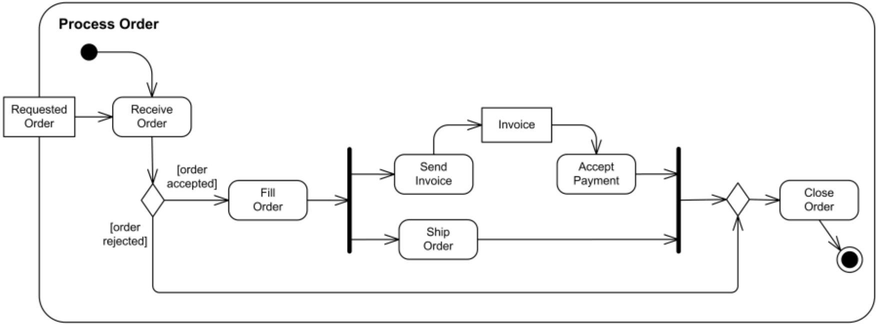
Object flow edge

Object flow edges are activity edges used to show data flow between action nodes. Object flow edges have no specific notation.

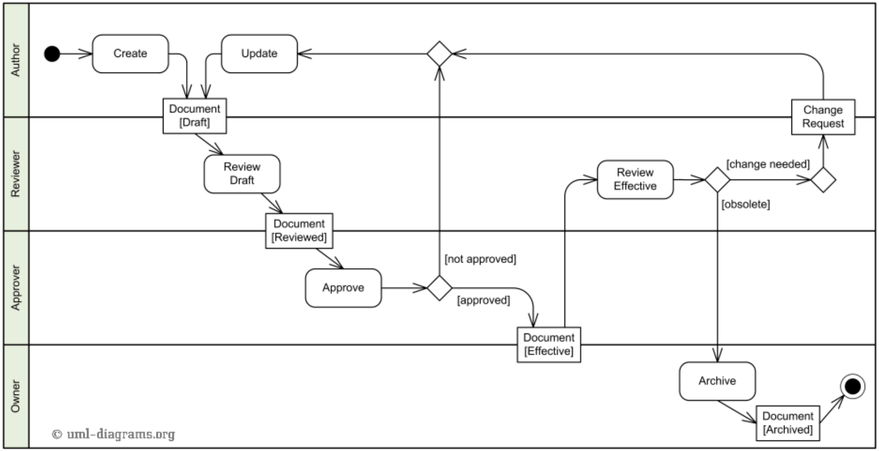
Activity diagram - Example - Reading - 1



Activity diagram - Example - Reading - 2



Activity diagram - Example - Reading - 3



Activity diagram - Example - Writing

Exercise 1

Develop an activity diagram based on the following narrative. The purpose of the Open Access Insurance System is to provide automotive insurance to car owners. Initially, prospective customers fill out an insurance application, which provides information about the customer and his or her vehicles. This information is sent to an agent, who sends it to various insurance companies to get quotes for insurance. When the responses return, the agent then determines the best policy for the type and level of coverage desired and gives the customer a copy of the insurance policy proposal and quote.

Instructions

- Read the case (2-3 minutes)
- Participative design (7-10 minutes)
- Wrap-up (3-5 minutes)

Purpose: Experiment with modeling
Outcome: UML activity diagram

Activity diagram - Example - Writing

Exercise 2

The purchasing department handles purchase requests from other departments in the company. People in the company who initiate the original purchase request are the "customers" of the purchasing department. A case worker within the purchasing department receives that request and monitors it until it is ordered and received. Case workers process the requests for purchasing products under $1,500, write a purchase order, and then send it to the approved vendor. Purchase requests over $1,500 must first be sent out for a bid from the vendor that supplies the product. When the bids return, the case worker selects one bid. Then, the case worker writes a purchase order and sends it to the approved vendor.

Instructions

- Please do try it at home!

Purpose: Experiment with modeling
Outcome: UML Activity diagram

How to solve the exercise?

1. Read text
2. Identify actions → What activities are involved? (Verbs)
3. Identify keywords:
   - **Subjects**
   - **before/after**
   - **simultaneously/at the same time/...**
   - **if/in case of/...**

How to solve the exercise?

1. Read text
2. Identify actions → What activities are involved? (Verbs)
3. Identify keywords:
    - **Subjects** → Who is doing the action?
    - **before/after**
    - **simultaneously/at the same time/...**
    - **if/in case of/...**

How to solve the exercise?

1. Read text
2. Identify actions → What activities are involved? (Verbs)
3. Identify keywords:
    - **Subjects** → Who is doing the action?
    - **before/after** → Concept of sequence flow between the actions
    - **simultaneously/at the same time/...**
    - **if/in case of/...**

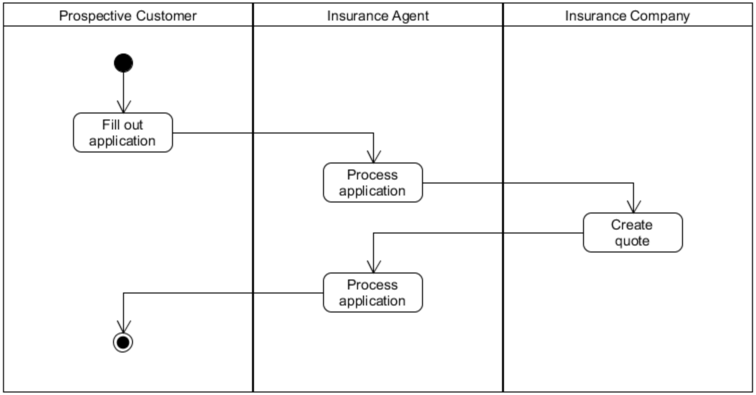
How to solve the exercise?

1. Read text
2. Identify actions → What activities are involved? (Verbs)
3. Identify keywords:
    - **Subjects** → Who is doing the action?
    - **before/after** → Concept of sequence flow between the actions
    - **simultaneously/at the same time/...** → Concept of parallel flow between the actions
    - **if/in case of/...**

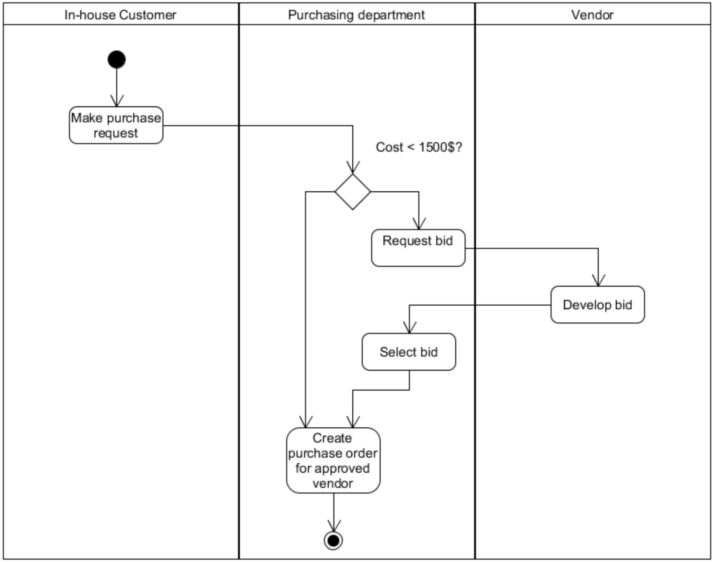
How to solve the exercise?

1. Read text
2. Identify actions → What activities are involved? (Verbs)
3. Identify keywords:
    - **Subjects** → Who is doing the action?
    - **before/after** → Concept of sequence flow between the actions
    - **simultaneously/at the same time/...** → Concept of parallel flow between the actions
    - **if/in case of/...** → Concept of alternative flow between the actions

Activity diagram - Writing 1 - Possible solution



Activity diagram - Writing 2 - Possible solution



Key points

- A model is an abstract view of a system that ignores system details. Complementary system models can be developed to show the system's context, interactions, structure and behavior.
- Context models show how a system that is being modeled is positioned in an environment with other systems and processes.
- Use case diagrams and sequence diagrams are used to describe the interactions between users and systems in the system being designed. Use cases describe interactions between a system and external actors; sequence diagrams add more information to these by showing interactions between system objects.
- Structural models show the organization and architecture of a system. Class diagrams are used to define the static structure of classes in a system and their associations.

Key points

- Behavioral models are used to describe the dynamic behavior of an executing system. This behavior can be modeled from the perspective of the data processed by the system, or by the events that stimulate responses from a system.
- Activity diagrams may be used to model the processing of data, where each activity represents one process step.
- State diagrams are used to model a system's behavior in response to internal or external events.
- Model-driven engineering is an approach to software development in which a system is represented as a set of models that can be automatically transformed to executable code.

Formative Assignment - UML Modeling

- The formative assignment on UML modeling will be available on Brightspace on Friday
- The deadline for the assignment is the **17/03**

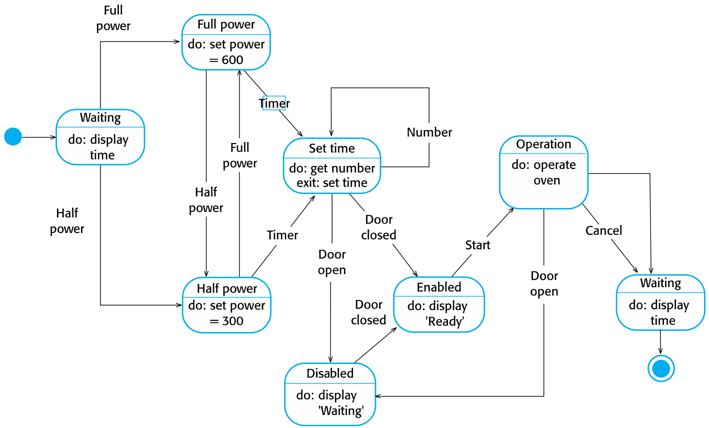
Appendix

Interaction modeling

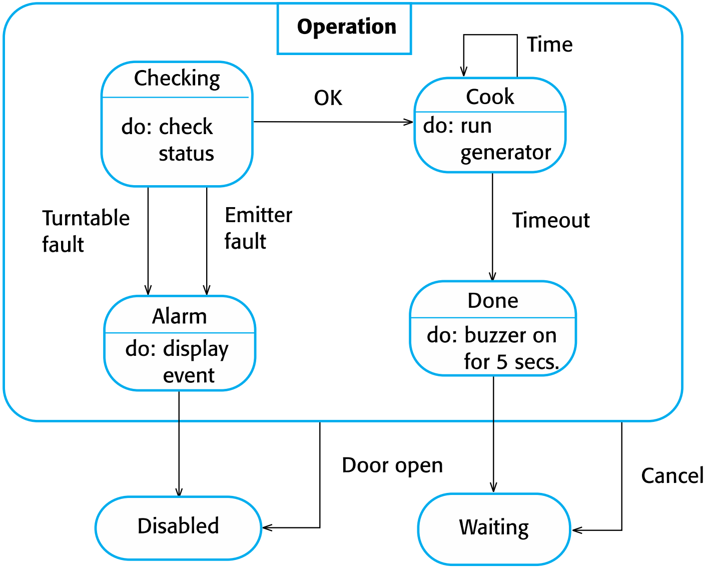
State machine models

- These model the behaviour of the system in response to external and internal events.
- They show the system's responses to stimuli so are often used for modelling real-time systems.
- State machine models show system states as nodes and events as arcs between these nodes. When an event occurs, the system moves from one state to another.
- Statecharts are an integral part of the UML and are used to represent state machine models.

State diagram of a microwave oven



Microwave oven operation



States and stimuli for the microwave oven (a)

State	Description
Waiting	The oven is waiting for input. The display shows the current time.
Half power	The oven power is set to 300 watts. The display shows ‘Half power’.
Full power	The oven power is set to 600 watts. The display shows ‘Full power’.
Set time	The cooking time is set to the user’s input value. The display shows the cooking time selected and is updated as the time is set.
Disabled	Oven operation is disabled for safety. Interior oven light is on. Display shows ‘Not ready’.
Enabled	Oven operation is enabled. Interior oven light is off. Display shows ‘Ready to cook’.
Operation	Oven in operation. Interior oven light is on. Display shows the timer countdown. On completion of cooking, the buzzer is sounded for five seconds. Oven light is on. Display shows ‘Cooking complete’ while buzzer is sounding.

States and stimuli for the microwave oven (b)

Stimulus	**Description**
Half power	The user has pressed the half-power button.
Full power	The user has pressed the full-power button.
Timer	The user has pressed one of the timer buttons.
Number	The user has pressed a numeric key.
Door open	The oven door switch is not closed.
Door closed	The oven door switch is closed.
Start	The user has pressed the Start button.
Cancel	The user has pressed the Cancel button.

Model-driven engineering

Usage of model-driven engineering

- Model-driven engineering is still at an early stage of development, and it is unclear whether or not it will have a significant effect on software engineering practice.
- **Pros**
  - Allows systems to be considered at higher levels of abstraction
  - Generating code automatically means that it is cheaper to adapt systems to new platforms.
- **Cons**
  - Models for abstraction and not necessarily right for implementation.
  - Savings from generating code may be outweighed by the costs of developing translators for new platforms.

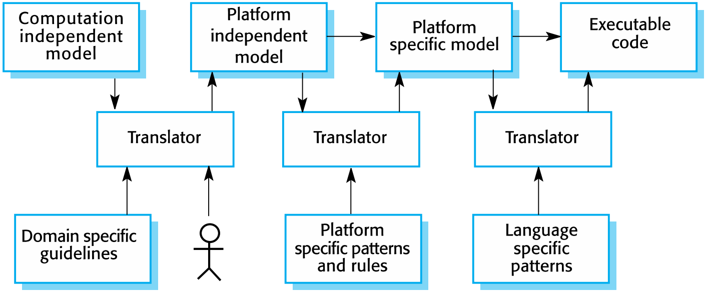
Model driven architecture

- Model-driven architecture (MDA) was the precursor of more general model-driven engineering
- MDA is a model-focused approach to software design and implementation that uses a subset of UML models to describe a system.
- Models at different levels of abstraction are created. From a high-level, platform independent model, it is possible, in principle, to generate a working program without manual intervention.

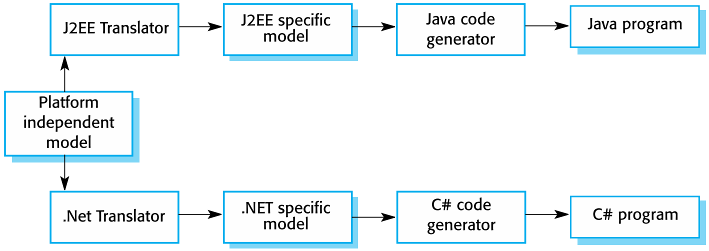
Types of model

- A computation independent model (CIM)
  - These model the important domain abstractions used in a system. CIMs are sometimes called domain models.
- A platform independent model (PIM)
  - These model the operation of the system without reference to its implementation. The PIM is usually described using UML models that show the static system structure and how it responds to external and internal events.
- Platform specific models (PSM)
  - These are transformations of the platform-independent model with a separate PSM for each application platform. In principle, there may be layers of PSM, with each layer adding some platform-specific detail.

MDA transformations


Computation independent model
Platform independent model
Platform specific model
Executable code
Translator
Translator
Translator
Domain specific guidelines
Platform specific patterns and rules
Language specific patterns

Multiple platform-specific models



Adoption of MDA

- A range of factors has limited the adoption of MDE/MDA
- Specialized tool support is required to convert models from one level to another
- There is limited tool availability and organizations may require tool adaptation and customisation to their environment
- For the long-lifetime systems developed using MDA, companies are reluctant to develop their own tools or rely on small companies that may go out of business

Adoption of MDA

- Models are a good way of facilitating discussions about a software design. However the abstractions that are useful for discussions may not be the right abstractions for implementation.
- For most complex systems, implementation is not the major problem – requirements engineering, security and dependability, integration with legacy systems and testing are all more significant.

Adoption of MDA

- The arguments for platform-independence are only valid for large, long-lifetime systems. For software products and information systems, the savings from the use of MDA are likely to be outweighed by the costs of its introduction and tooling.
- The widespread adoption of agile methods over the same period that MDA was evolving has diverted attention away from model-driven approaches.