Department Of Computer Science

Lecture 09

Introduction Object-Oriented Software Engineering and UML Diagrams

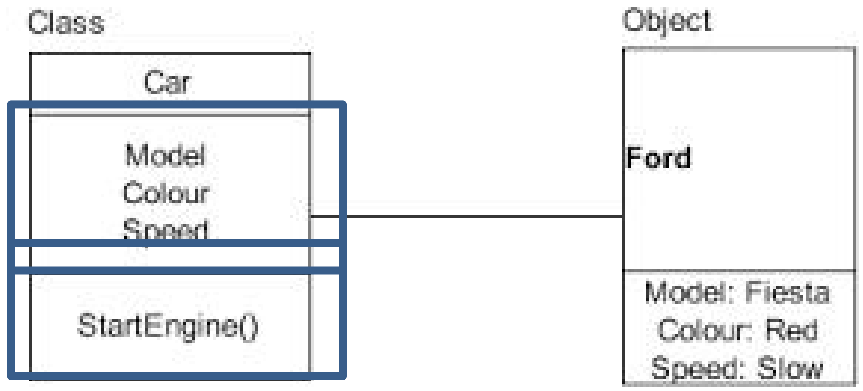
Topics covered

- Context Models
- Process Models
- Interaction Models
- Structural Models
- Behavioral models
- Data models

Object Oriented Software Engineering (OOSE)

- OOSE as a sub-domain of SE focuses on exploiting objects (instantiated from classes) to ***design***, ***develop***, ***test*** and ***deploy*** the software-intensive systems.

- Object
  - encapsulates both data (attributes) and data manipulation functions (called methods, operations, and services)
- Class
  - generalised description (template or pattern) that describes a ***collection of similar objects***
- Superclass
  - a collection of objects
- Subclass
  - an instance of a class



The Object-Oriented Paradigm

The structured paradigm was successful initially

- It started to fail with larger products (> 50,000 Lines of Code (LOC))

Post-delivery maintenance problems (today, 70 to 80 percent of total effort)

Reason: Structured methods are

- Action oriented (e.g., finite state machines, data flow diagrams); or
- Data oriented (e.g., entity-relationship diagrams, Jackson's method);
- But not both

The Object-Oriented Paradigm

Both data and actions are of equal importance

Object:

- A software component that incorporates both data and the actions that are performed on that data

Example:

- Bank account
  - Data: *account balance*
  - Actions: *deposit*, *withdraw*, *determine balance*

Strengths of the Object-Oriented Paradigm (contd)

Well-designed objects are independent units

- Everything that relates to the real-world object being modeled is in the **object** — *encapsulation*
- Communication is by sending *messages*
- This independence is enhanced by *responsibility-driven design*

Strengths of the Object-Oriented Paradigm (contd)

A classical product conceptually consists of a single unit (although it is implemented as a set of modules)

- The object-oriented paradigm **reduces complexity** because the product generally consists of **independent units**

The object-oriented paradigm **promotes reuse**

- Objects are independent entities

Objects and object classes

- Objects are entities in a software system which represent instances of real-world and system entities.
- Object classes are **templates for objects**. They may be used to create objects.
- Object classes may **inherit attributes** and services from other object classes.

Objects and object classes

An object is an entity that has a state and a defined set of operations which operate on that state. The state is represented as a set of object attributes. The operations associated with the object provide services to other objects (clients) which request these services when some computation is required.

Objects are created according to some **object class** definition. An object class definition serves as a template for objects. It includes declarations of all the attributes and services which should be associated with an object of that class.

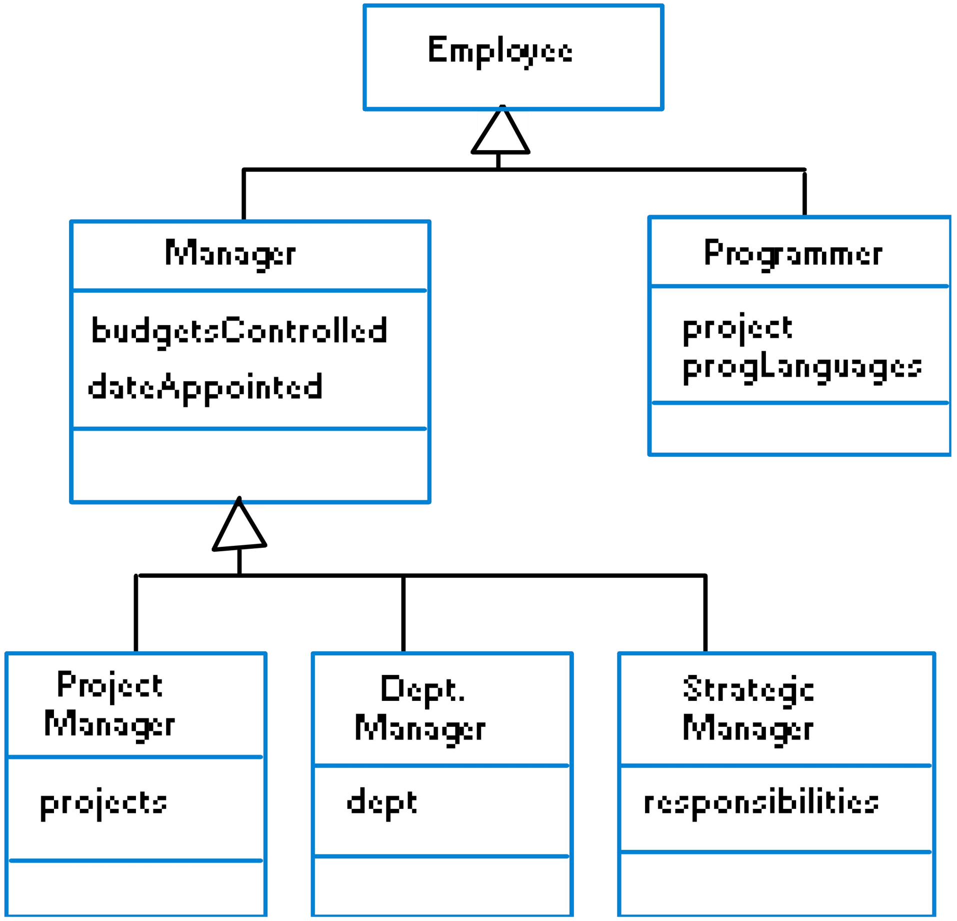
Object communication

- Conceptually, objects communicate by message passing.
- Messages
  - The name of the service requested by the calling object;
  - Copies of the information required to execute the service and the name of a holder for the result of the service.
- In practice, messages are often implemented by procedure/method/operation calls
  - Name = procedure/method/operation name;
  - Information = parameter list.

Generalisation and inheritance

- Objects are members of classes that define attribute types and operations.
- Classes may be arranged in a class hierarchy where one class (a super-class) is a generalisation of one or more other classes (sub-classes).
- A sub-class inherits the attributes and operations from its super class and may add new methods or attributes of its own.
- Generalisation in the UML is implemented as inheritance in OO programming languages.

A generalisation hierarchy



Problems with inheritance

- Object classes are not self-contained. they cannot be understood without reference to their super-classes.
- Designers have a tendency to reuse the inheritance graph created during analysis. Can lead to significant inefficiency.
- The inheritance graphs of analysis, design and implementation have different functions and should be separately maintained.

The Object-Oriented Paradigm

What is UML?

- Standard language for **specifying**, **visualizing**, **constructing**, and **documenting** the artifacts of software systems, business modeling and other non-software systems.
- The UML is a very important part of developing object **oriented software** and the software development process.
- The UML uses mostly graphical notations to express the design of software projects.
- Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

Origins of UML

- In the 1980s, object-oriented programming moved from research labs into the real world
- Smalltalk and C++ became popular languages and various people started thinking about object-oriented graphical design languages
- Between 1988 and 1992, the key authors were Booch, Coad, Jacobson, Odell, Rumbaugh, Shlaer, Mellor, and Wirfs-Brock
  - Each author was informally leading a group of practitioners who liked those ideas
  - The same basic OO concepts would reappear in very different notations, causing confusion with clients
- When Jim Rumbaugh left GE to join Grady Booch at Rational, an alliance was formed and a critical mass of market share occurred
- In 1997, Rational released UML 1.0

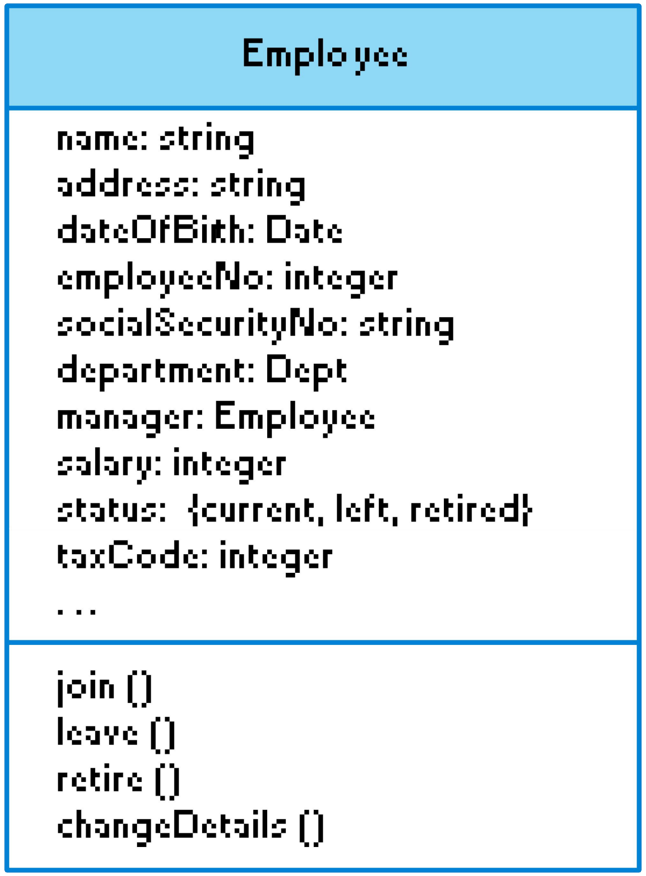
Origins of UML (continued)

- Consists of a family of graphical notations that help in describing and designing software systems
- Focuses particularly on software systems built using the object-oriented style
- Controlled by the Object Management Group, which is an open consortium of companies
- Comes from the unification of many OO graphical modeling languages that thrived in the 1980s and early 1990s

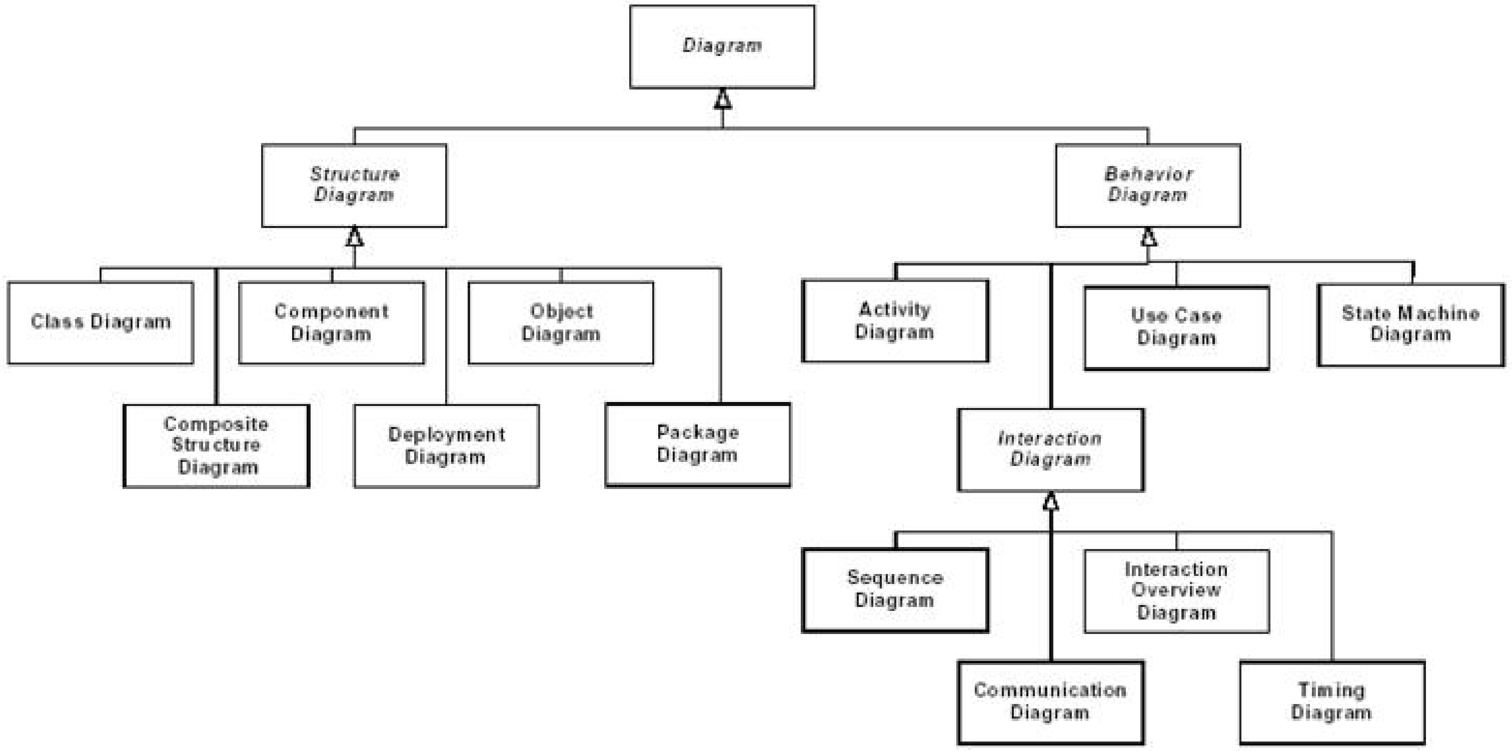
The Unified Modeling Language

- Several different notations for describing object-oriented designs were proposed in the 1980s and 1990s.
- The Unified Modeling Language is an integration of these notations.
- It describes notations for a number of different models that may be produced during **OO analysis and design**.
- It is now a ***de facto* standard** for **OO modelling**.

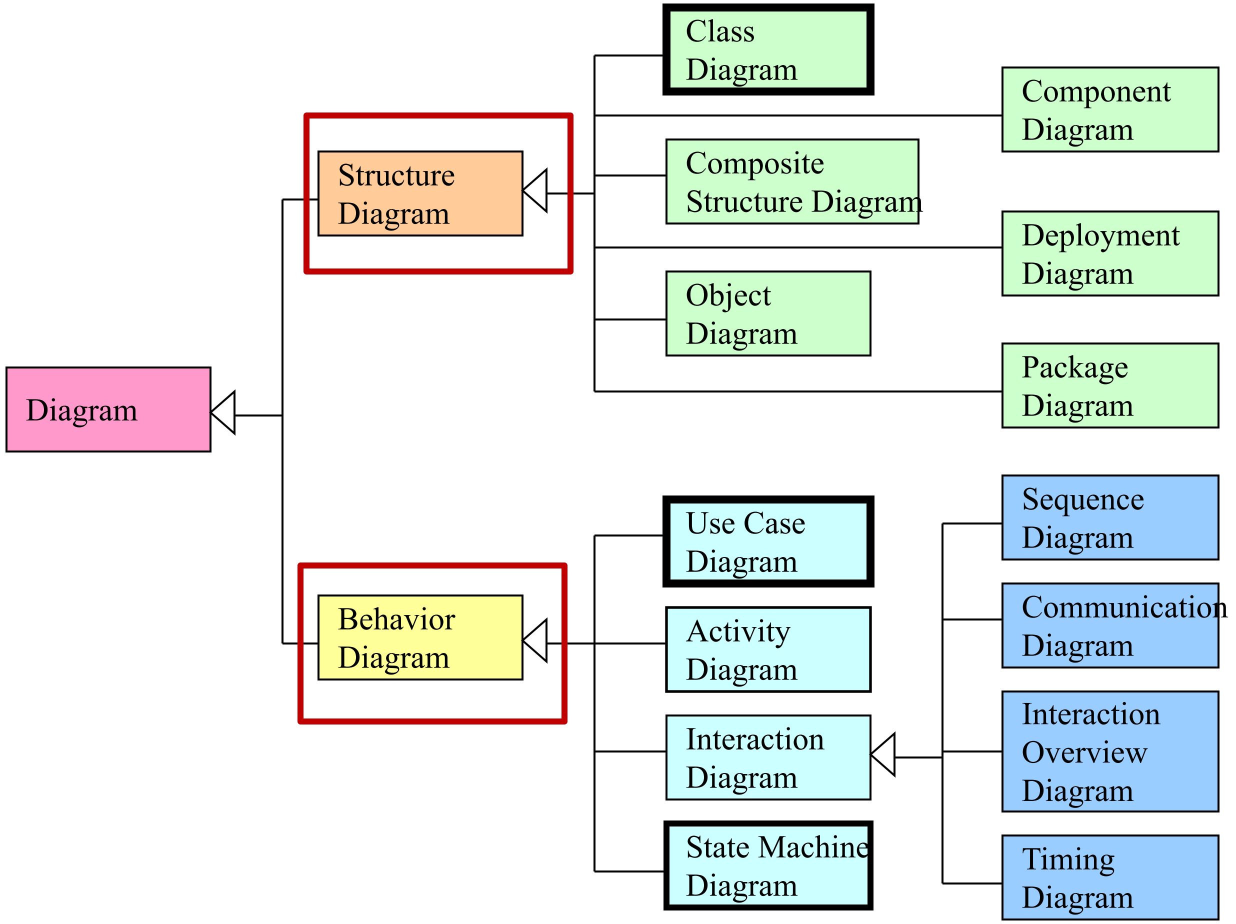
Employee object class (UML)



UML diagram hierarchy



Classification of Diagram Types



Overview of UML Diagrams

Structural

: element of spec. irrespective of time

- Class
- Component
- Deployment
- Object
- *Composite structure*
- *Package*

Behavioral

: behavioral features of a system / business process

- Activity
- State machine
- Use case
- *Interaction*

Interaction

: emphasize object interaction

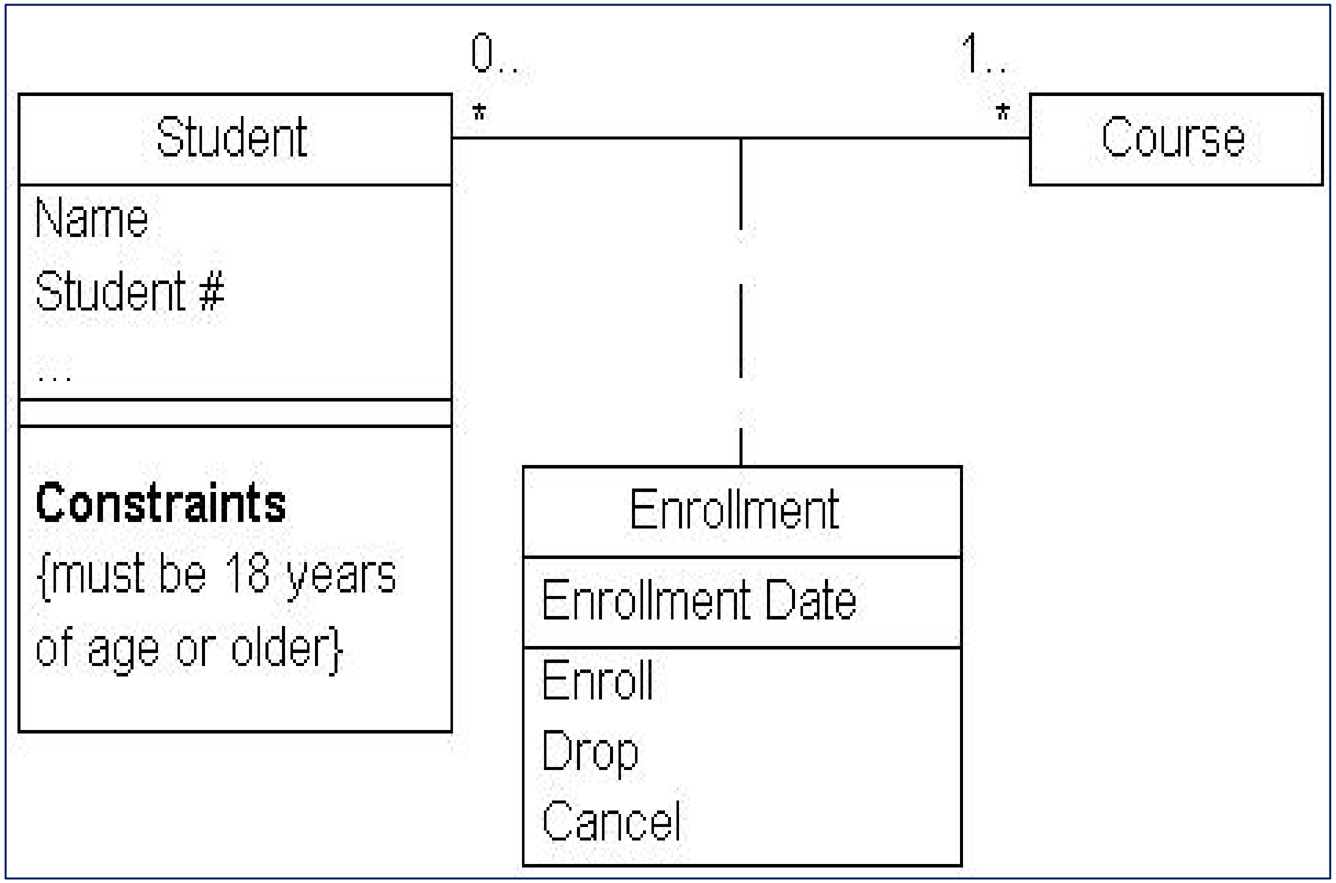
- Communication
- Sequence
- *Interaction overview*
- *Timing*

Class diagram

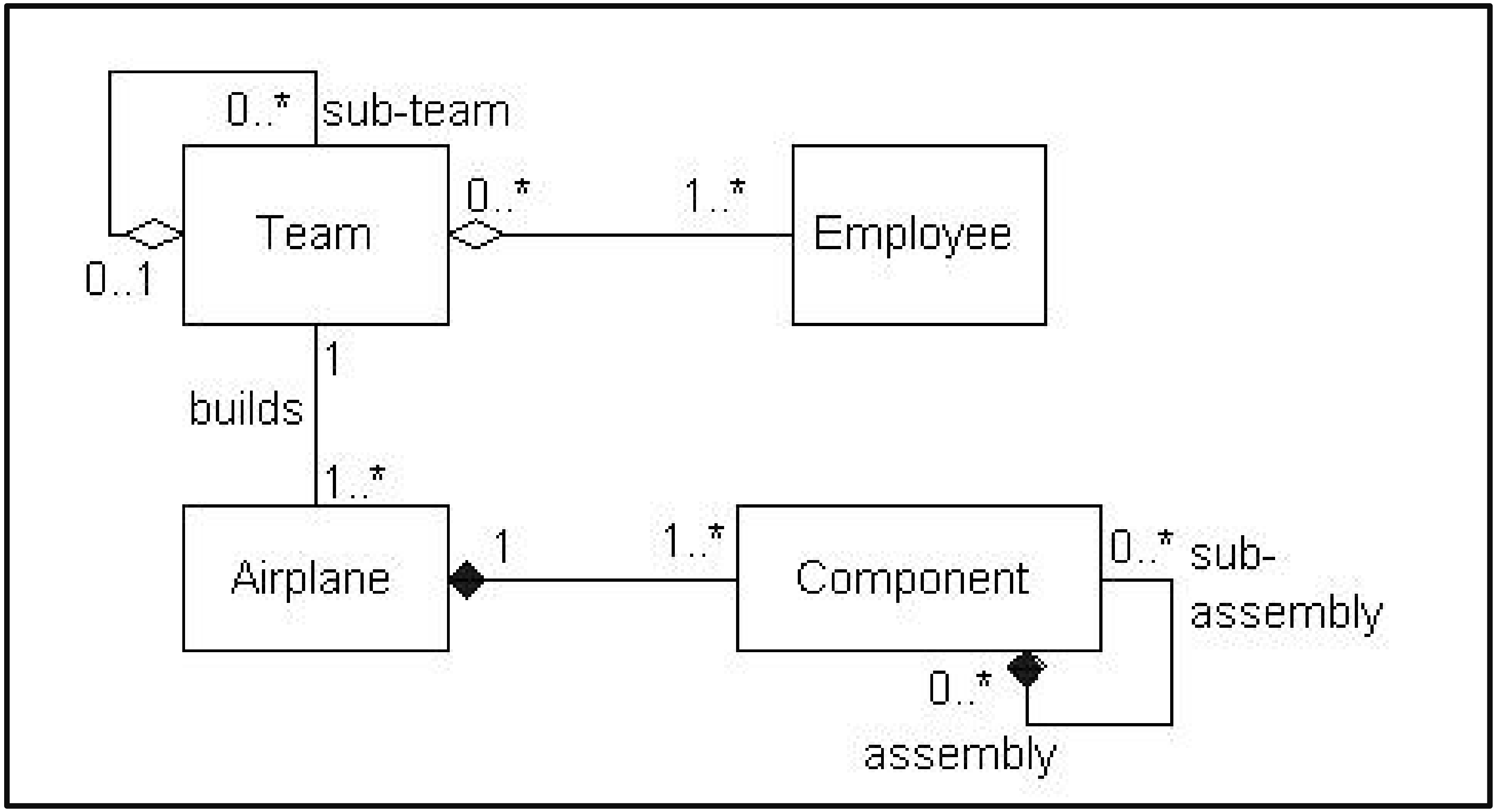
UML class diagrams show the classes of the system, their **inter-relationships**, **and the operations** and **attributes** of the classes

- Explore domain concepts in the form of a domain model
- Analyze requirements in the form of a conceptual/analysis model
- Depict the detailed design of object-oriented or object-based software

Class diagram



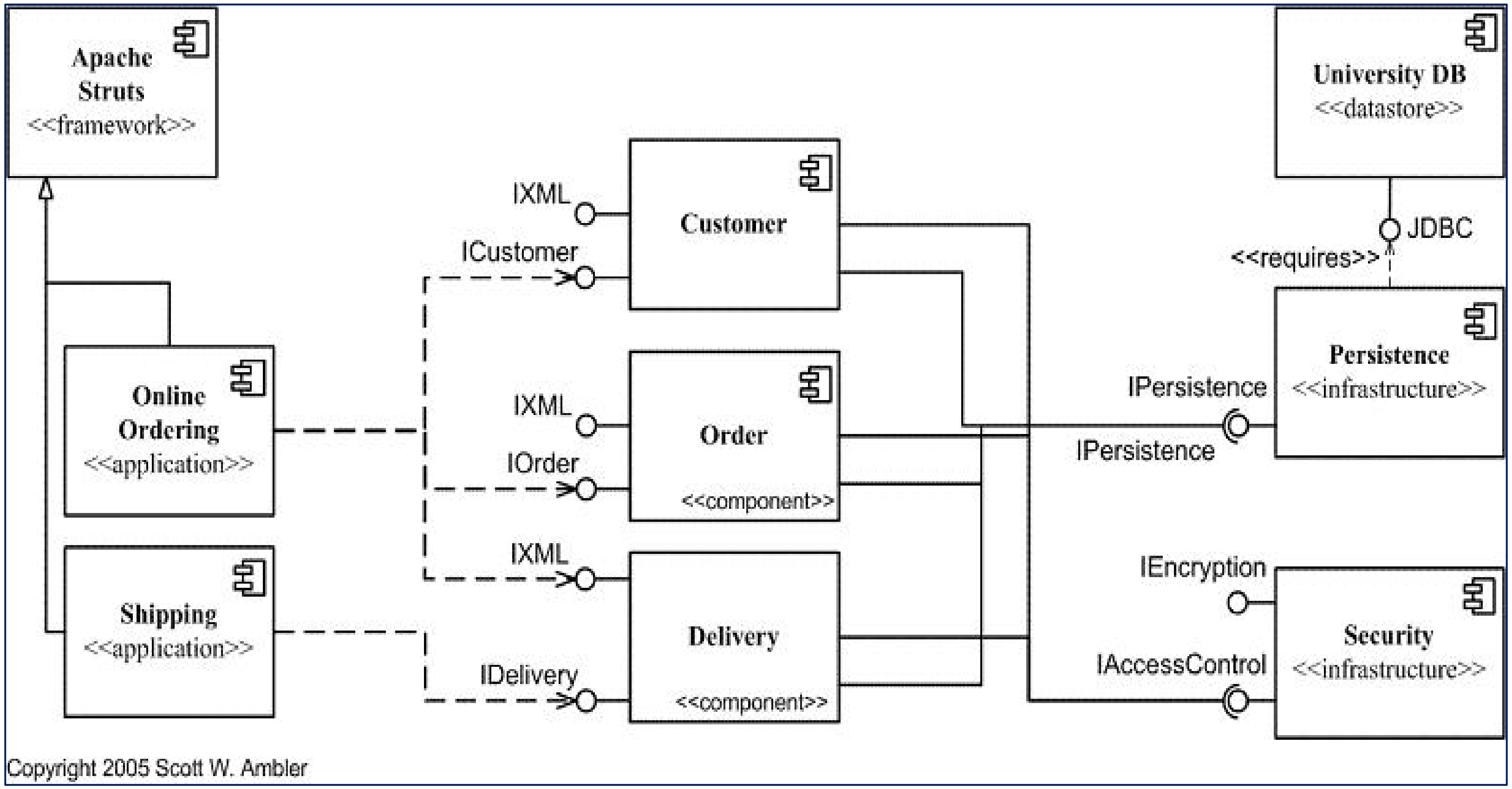
Class diagram



Component diagram

UML component diagrams shows the **dependencies** among **software components**, including the classifiers that specify them (for example implementation classes) and the artifacts that implement them; such as *source code files*, *binary code files*, *executable files*, *scripts and tables*.

Component diagram

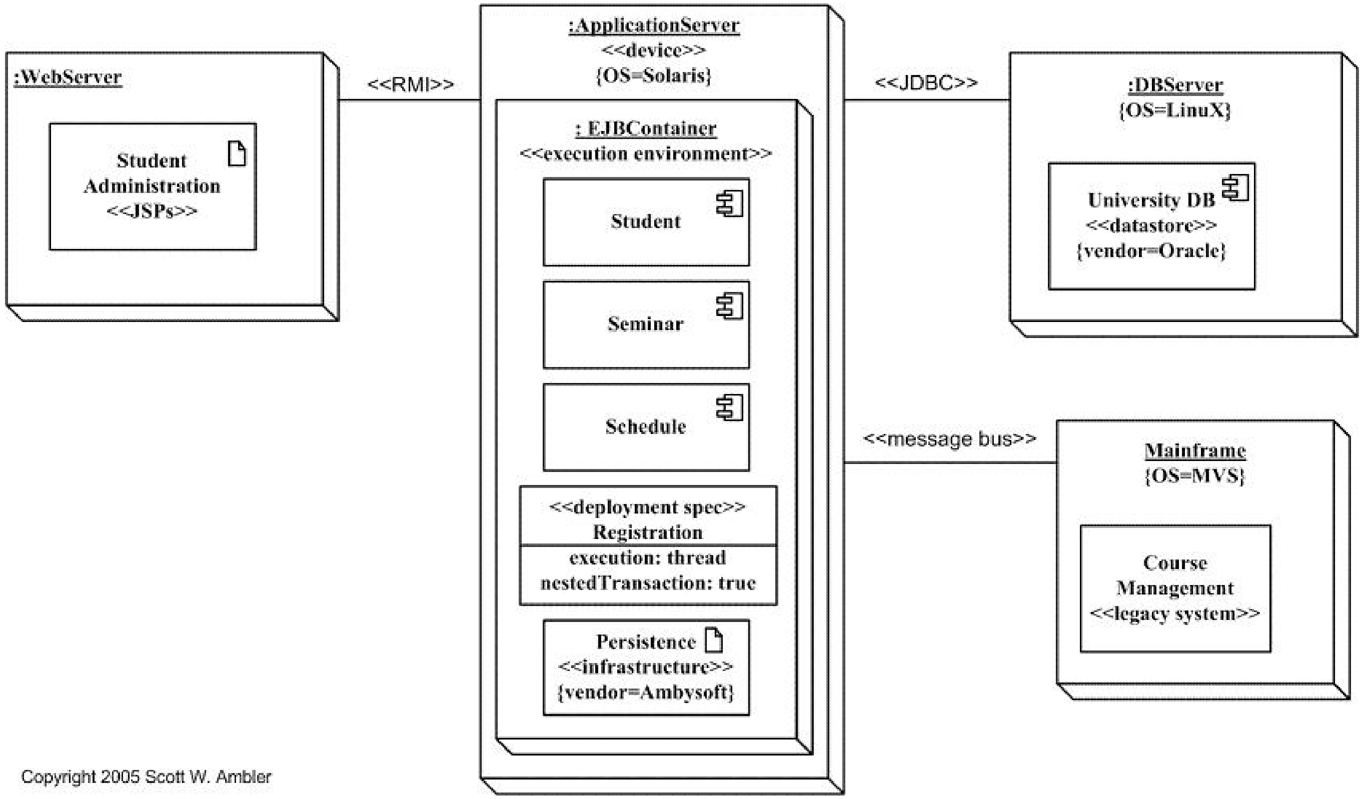


Deployment diagram

UML deployment diagram **depicts a static view** of the **run-time configuration** of **hardware nodes** and the **software components** that run on those nodes.

Deployment diagrams show the hardware for your system, the software that is installed on that hardware, and the middleware used to connect the disparate machines to one another.

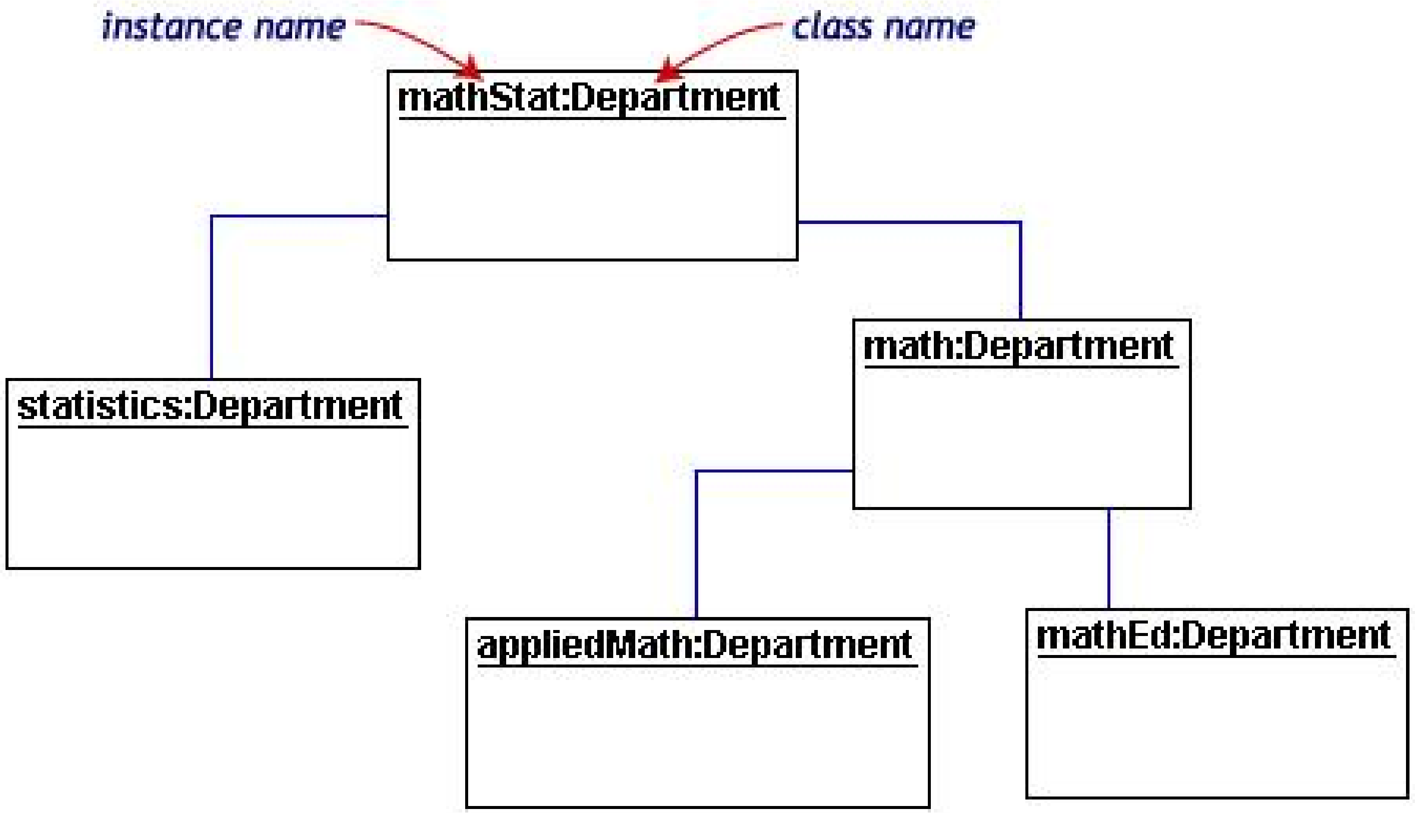
Deployment diagram



Object diagram

- UML 2 Object diagrams (instance diagrams), are useful for exploring real world examples of objects and the relationships between them.
- **It shows instances instead of classes.** They are useful for explaining small pieces with complicated relationships, especially recursive relationships.

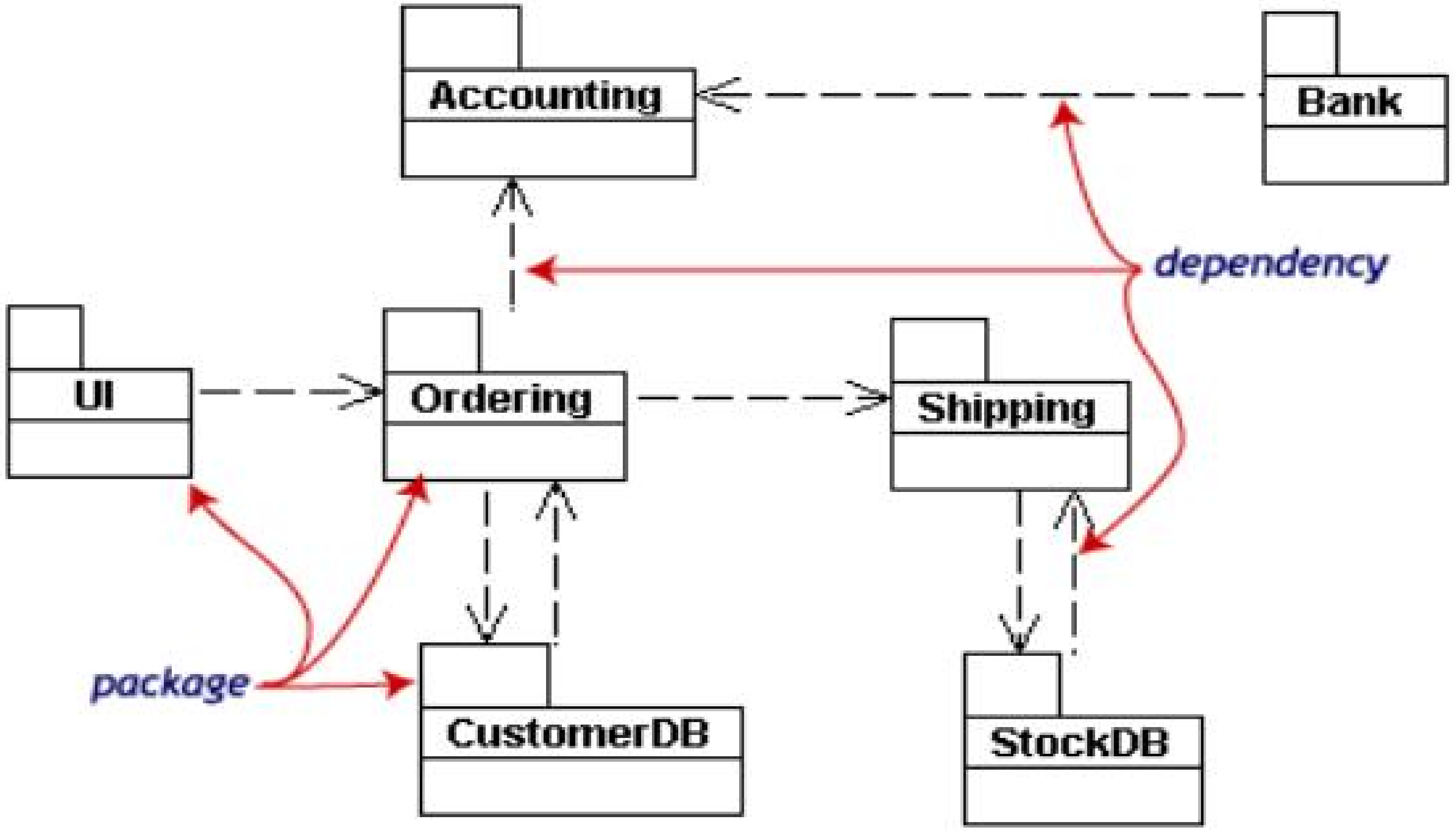
Object diagram



Package diagram

- UML 2 Package diagrams **simplify complex class diagrams**, it can group **classes into packages**.
- A package is a **collection of logically related UML elements**. Packages are depicted as file folders and can be used on any of the UML diagrams.

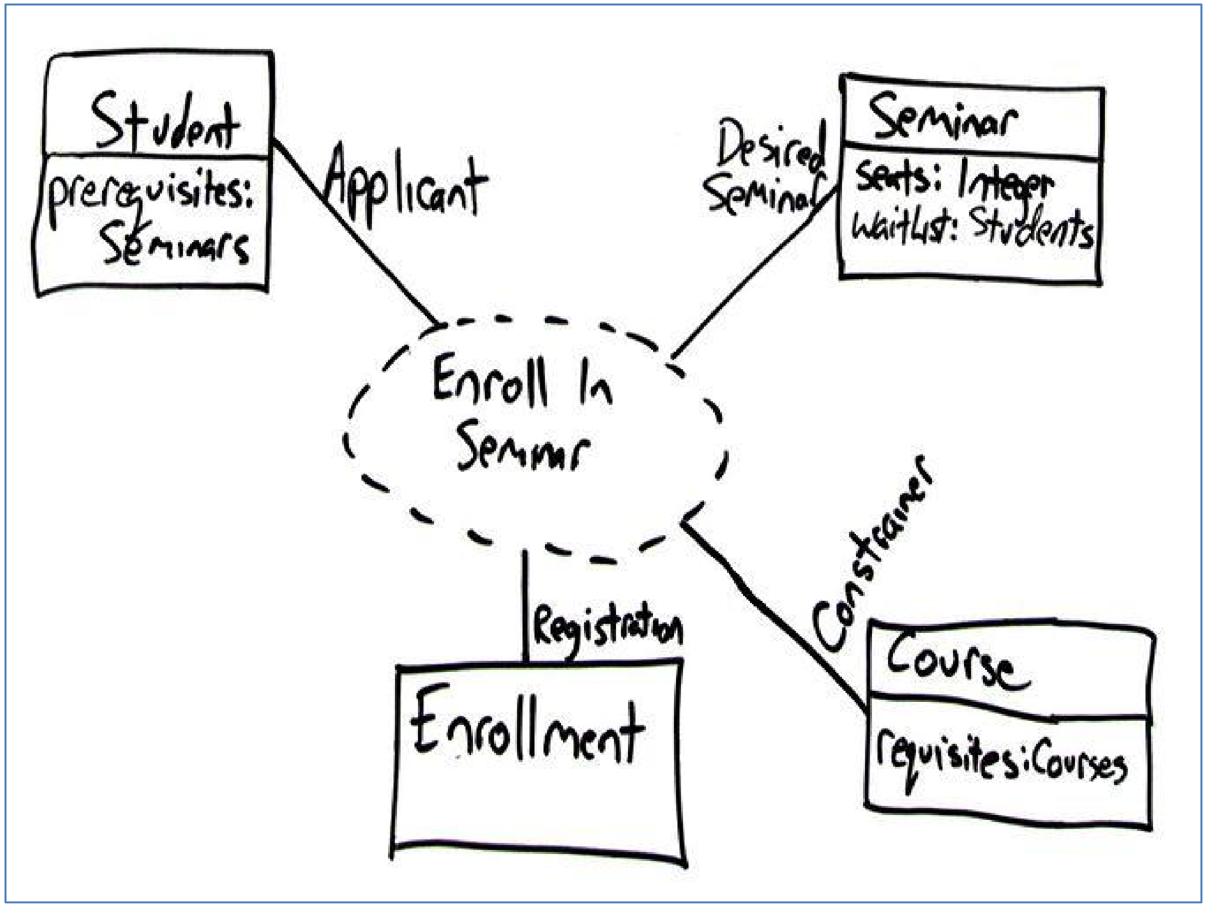
Package diagram



Composite structure diagram

- UML 2 Composite structure diagrams used to explore run-time instances of interconnected instances collaborating over communications links.
- It shows the internal structure (including parts and connectors) of a structured classifier or collaboration.

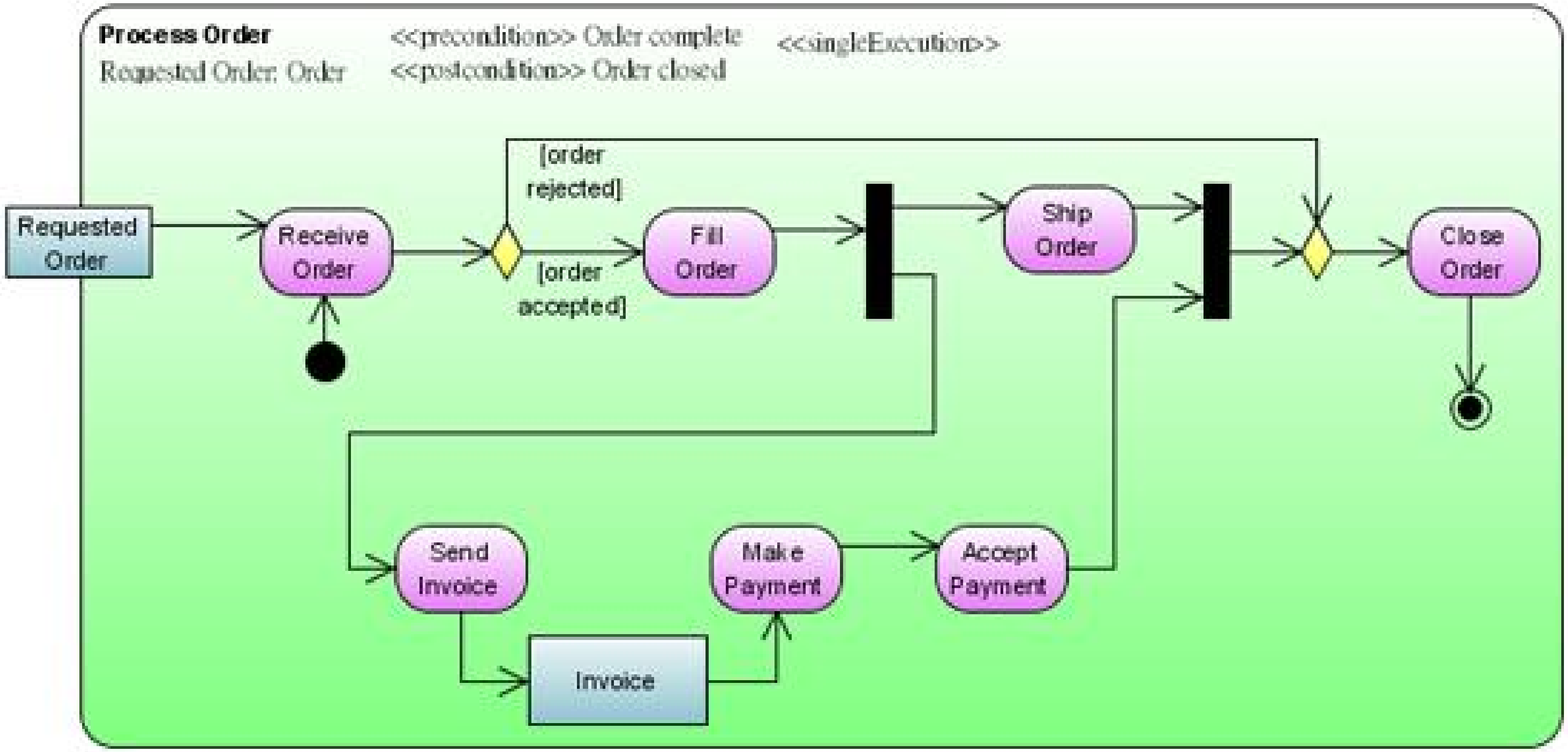
Composite structure diagram



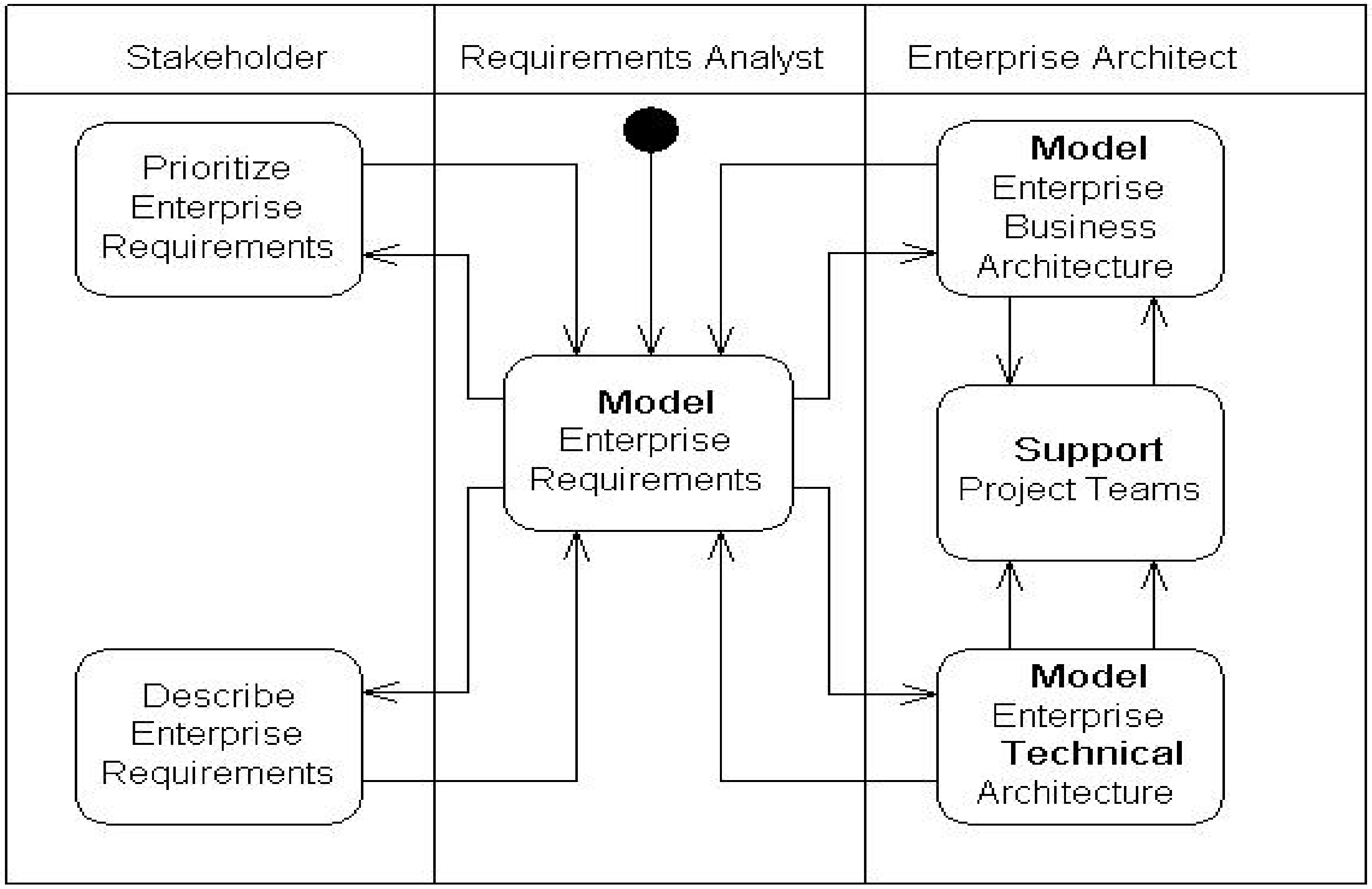
Activity diagram

- UML 2 Activity diagrams helps to describe the flow of control of the target system, such as the exploring complex business rules and operations, describing the use case also the business process.
- It is object-oriented equivalent of flow charts and data-flow diagrams (DFDs).

Activity diagram



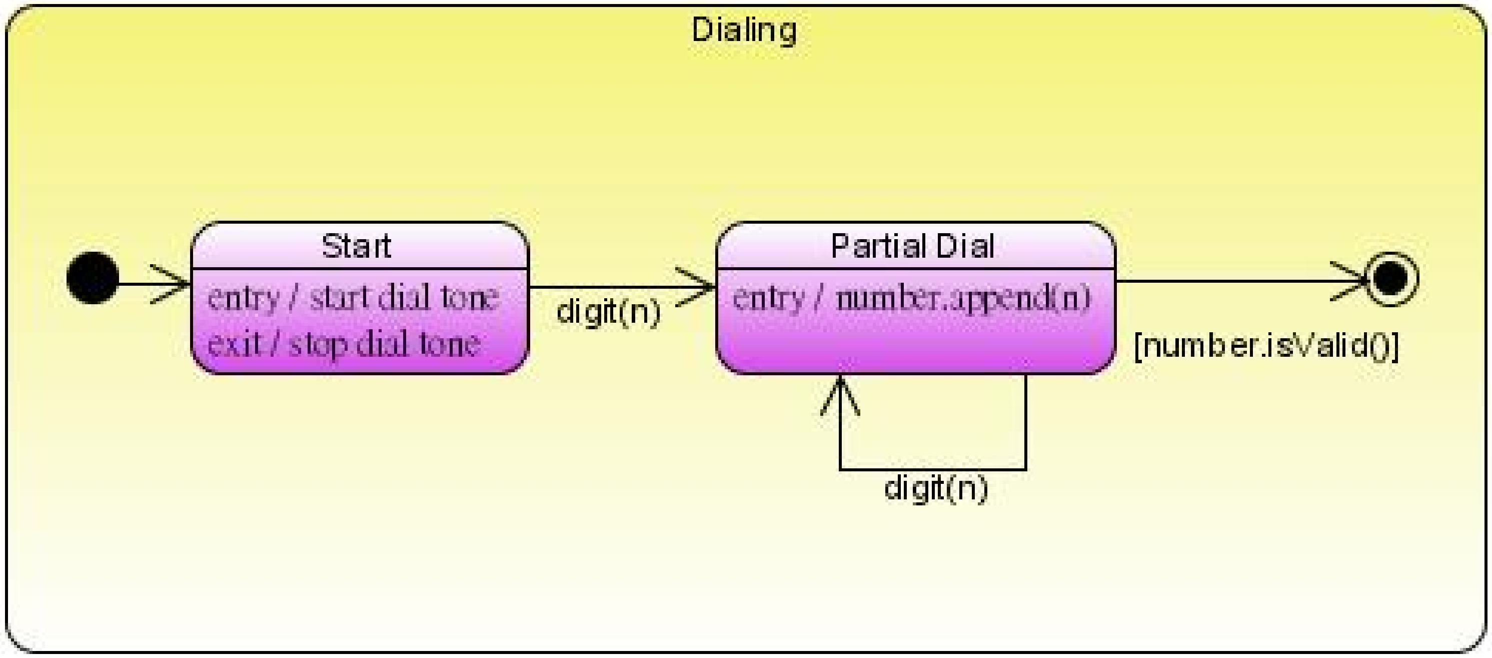
Activity diagram


Stakeholder
Requirements Analyst
Enterprise Architect
Prioritize Enterprise Requirements
Describe Enterprise Requirements
Model Enterprise Requirements
Model Enterprise Business Architecture
Support Project Teams
Model Enterprise **Technical** Architecture

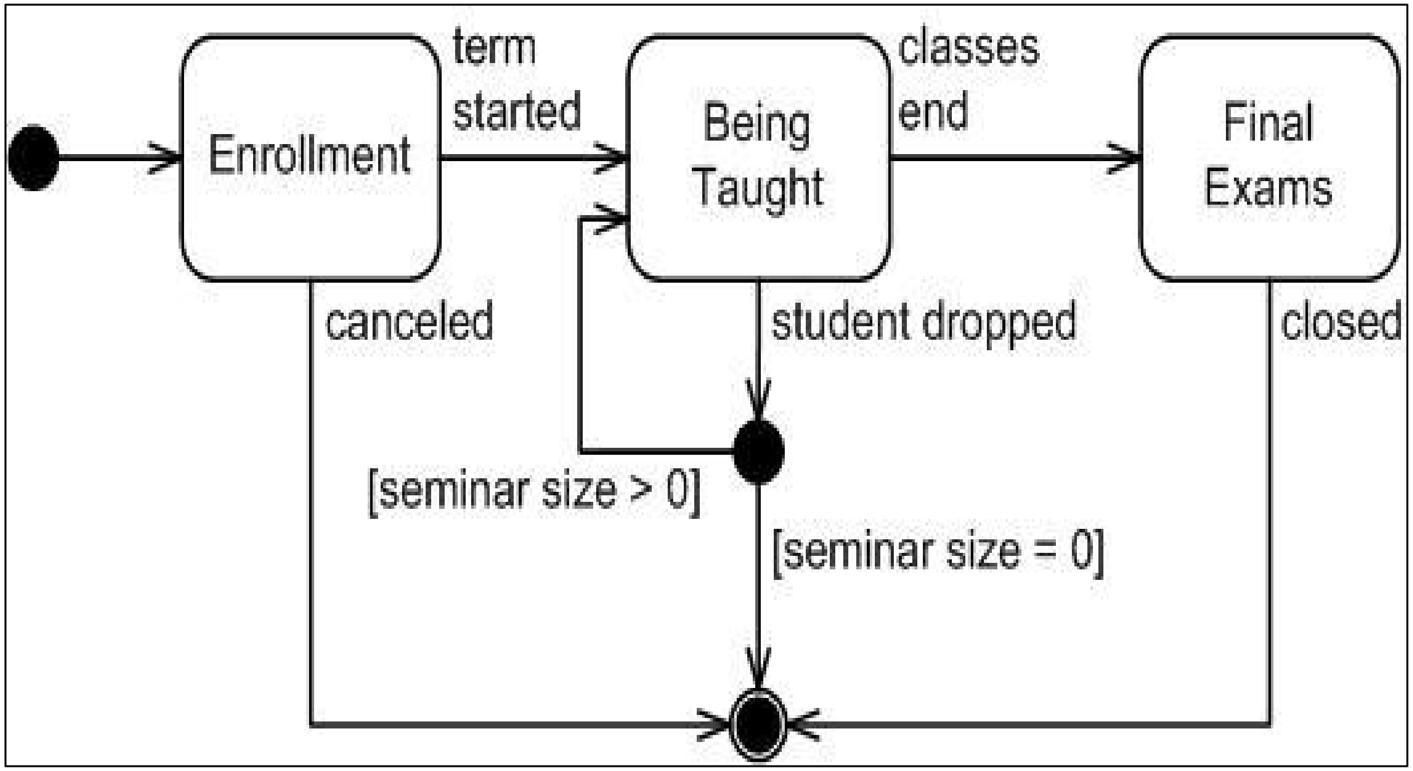
State machine diagram

- UML 2 State machine diagrams can show the **different states of an entity** also how an entity responds to various events by changing from one state to another. The history of an entity can best be modeled by a finite state diagram.

State machine diagram



State machine diagram



Use cases diagram

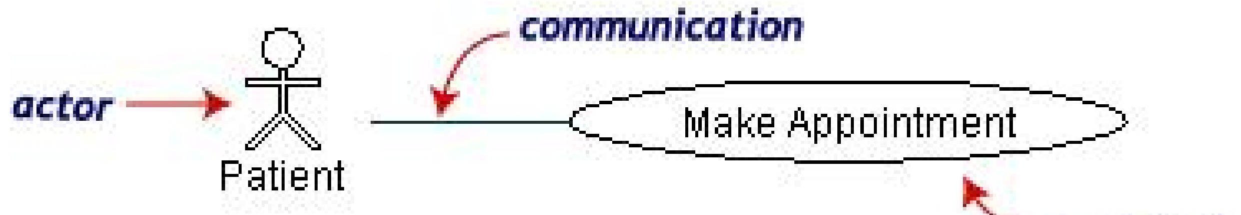
UML 2 Use cases diagrams describes the behavior of the target system from an **external point of view**. Use cases describe "the meat" of the actual requirements.

•**Use cases**. A use case describes a sequence of actions that provide something of measurable value to an actor and is drawn as a horizontal ellipse.

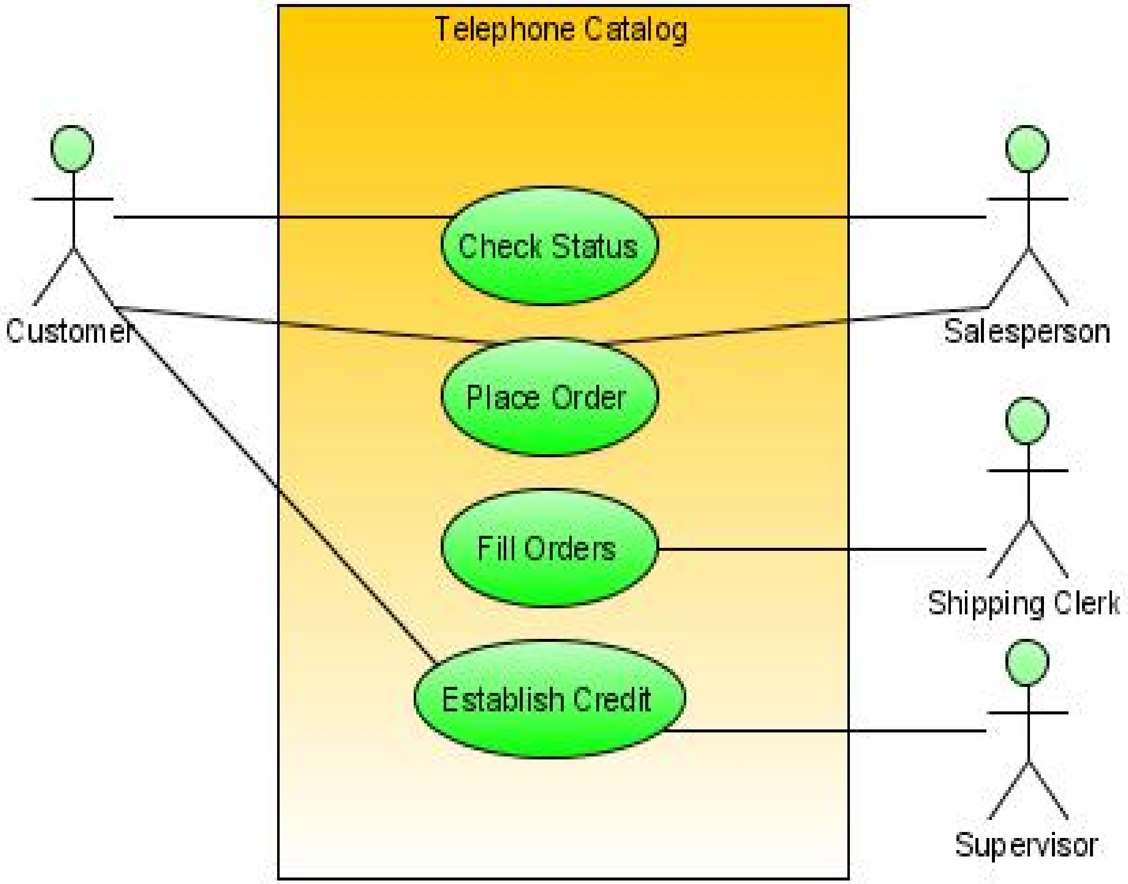
•**Actors**. An actor is a person, organization, or external system that plays a role in one or more interactions with your system. Actors are drawn as stick figures.

•**Associations**. Associations between actors and use cases are indicated by solid lines. An association exists whenever an actor is involved with an interaction described by a use case.

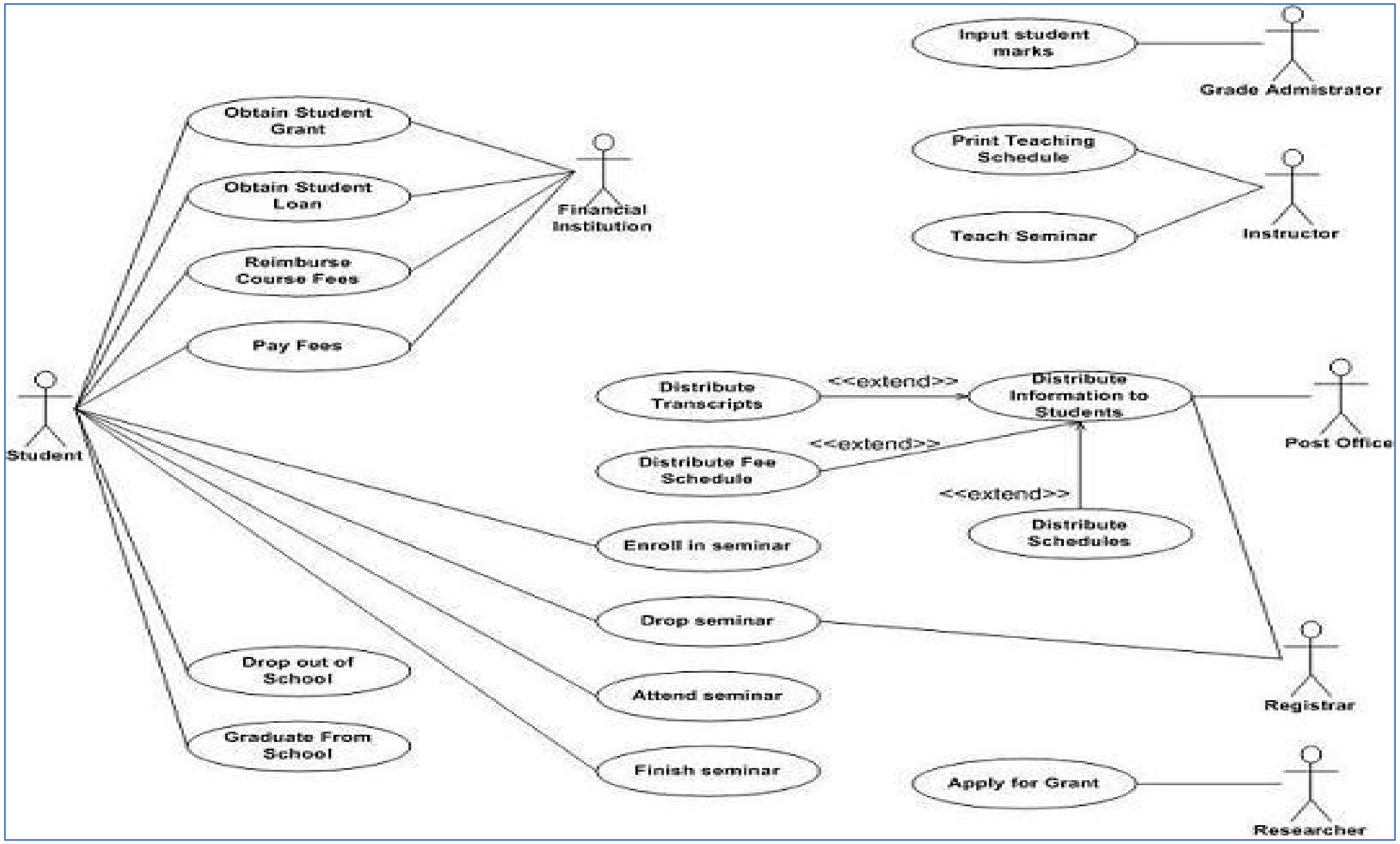
Use cases diagram



Use cases diagram



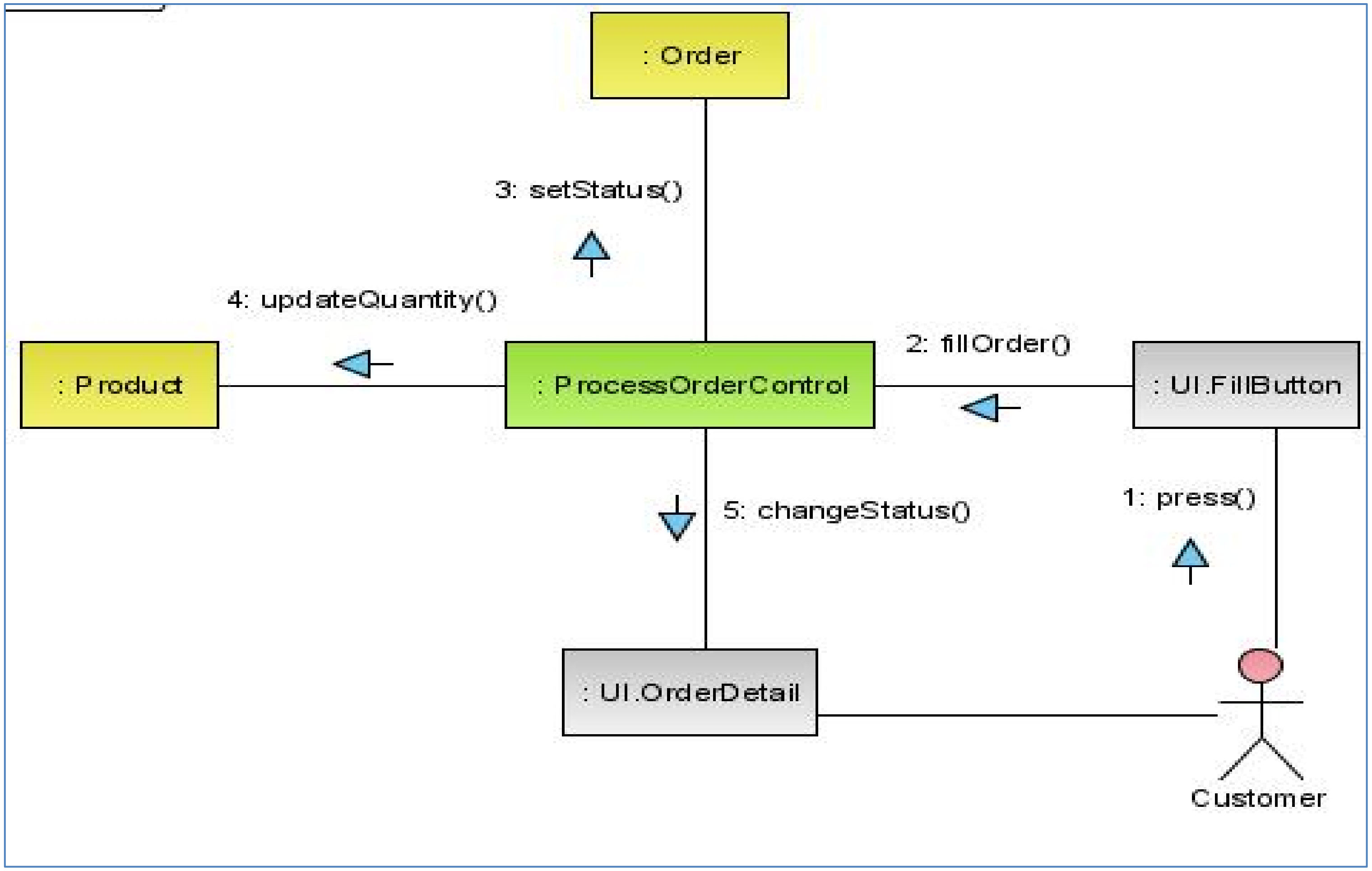
Use cases diagram



Communication diagram

- UML 2 Communication diagrams used to model the dynamic behavior of the use case. When compare to Sequence Diagram, the Communication Diagram is more focused on showing the collaboration of objects rather than the time sequence.

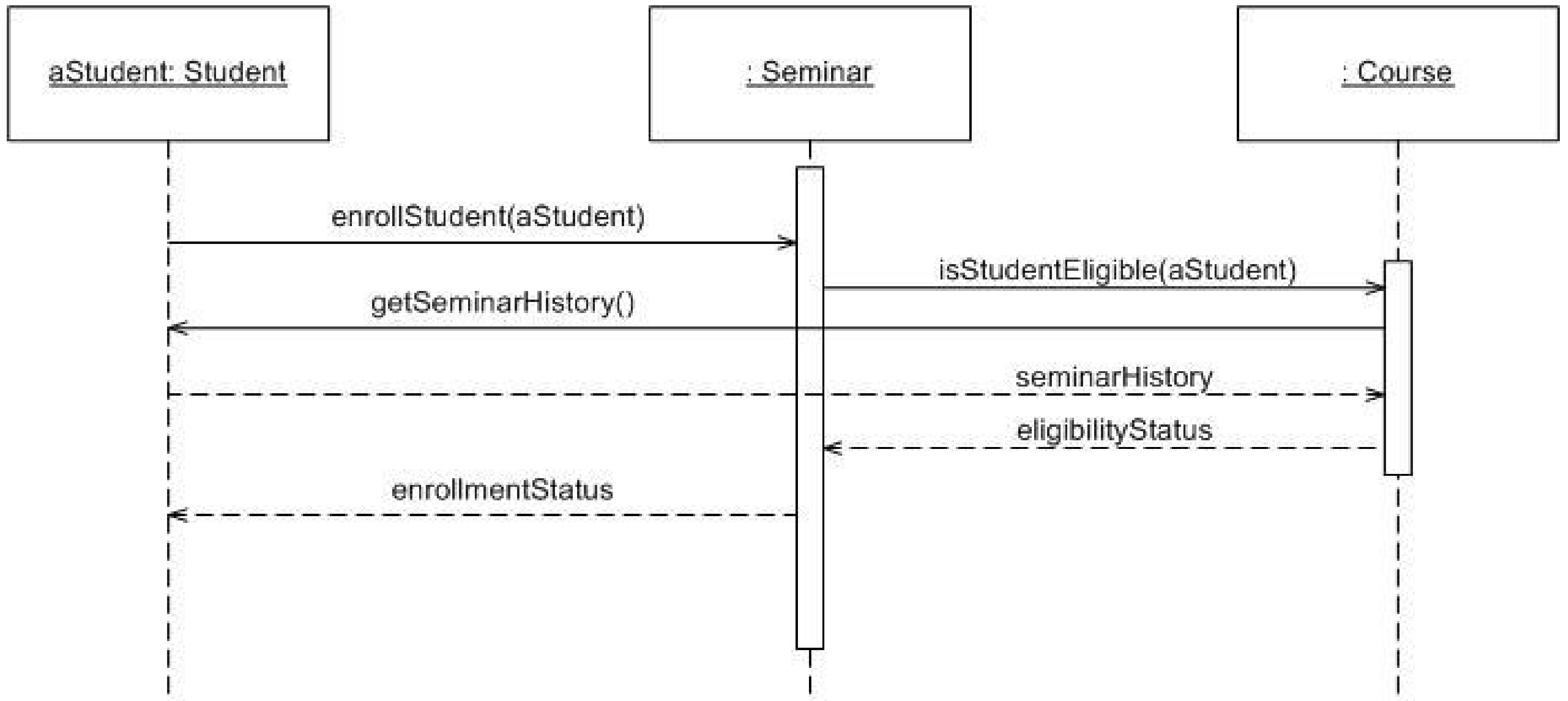
Communication diagram



Sequence diagram

- UML 2 Sequence diagrams models the ***collaboration of objects*** based on a time sequence. It shows how the objects interact with others in a particular scenario of a use case.

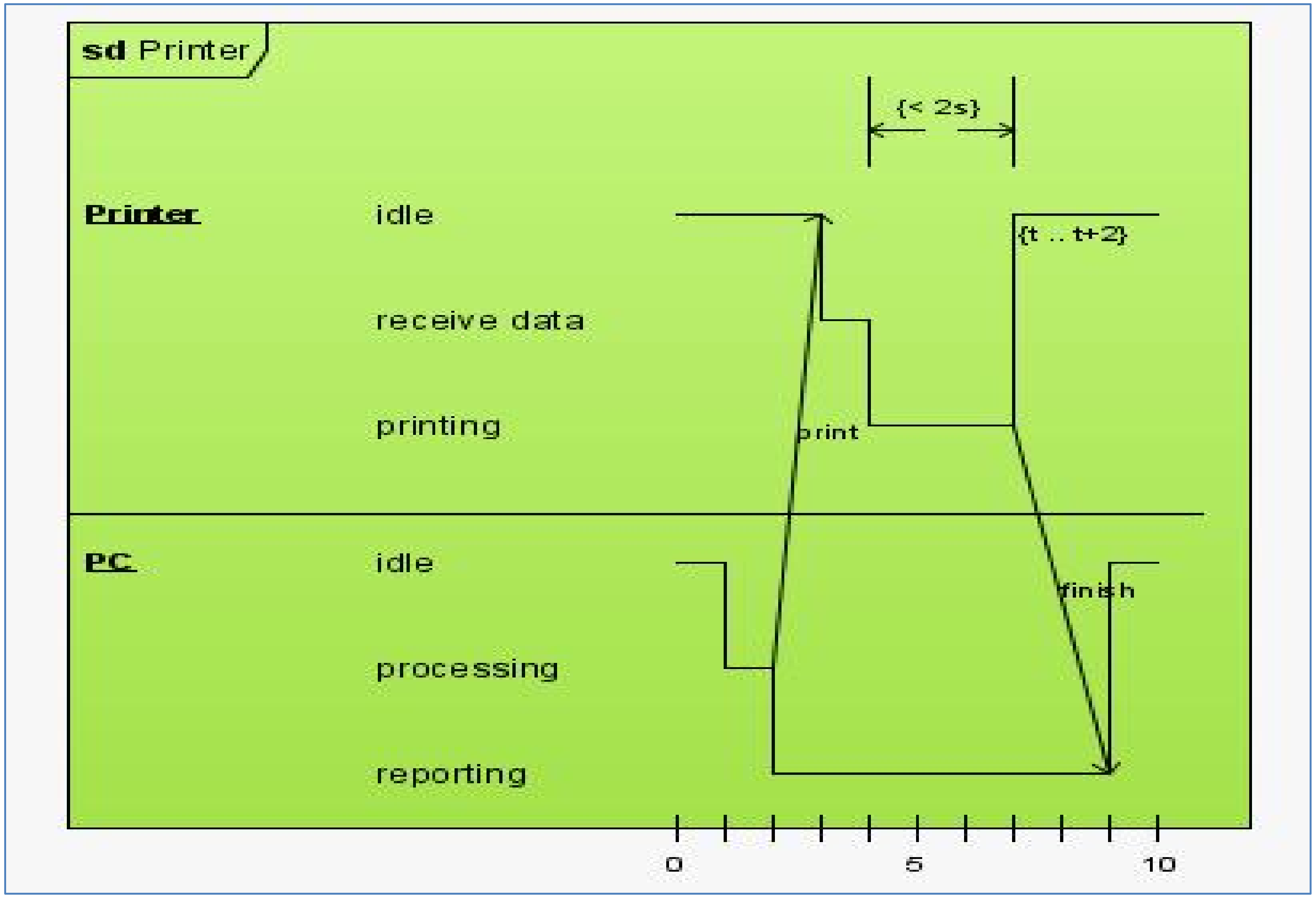
Sequence diagram



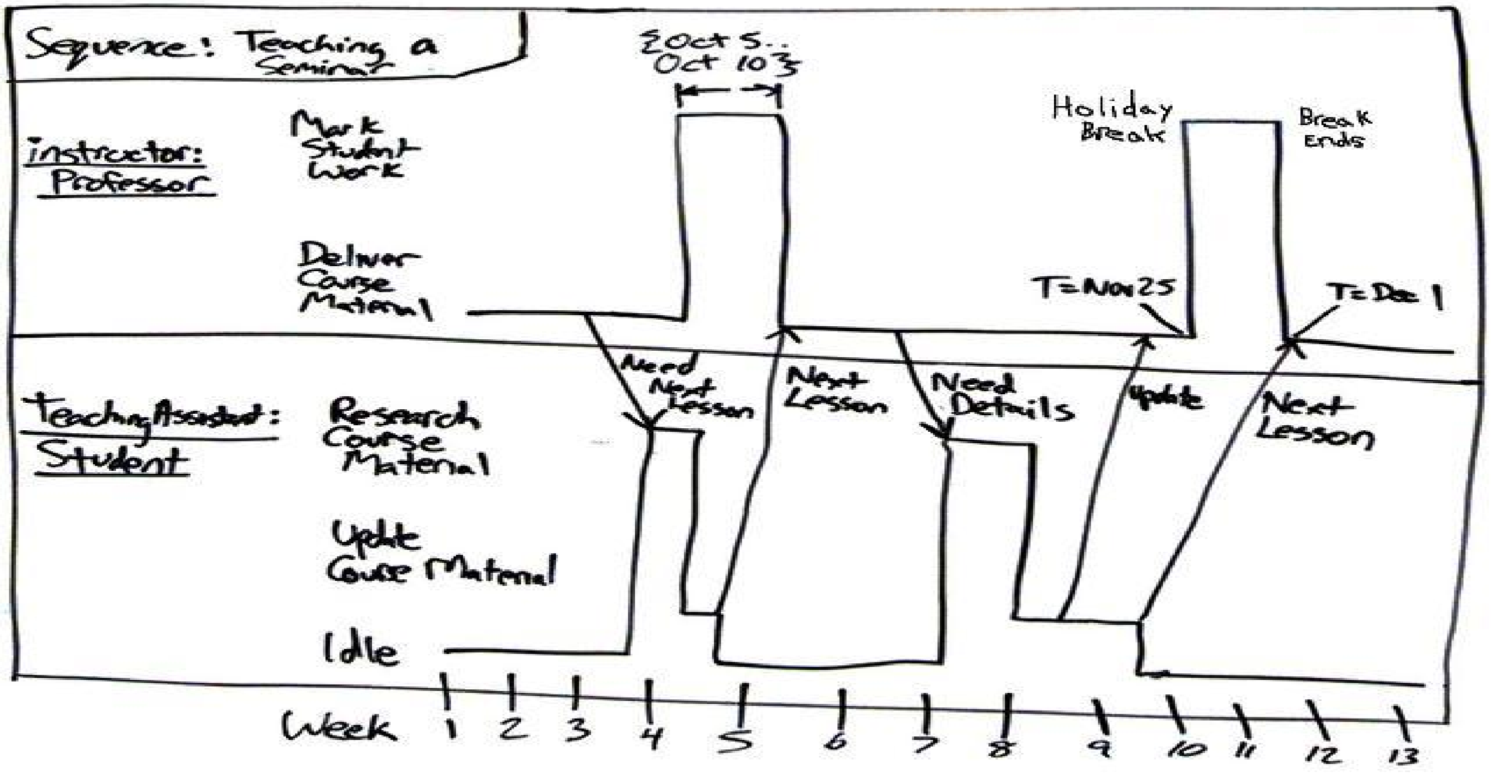
Timing diagram

- UML 2 Timing diagrams shows the behavior of the objects in a given period of time. Timing diagram is a special form of a sequence diagram.
- The differences between timing diagram and sequence diagram are the axes are reversed so that the time are increase from left to right and the lifelines are shown in separate compartments arranged vertically.

Timing diagram



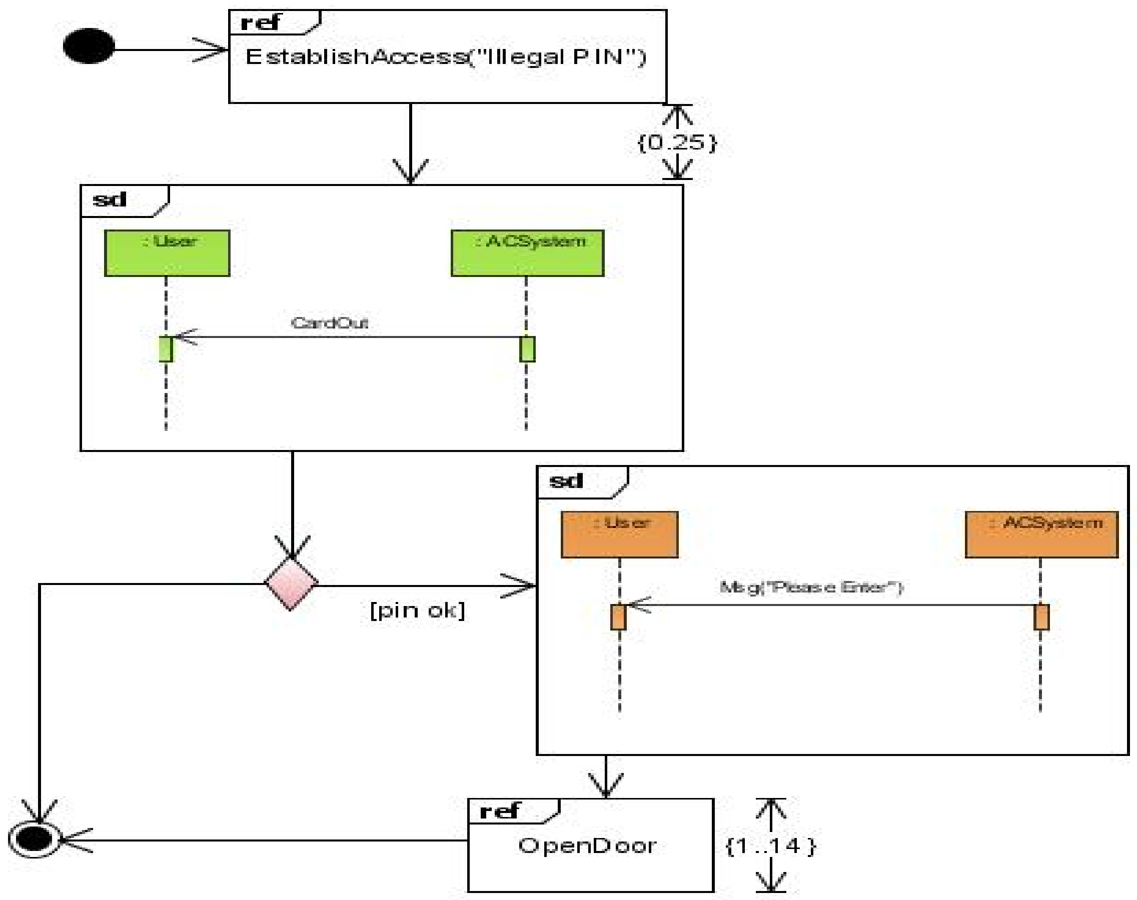
Timing diagram



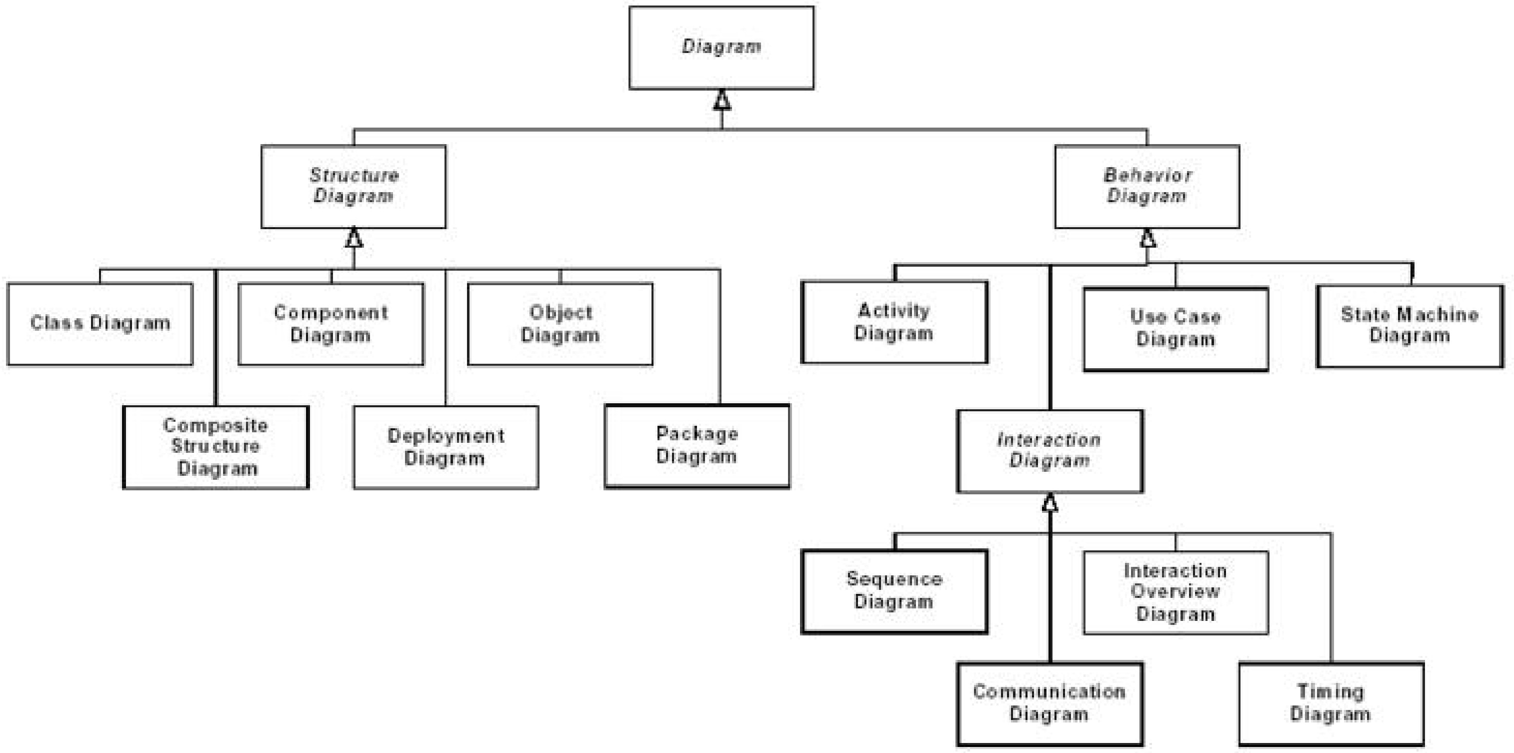
Interaction overview diagram

- UML 2 Interaction overview diagrams focuses on the **overview of the flow of control** of the interactions. It is a variant of the Activity Diagram where the nodes are the interactions or interaction occurrences. It describes the interactions where messages and lifelines are hidden.

Interaction overview diagram



UML diagram hierarchy



Thank you