Introduction to UML

Lecture # 1

Department of Computer Science and Technology
University of Bedfordshire

Written by David Goodwin,
based on the book *Applying UML and Patterns (3rd ed.)*
by *C. Larman* (2005).

Modelling and Simulation, 2012

Outline

Course Information

Aims and Objectives

- To understand the software development process, including requirement specification, analysis, design, implementation and testing.
- To learn and use various methodologies in software development,
- To understand the process of modelling real world problems and systems using UML,
- To develop skills on object oriented software development (OOSD).

Assesment

- Assignment:
  - One long, report style assignment, 50% (due 1630 27^{th} September 2012)
- Exam:
  - Final exam (Perception), 50% (2 hours, 1900-2100 27^{th} September 2012)

Books

- Applying UML and patterns by Craig Larman
- Object-Oriented Software Engineering – a use case driven approach (revised edition) by Ivar Jacobson
- UML Distilled (2nd Edition) by Martin Fowler
- Software Engineering (4th Edition) by Ian Sommerville
- Developing Applications with Java and UML by Paul R and Reed Jr
- Practical Software Engineering by Leszek A Maciaszek and Bruc Lee Liong

UML

What is UML?

- UML stands for Unified Modelling Language.
- An industry standard modelling language for object-oriented software engineering.
- Developed in the mid-1990's and standardised in 1997 (UML 1.1).
- UML 2.x is the current revision in use (we will focus on UML 2.0, revision from 2005).
- UML includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems.

List of object-oriented programming languages

- Languages designed mainly for object-oriented programming:
  - C++
  - Java
  - C#
  - Python
- Languages with some object-oriented features:
  - Visual Basic
  - Fortran
  - Perl
  - PHP

List of Unified Modelling Language tools

NAME	PLATFORM	LICENSE
ArgoUML	Cross-platform (Java)	Open source
astah*	Multi-platform	Commercial, Free Community version
Dia	Cross-platform (GTK+)	Open source
Modelio	Windows, Linux	Open source
Rational Rhapsody	Windows, Linux, MacOS X	Commercial
Software Ideas Modeler	Windows, Linux	Commercial, Freeware for non-commercial
StarUML	Windows	Open source
Umbrello UML Modeller	Unix-like; Windows	Open source
Visual Paradigm for UML	Cross-platform (Java)	Commercial, Free Community Edition

UML 2.0

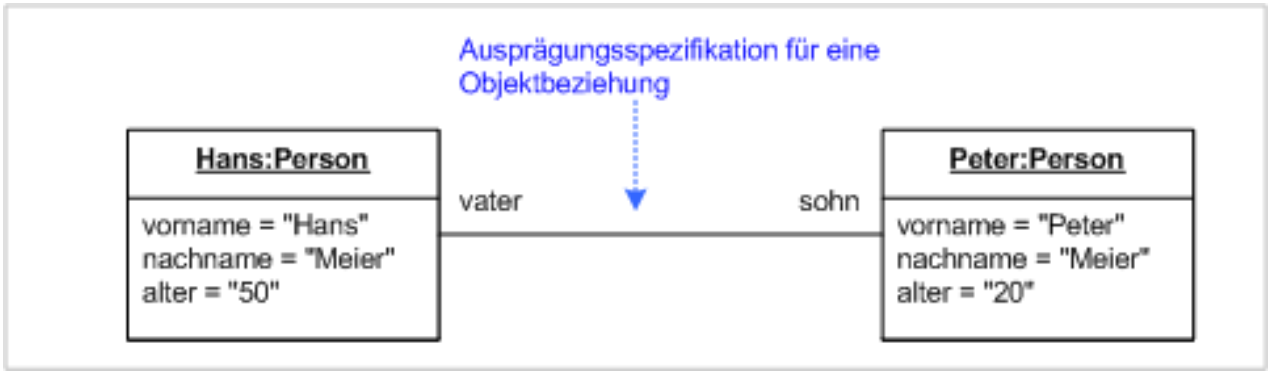
- UML 2.0 defines thirteen types of diagrams:
  - divided into three categories:
    - Six diagram types represent static application **structure**;
    - Three diagram types represent general types of **behaviour**;
    - Four diagram types represent different aspects of **interactions**.

Class Diagram

Shows a collection of static model elements such as classes and types, their contents, and their relationships.

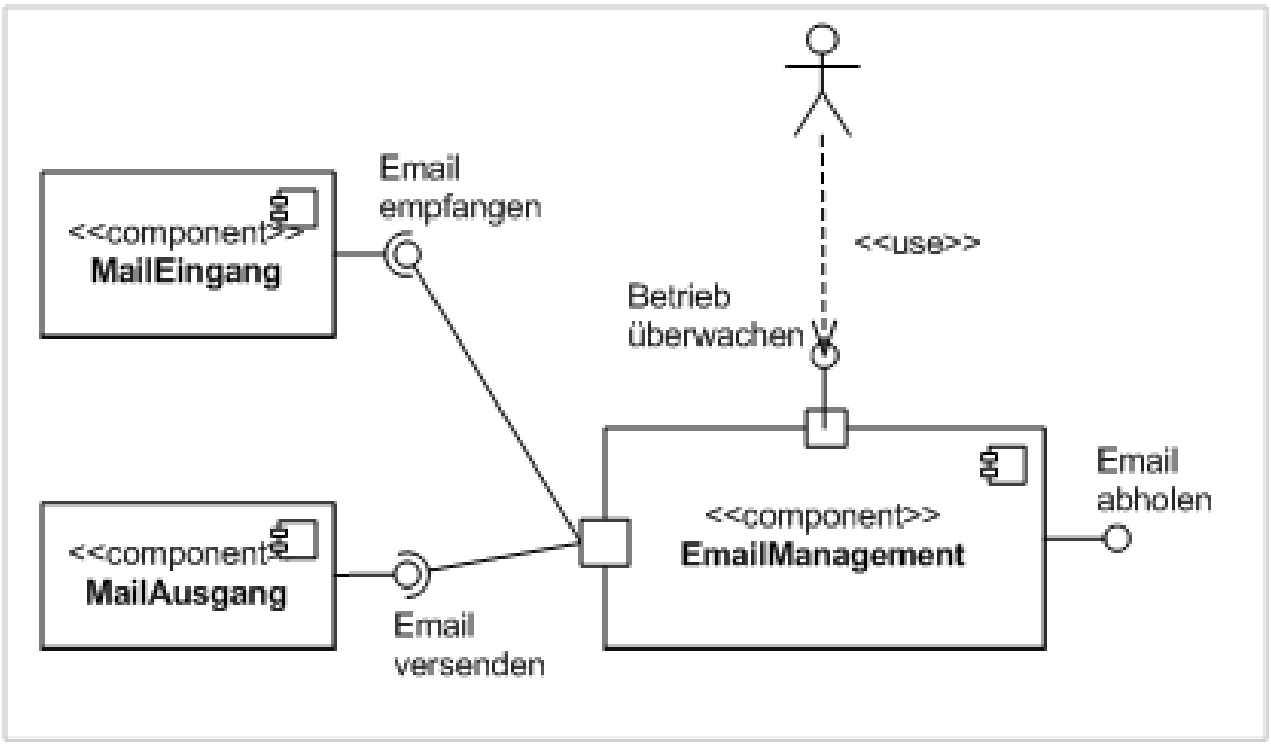
Object Diagram

Depicts objects and their relationships at a point in time, typically a special case of either a class diagram or a communication diagram.



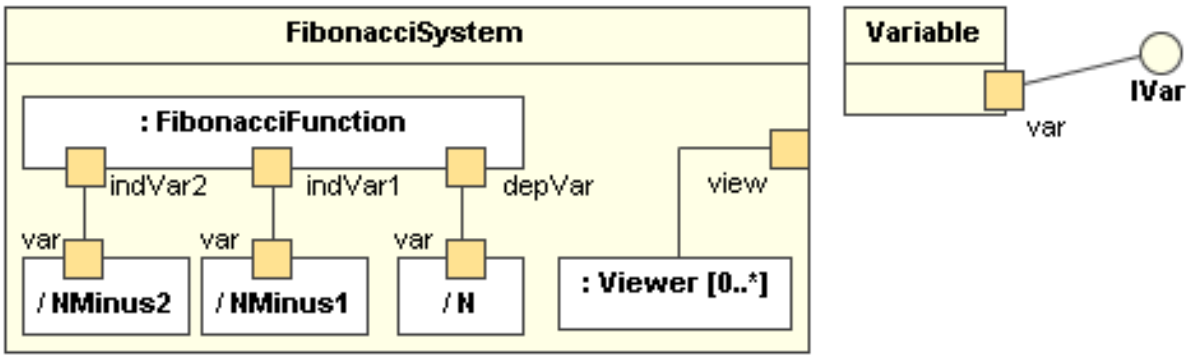
Component Diagram

Depicts the components that compose an application, system, or enterprise. The components, their interrelationships, interactions, and their public interfaces are depicted.



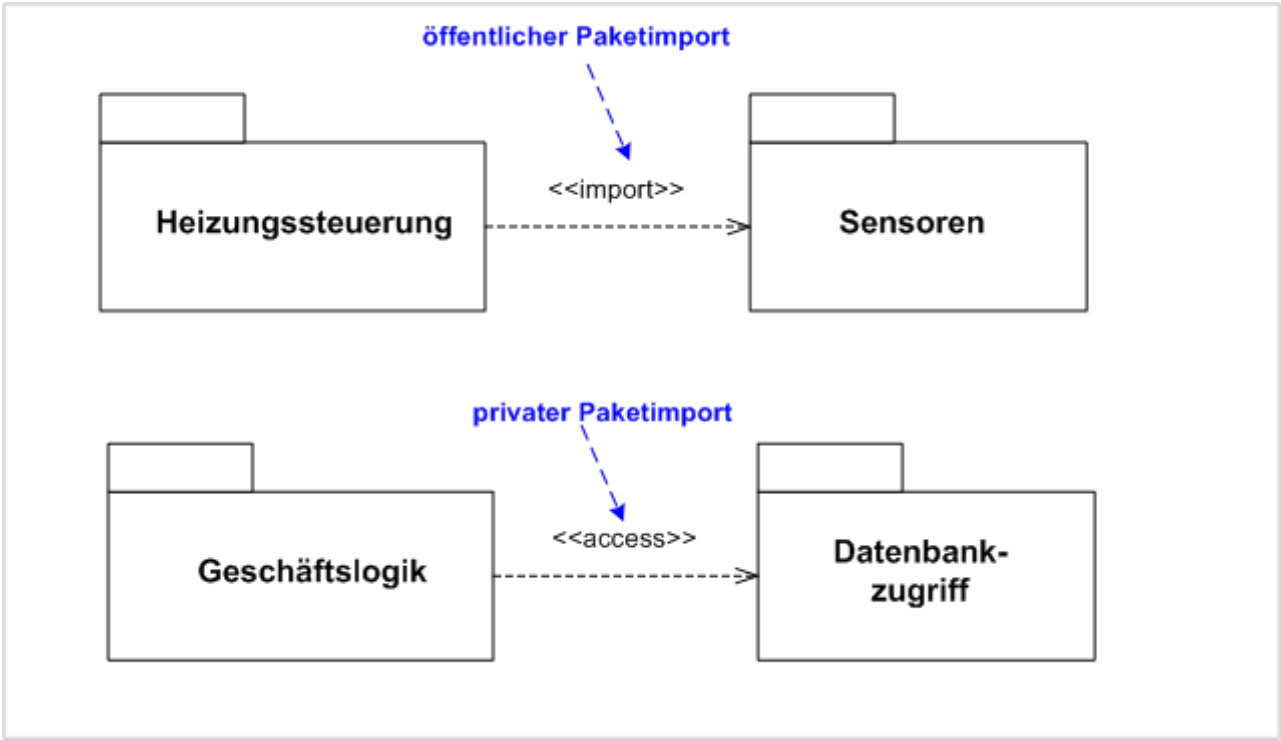
Composite Structure Diagram

Depicts the internal structure of a classifier (such as a class, component, or use case), including the interaction points of the classifier to other parts of the system.



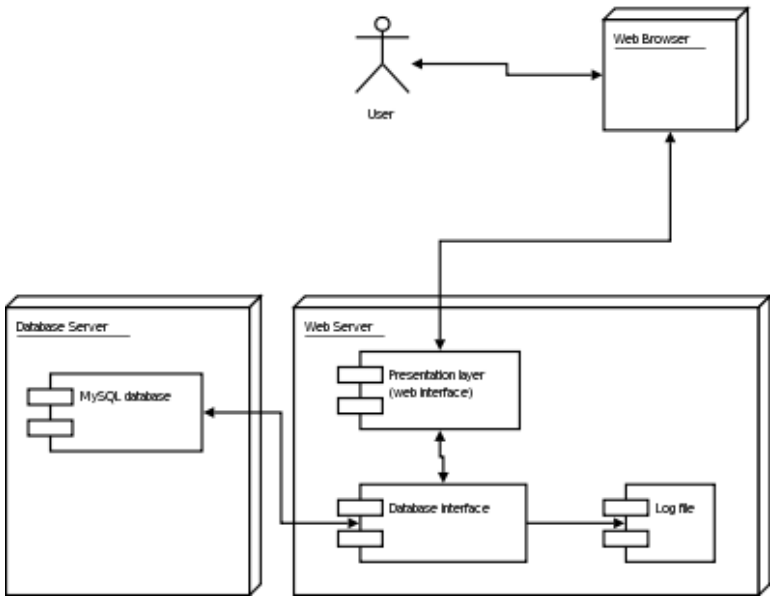
Package Diagram

Shows how model elements are organized into packages as well as the dependencies between packages.



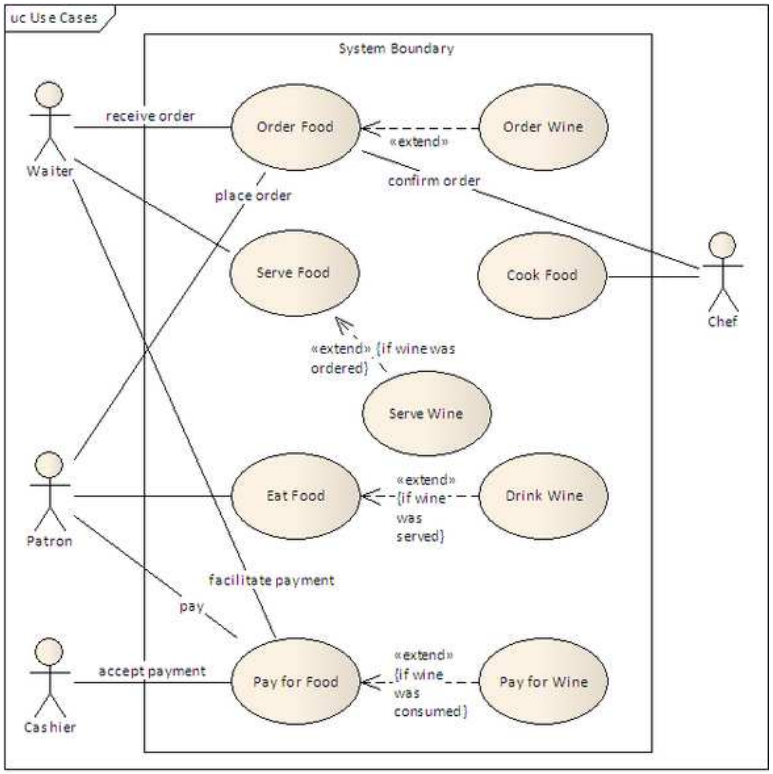
Deployment Diagram

Shows the execution architecture of systems. This includes nodes, either hardware or software execution environments, as well as the middleware connecting them.



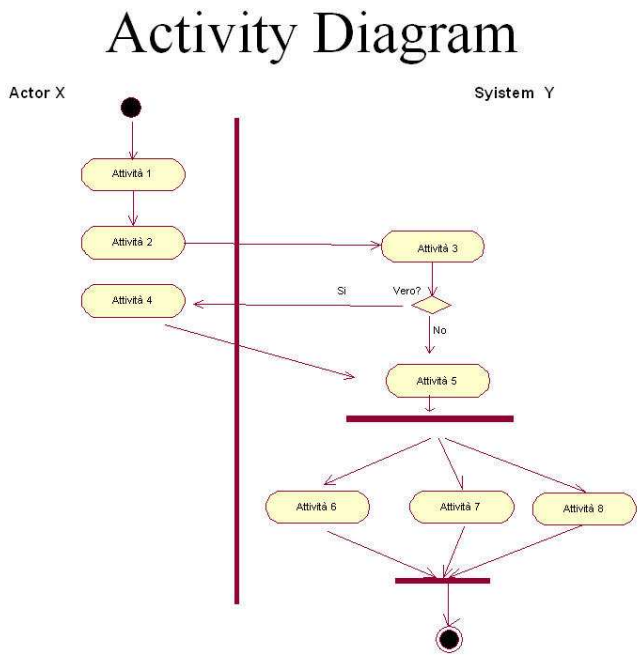
Use Case Diagram

Shows use cases, actors, and their interrelationships.



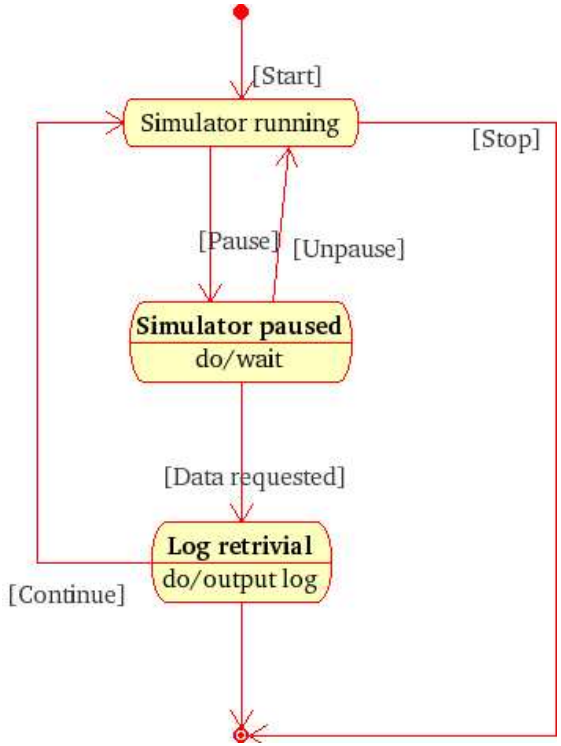
Activity Diagram

Depicts high-level business processes, including data flow, or to model the logic of complex logic within a system.



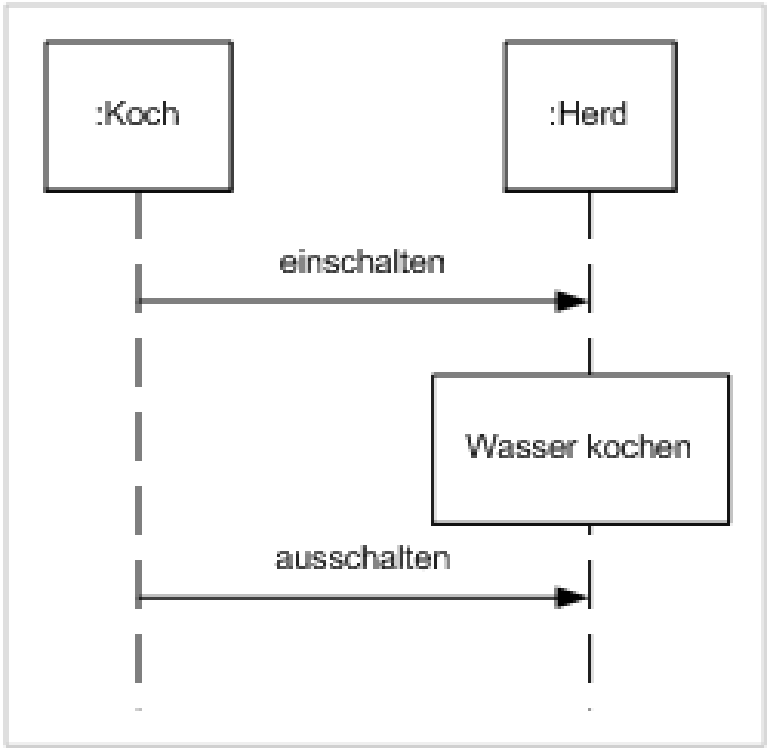
State Machine Diagram

Describes the states an object or interaction may be in, as well as the transitions between states. Formerly referred to as a state diagram, state chart diagram, or a state-transition diagram.



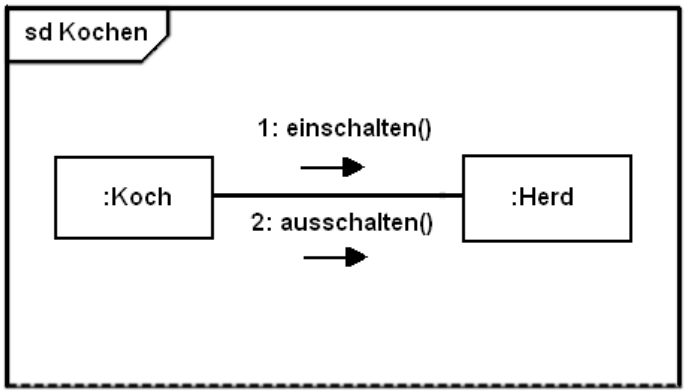
Sequence Diagram

Models the sequential logic, in effect the time ordering of messages between classifiers.



Communication Diagram

Shows instances of classes, their interrelationships, and the message flow between them. Communication diagrams typically focus on the structural organisation of objects that send and receive messages. Formerly called a Collaboration Diagram.

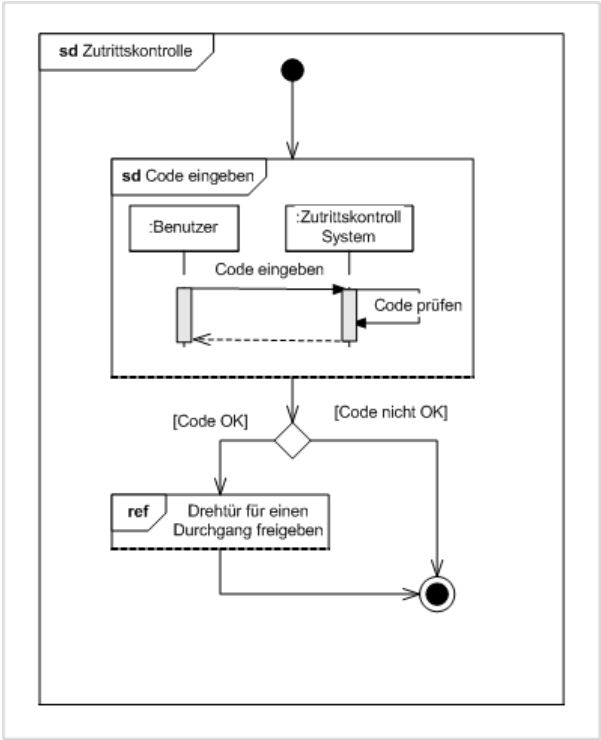


Timing Diagram

Depicts the change in state or condition of a classifier instance or role over time. Typically used to show the change in state of an object over time in response to external events.

Interaction Overview Diagram

A variant of an activity diagram which overviews the control flow within a system or business process. Each node/activity within the diagram can represent another interaction diagram.



Modelling

System development

- System development is model building
  - Complexity of a large project
    - A large number of components
    - A large amount of team work
  - Linguistic communication between teams or between team members is neither accurate nor reliable
  - Models are standard representations and they are accurate and reliable
  - Modelling is the process of developing a model
  - Various types of models for different purposes and stages in software development

Types of models

- Various types of models
  - Requirement model describes
    - Users' requirements
    - Functionality
  - Analysis model gives
    - System specifications
    - A robust and changeable structure and structured components
  - Design model presents
    - A refined structure to the current implementation environment
  - Implementation model documents
    - Details of how a design is implemented
  - Test model gives
    - Verification
    - Validation

Architecture

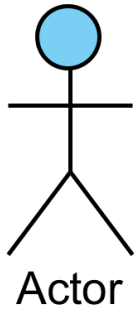
- Architecture of a modelling language
  - Model architecture is a set of modelling language, notation and modelling techniques.
  - A modelling language contains:
    - Syntax - how it looks
    - Semantics - what it means
    - Pragmatics - heuristics and rules of thumb for using it
- UML
  - Unified Modelling Language is commonly accepted
  - Idea first came from Ivar Jacocson in 1997

Requirements Model

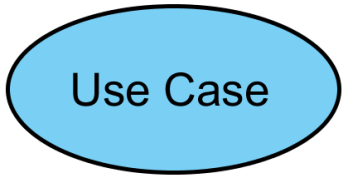
- Users' requirements in software engineering
  - A client/end-user's needs and expectations
  - Essential characteristics of the client/end-user's goal
  - They are purely the user's view of a system
  - Requirements should be problem-based and not describe solutions (Remember that no solution has yet been developed)
  - Requirements are often given in terms of what actually happens within a physical, chemical, biochemical, business, transportation,. . . process.
  - Requirements are modeled using Use Case diagrams

Use Case diagram - syntax

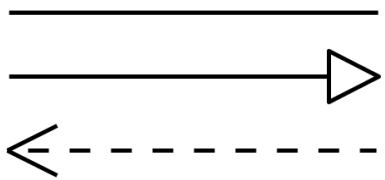
- Actor - interaction with a process often initialised by outsiders



- Use case - a series events taking place within a process and they are often triggered by an actor



- Relationship - information flow between an actor and a use case or between two use cases



Use Case diagram - pragmatics

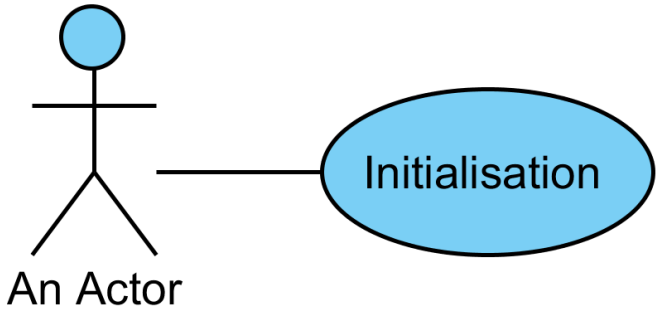
- Identify actors
  - What are entities outside of a process which trigger information exchange with the process?
  - Can we classify them?
- Identify use cases
  - What are the main tasks of each actor?
  - How should the process response to each actor?
  - A use case should link to a scenario representing what happens in the process in response to an actor
  - A use case should contain a complete course of events related to the scenario
- Add relationship
  - We need to pay extra attention on those between use cases

Use Case diagram - semantics

- An actor represents anything that is outside of a process being described and that needs to exchange information with the process
  - An individual person (e.g. an end-user, an engineer)
  - A group of people who play the same role (e.g. cashers in a bank)
  - An individual that can play different roles should be represented as several actors according to his role in a process (e.g. HoD, researcher, lecturer)
  - A machine
  - An object
  - ...

Use Case diagram - events

- A use case represents a special sequence of events triggered by an actor
  - Example: initialising a process through a menu

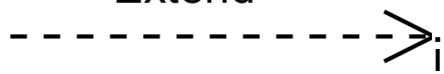


    - The actor triggers a sequence of events which take place in a process
  - A Use Case can contain the following events
    - display a list of variables
    - Accept values given by the actor and assigned to the variables
    - give a warning signal if the values given are out of range
    - give acknowledgement

Use Case diagram - relationship

- Relationship represents information exchange between an actor and a Use Case or between two Use Cases
  - Different types of relationship exiting between an actor and a use case and between two use cases:
    - Ordinary relationship showing the simple information exchanges between an actor and a use case or between two use cases
    - The symbol ———————— is used to represent this type of relationship

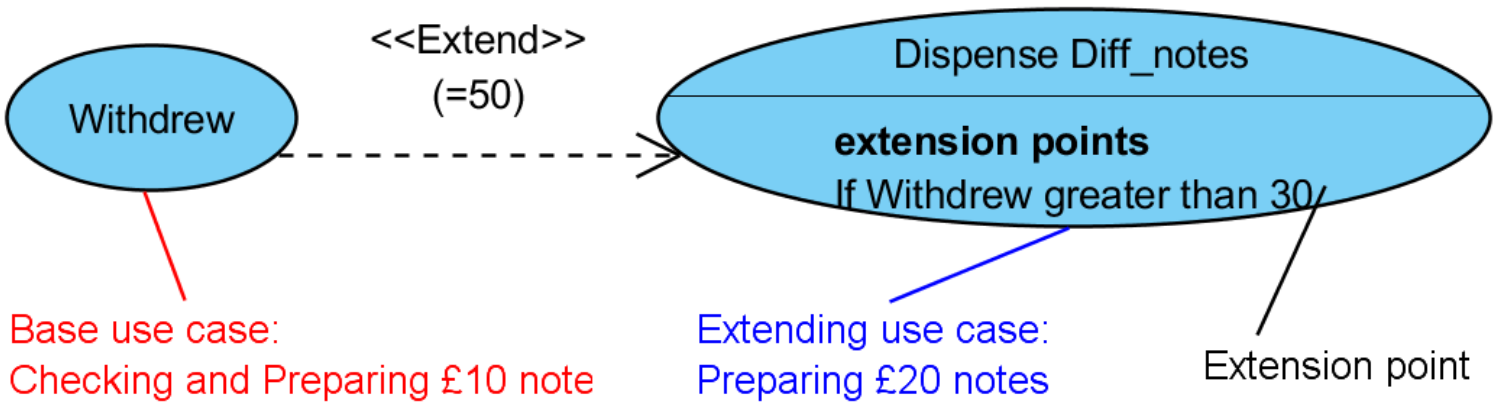
Use Case diagram - relationship

- An Extend relationship exists between two similar Use Cases where the second one has some extra activities, that is, the activities of the first Use Case are extended in the second one
  - The first Use Case sends information to the second Use Case to invoke the extra activities
  - **Terms**:
    1) *first Use Case*: base Use Case;
    2) *second Use Case*: extending Use Case (which has two sections, one is name and the other is the condition for the extra activities);
    3) *the condition*: extension point.
  - The Symbol 

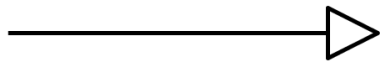
is used to represent this type of relationship

Use Case diagram - example

- Think about a cash machine (atm). You can withdrew £10 and £20 notes from your account through a cash machine. The cash machine will check your pin, account balance, etc. If you withdrew £20, the machine will simply dispense two £10 notes. If you withdrew, say, £50, the machine will dispense two £20s and one £10. This means that the machine check the withdrew amount and then decide what notes will be given out. This can be expressed as shown in the following.

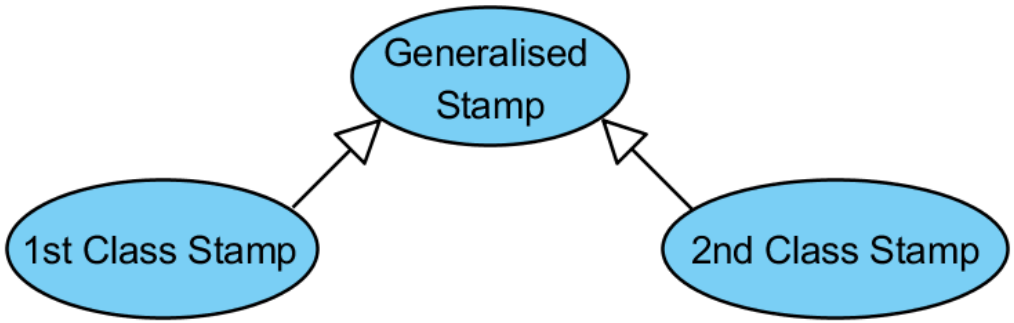


Use Case diagram - relationship

- A Generalisation relationship also exists between two use cases
  - A group of use cases may have some common activities
  - A generalised use case contains those common activities extracted from the group use cases
  - What left to the group of use cases are the specific activities
  - The Symbol  is used to represent this type of relationship

Use Case diagram - example

- Think about auto-stamp machine where you can buy 1st class and 2nd class stamps. Consider two Use Case, one is for 1st class stamps and the other is for 2nd class stamps. Both need to check coins you insert into the machine, calculate balance, dispense changes but dispense different stamps.
- The common activities are checking coins, calculating balance and giving change.
- Special activities are dispensing 1st class stamps and dispensing 2nd class stamps.
- Extracting the common activities from the two use cases and placing them into a new use case forms a generalised use case



SUMMARY

Key Terms