UML

Literature

- Alhir: UML in a Nuthshell, O'Reilly
- Bennett, McRobb, Farmer: Object-Oriented Systems Analysis and Design using UML, McGraw-Hill
- Booch, Rumbaugh,Jacobson: The Unified Modeling Language, User Guide, Addison-Wesley
- Buschmann, Meunier, Rohnert, Sommerlad, Stal: A system of Patters – Pattern-Oriented Software Architecture
- Gamma, Helm, Johnson, Vlissides: Design Patterns – Elements of Reusable Object-Oriented Software, Addison Wesley, 1994
- Larman: Applying UML and Patterns, Prentice Hall PTR
- Maciaszek: Requirements Analysis and System Design, Addison-Wesley
- Rumbaugh, Jacobson,Booch: The Unified Modeling Language, Reference Manual, Addison-Wesley
- Schach: Object-Oriented Software Engineering, McGraw-Hill
- Schmuller: Teach Yourself UML in 24 hours, SAMS

Modelling

Why do we model?

- The primary product of a software producer (individual, team, company) is good software that satisfies the evolving needs of its users and the business, everything else is secondary.
- Although everything else is secondary it is not irrelevant!
- Production of quality software is practically impossible without proper planning and design

Why do we model?

To be successful a software organisation has to consistently deploy quality software that:

- meets the needs of its users
- is developed in time
- is developed in a predictable fashion
- is developed with efficient and effective use of resources

What is needed?

In addition to the raw executable code at least the following artifacts are produced:

- Requirements specification
- Architecture description
- Analysis documentation
- Design documentation
- Source code
- Test cases, testing results
- Prototypes
- Release
- Agreements
- Project plans
- Meeting minutes
- User guides
- installation guides
- etc.

Great Questions

- How should the software development work be organised and performed?
- Do such processes and methods exist that quarantee that the desired results?

Model

- A model is a simplification of reality that still has the features relevant from the point of view of software development (at least we hope so J)
- We build models in order to have a better understanding on the system that we are developing.
- A model is an abstract representation of a system from a particular point of view.

Model

- Models aim at expressing the essentials of some aspect without giving unnecessary details.
- The purpose is to enable people involved in the development to think about problems and solutions without getting sidetracked.
- To be useful, a model must have a precise and well understood meaning.

Models

- Help us to visualize a system as it is or as we want it to be
- Permit us to specify the the structure or behaviour of a system
- Give templates that guide in constructing a system
- Document the decisions

J Remember J

- All models are wrong - but some are useful.
- Do not fall in love with one model!
- Choose the tools according to the task.
- Every project reaches a point after which ”better is worse than good enough”.

Modelling languages

- Modelling language is a tool for describing the various models produced during the development process
- Modelling language defines a collection of model elements
- Modern modelling languages are usually graphical, but could be text based (the model has always a representation in text form e.g. XML)

Modelling languages

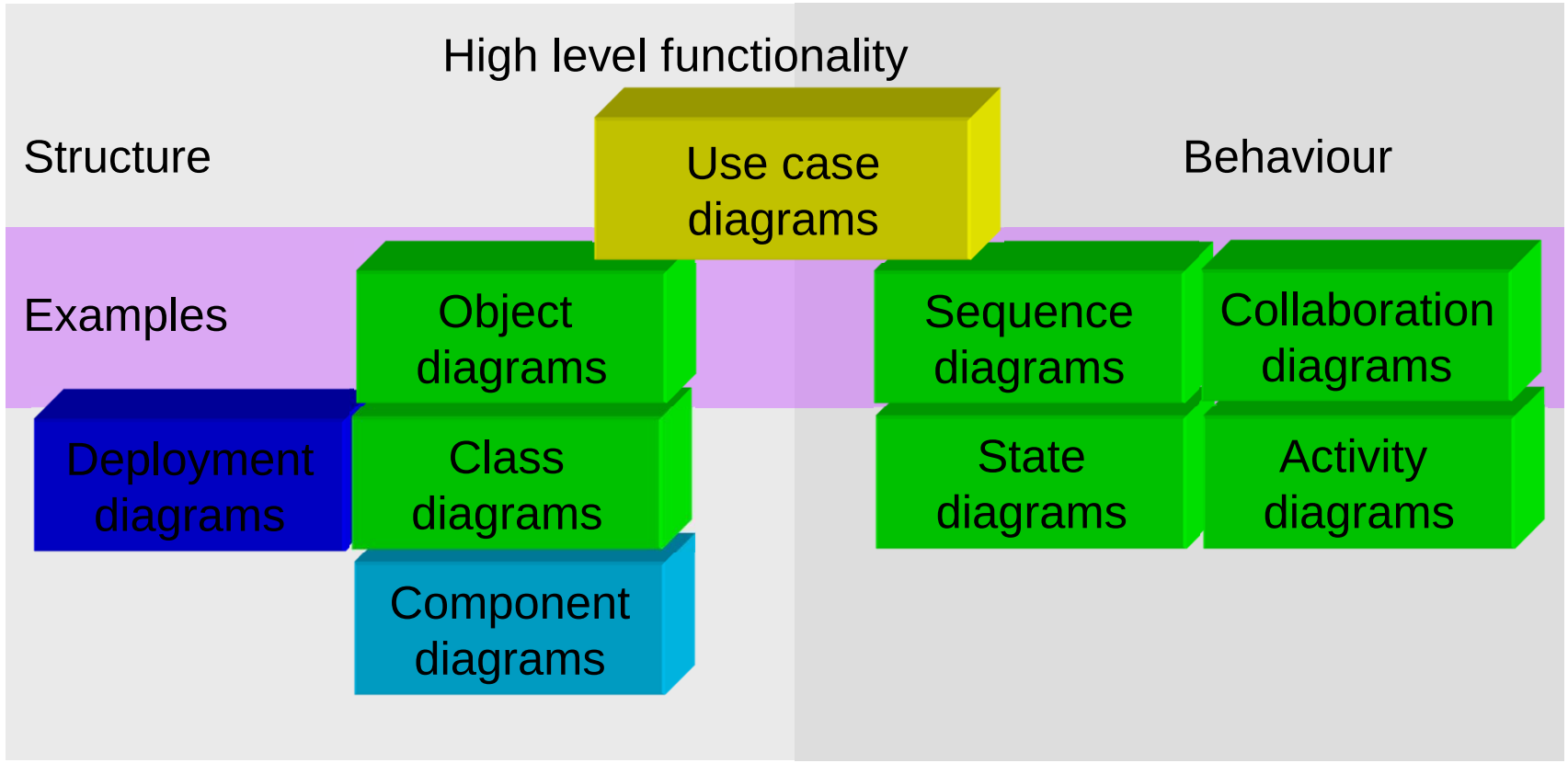
Modelling languages have the following aspects (as any language has)

- syntax: the rules that determine which expressions i.e. diagrams are legal
- semantics: the rules that determine what legal diagrams mean
- pragmatics: for what purpose are the diagrams used

UML - The Unified Modeling Language

- Based on several development lines of modelling languages that were drawn together in the middle of 1990's (the Three Amigos)
- De facto standard in (object-oriented) software modelling
- Several tools (and related recommended processes) available

UML diagrams



SOURCE:?

Requirements and Use Cases

Meeting the customer requirements

Source: http://www.agilesap.de/

System Behaviour

- The behaviour of the system under development is documented in a use case model that illustrates the system's intended functions (use cases), its surroundings (actors) and relationships between the use cases and actors, which together form use case diagrams
- Creation of the use case model starts with the identification of actors and principal use cases for the system
- More detailed information is added to the identified use cases, and additional use cases (and actors) are added on an as-needed basis.

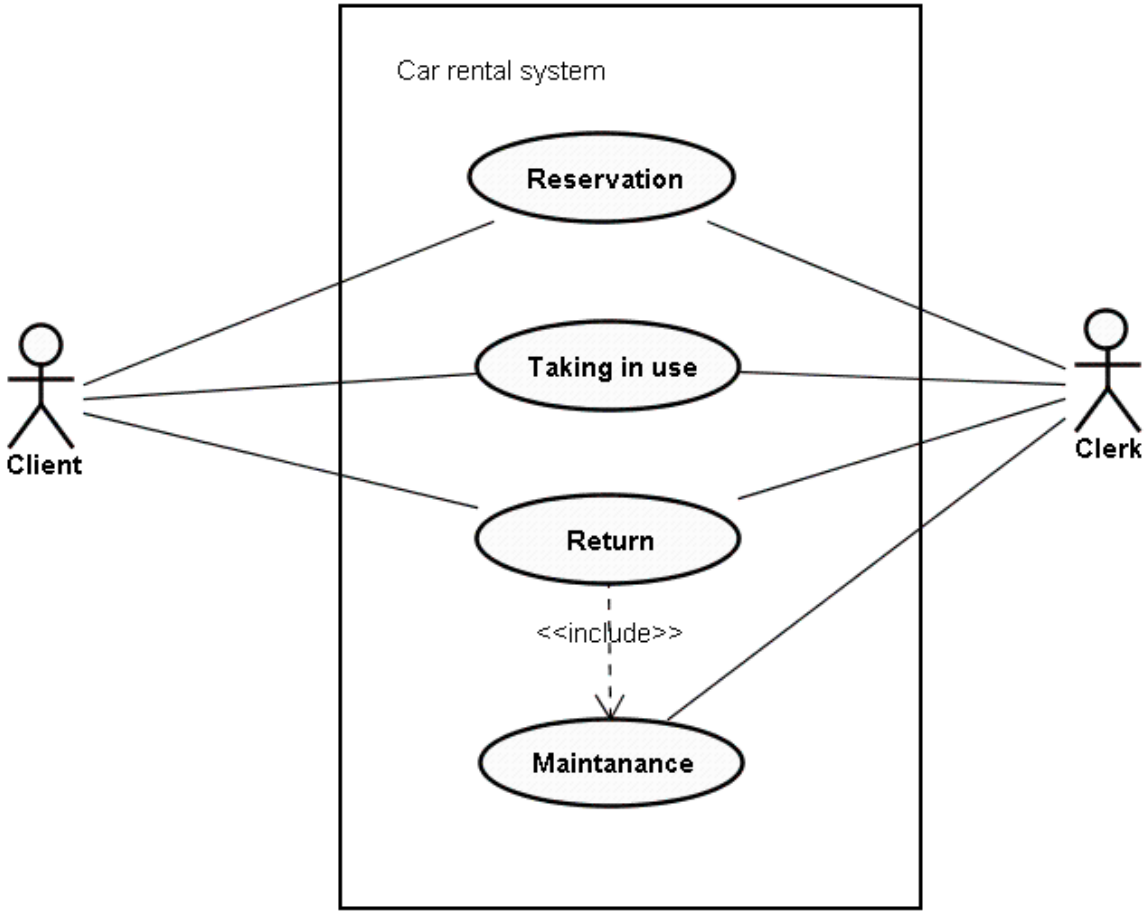
Use Case Diagram

actor, typically a user of the system, anyone or anything causing the system to start a function

use case, function of the system from actor's point of view

relationship between the actor and the use case

Use Case Diagram - Example



Actors

- Actors are not part of the system
- Actors represent anyone or anything that must interact with the system.
- An actor may
  - only input information to the system
  - only receive information from the system
  - input and receive information to and from the system

How to identify the actors

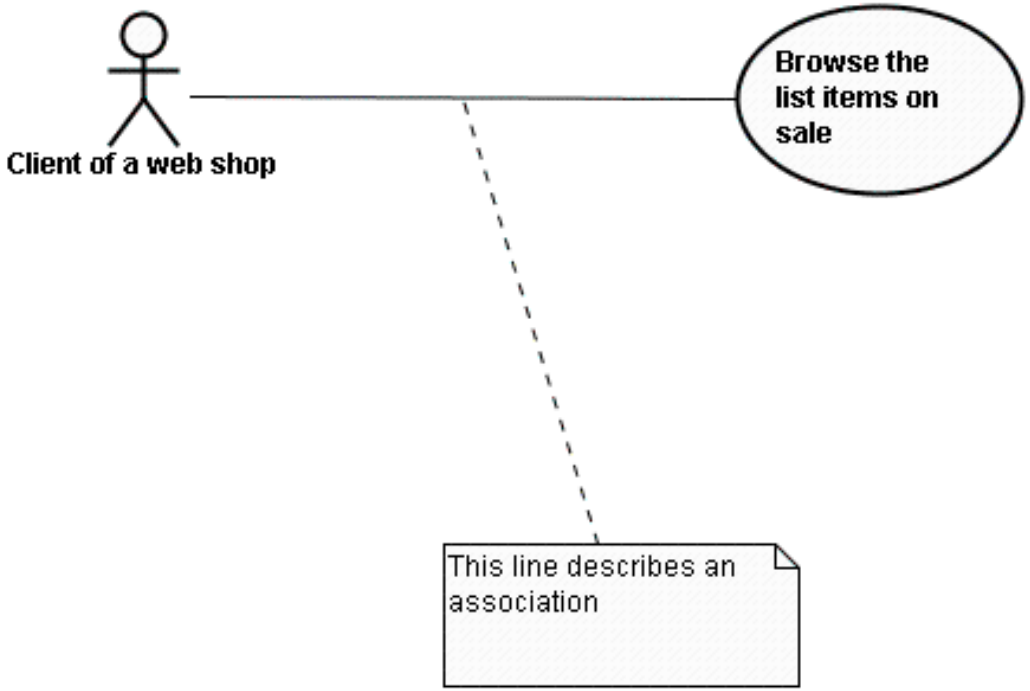
- Who is interested in a certain requirement?
- Where in the organisation is the system used?
- Who will benefit from the use of the system?
- Who will supply the system with this information, use this information, and remove this information?
- Who will support and maintain the system?
- Does the system use an external resource?
- Does one person play several different roles?
- Do several people play the same role?
- Does the system interact with a legacy system?

Use Cases

- Use cases model communication between actor(s) and the system
- Use cases represent the functionality provided by the system: what services will be provided to an actor by the system
- Use cases show *what* the system will offer for the user, they do not tell *how* the service is performed
- The system boundary lies between the actors and the use cases

Relationships between actors and use cases

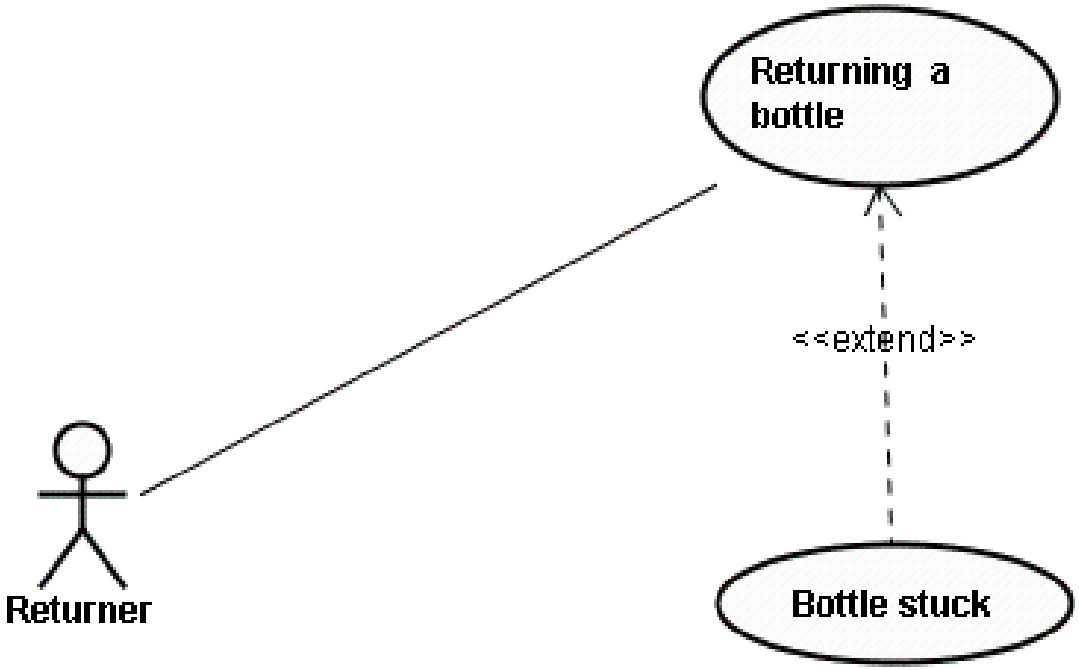
- Association describes the communication between the actor and the use case



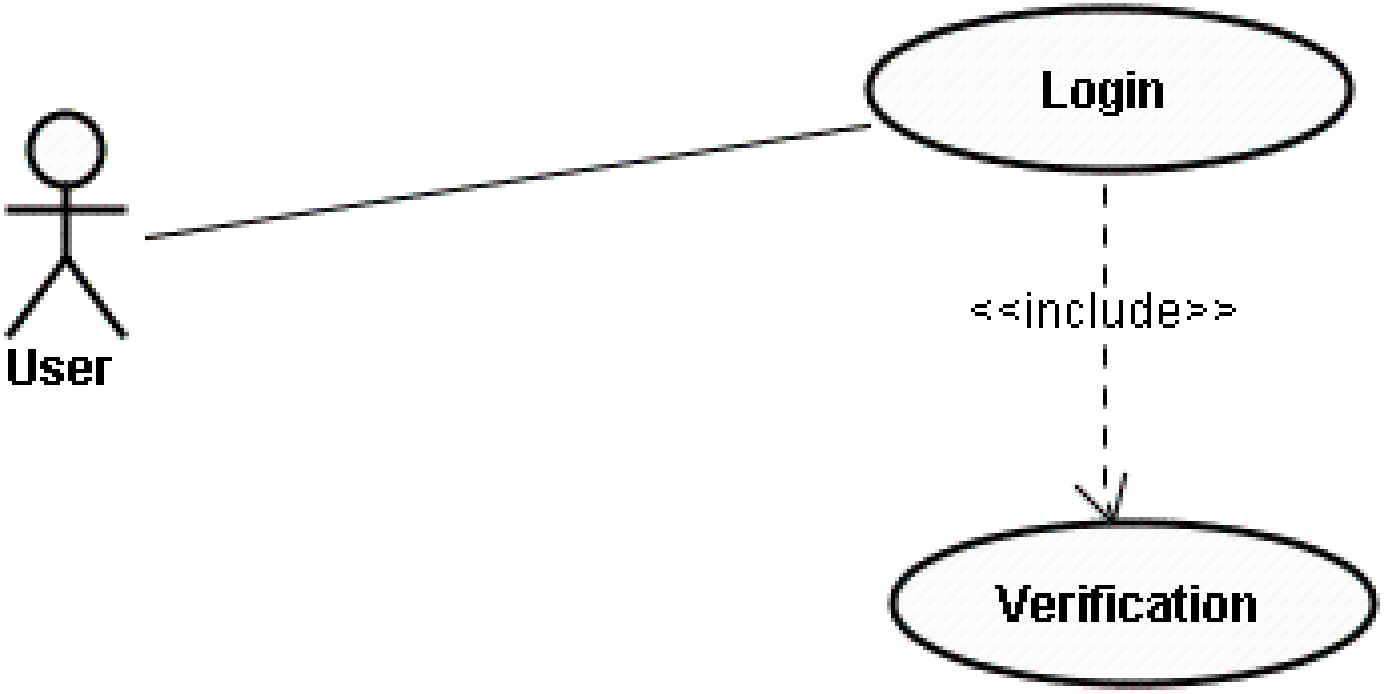
Relationships between use cases

- <<include>> type relatioship
  - Multiple use cases may share pieces of the same functionality. This functionality is placed in a separate use case rather than documenting it in every use case that uses it
- <<extend>> type relatioships
  - Optional behaviour
  - Behaviour that is only run under certain conditions, such as triggering an alarm
  - Several different flows which may be run based on actor selection
- Generalisation
  - One use case is a generalisation of the other

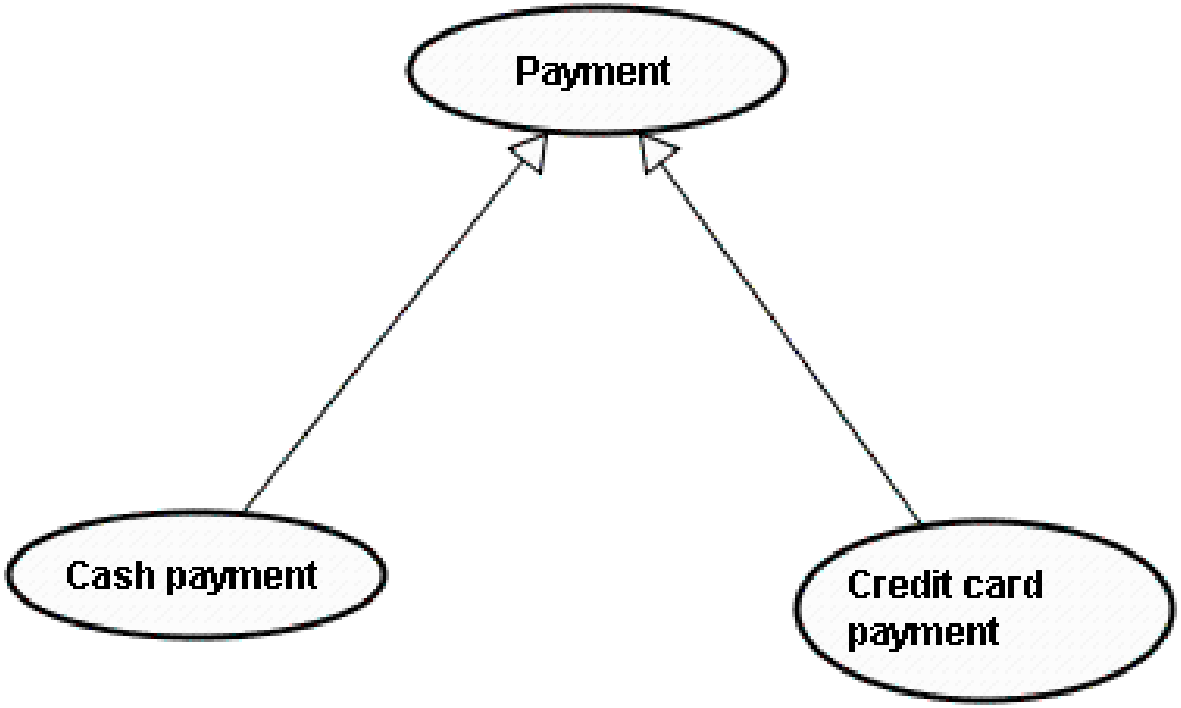
<<extend>>



<<include>>

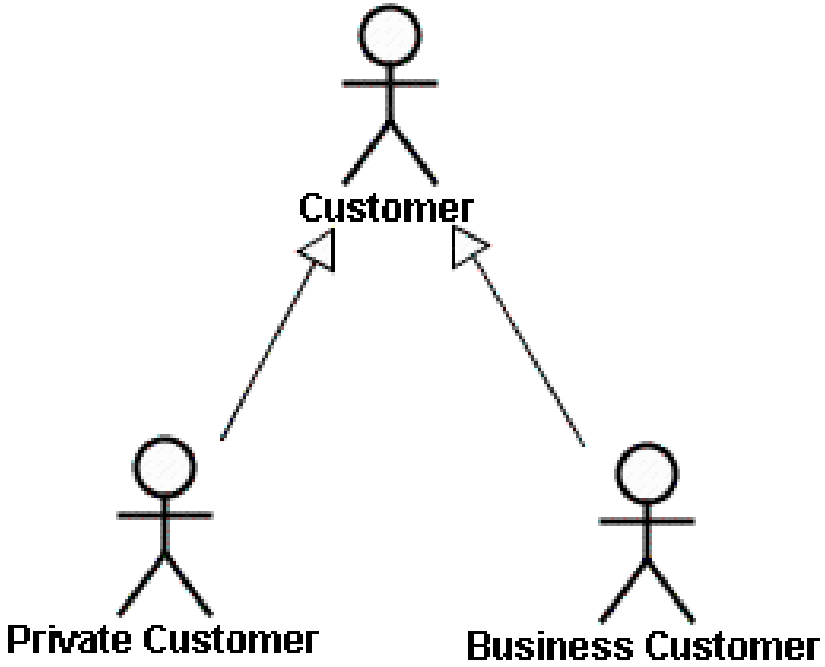


Use case generalisation



Relationships between actors

- Generalisation



Use Case Templates

- In addition to the use case diagram more detailed information is needed to fully document the use case.
- UML does not give tools for this pupose, but different templates can be used.
- A sample template and its detailed description can be found in http://www.technosolutions.com/use_case_template.html

Sample Use Case Template

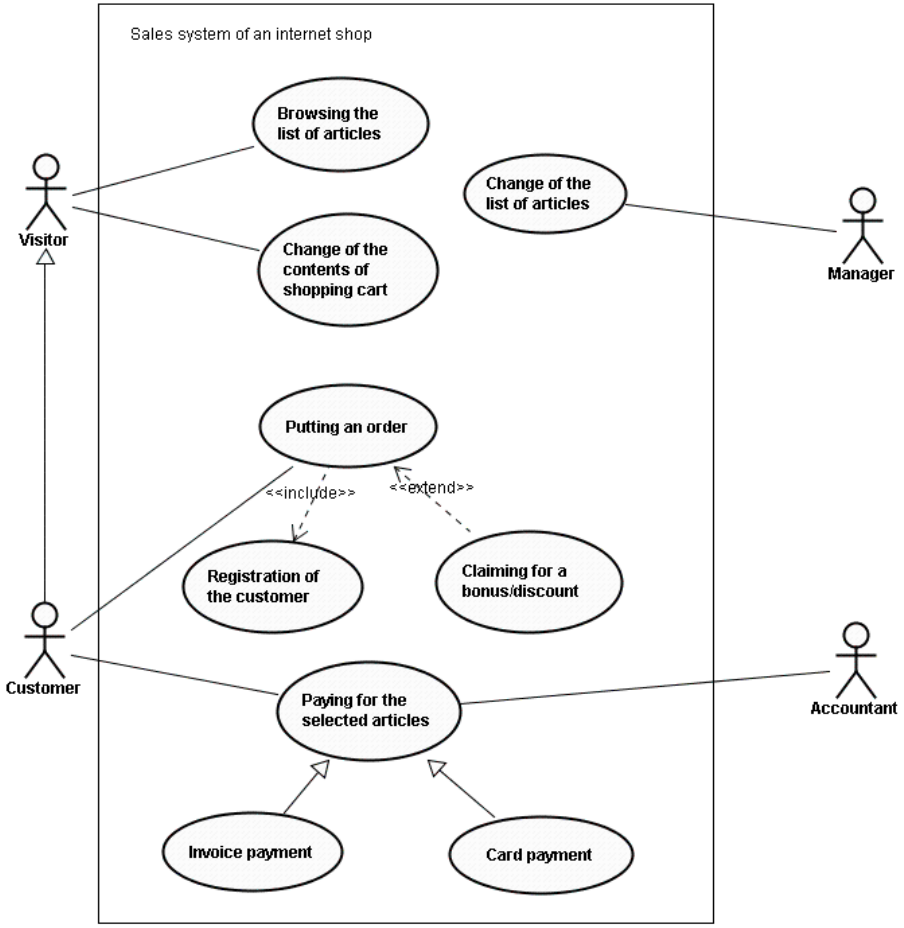
Use Case	*use case identifier and reference number and modification history*
Description	*goal to be achieved by use case and sources for requirement*
Actors	*list of actors involved in use case*
Assumptions	*Conditions that must be true for use case to terminate successfully*
Steps	*Interactions between actors and system that are necessary to achieve goal*
Variations *(optional)*	*Any variations in the steps of a use case*
Non-Functional *(optional)*	*List of non-functional requirements that the use case must meet.*
Issues	*List of issues that remain to be resolved*

User Stories

- User stories are one way to describe requirements:
  - "As a *<role>,* I want *<goal/desire>* so that *<benefit>*"

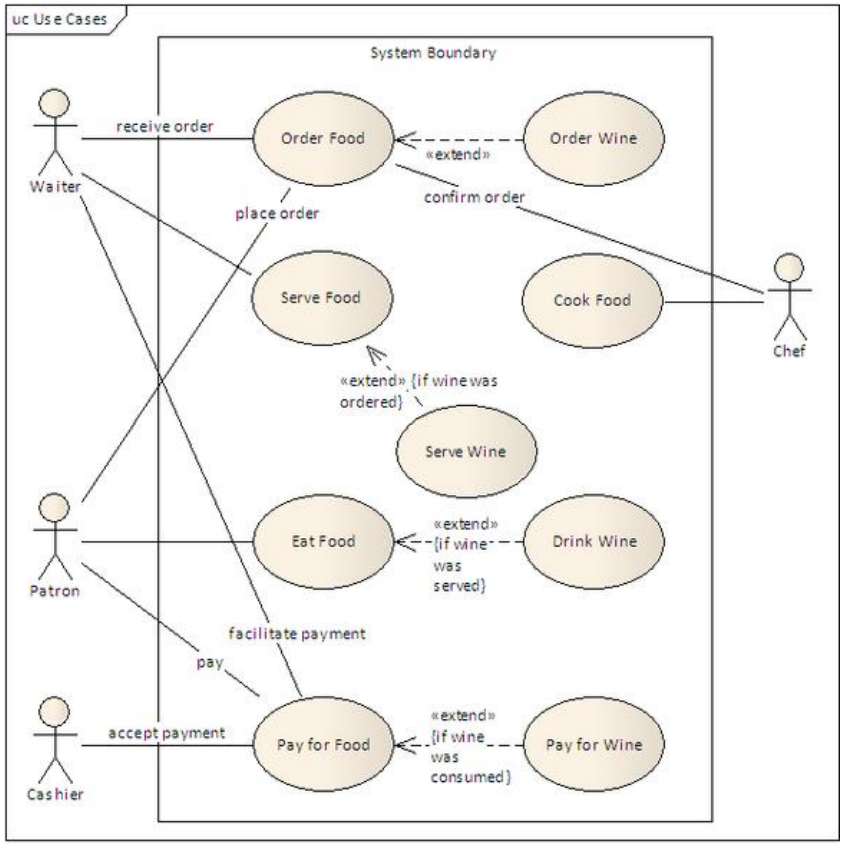
  or
  - "As *<who> <when> <where>,* I *<what>* because *<why>."*
- *E.g.* "As a user of the billing system, I want to search for my customers by their first and last names."

Example – Sales system of an Internet shop



UML (C) Olli Hämäläinen

Example – Restaurant Model



Source: http://en.wikipedia.org/wiki/File:Restaurant_Model.png

Typical outcomes of requirements modelling

- Use case model
- Requirements list
- Interface prototypes
- Initial system architecture
- Glossary