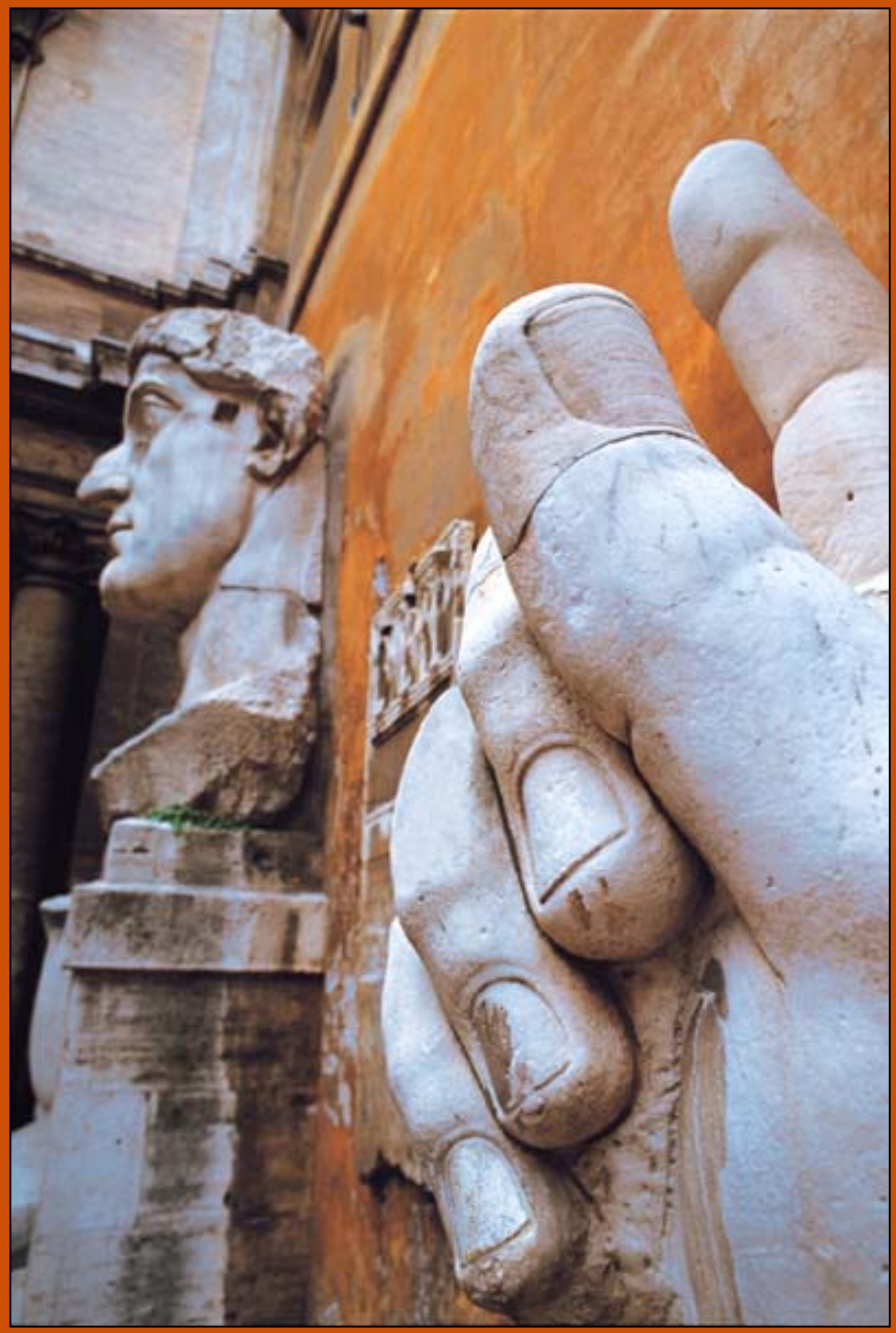


# *UML*

***Unified Modeling Language***

a standard language to analyze, design and document software intensive solutions

# Modeling with UML

## Building blocks

When you model something, you create a simplification of reality so that you can better understand the system you are developing. Using the UML, you build your models from basic building blocks, such as classes, interfaces, collaborations, components, nodes, dependencies, generalizations and associations.

## UML diagrams

Diagrams are the means by which you view these building blocks. A diagram is a graphical presentation of a set of elements. You use diagram to visualize your system from different prospective. Because no complex system can be understood in its entirety from only one perspective, the UML defines a number of diagrams so that you can focus on different aspects of your system independently.

# UML Diagrams

**Nine predefined diagrams**

- **Class diagram**
- **Object diagram**
- **Activity diagram**
- **Sequence diagram**
- **Collaboration diagram**
- **Use-case diagram**
- **Component diagram**
- **Deployment diagram**
- **Statechart diagram**

# Class diagram

**A class diagram describes the structure of a system. The structures are built from classes and relationships. The classes can represent and structure information, products, documents or organizations.**

**Class**

**set of objects with the same characteristics (Person, Invoice, Company, Order, Product, ...)**

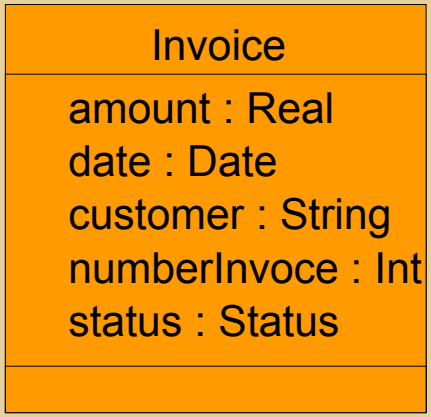
**Relationship**

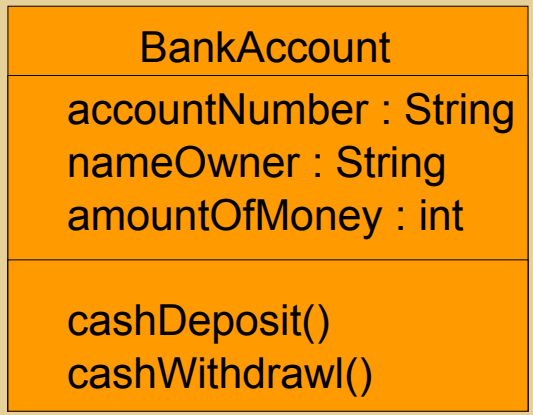
**indicates that between two classes there is a link or a type of dependence**

# Class

**The graphical notation for a class is a "rectangle" with tree compartments:**

- **name compartment**
- **attribute compartment**
- **operation compartment**

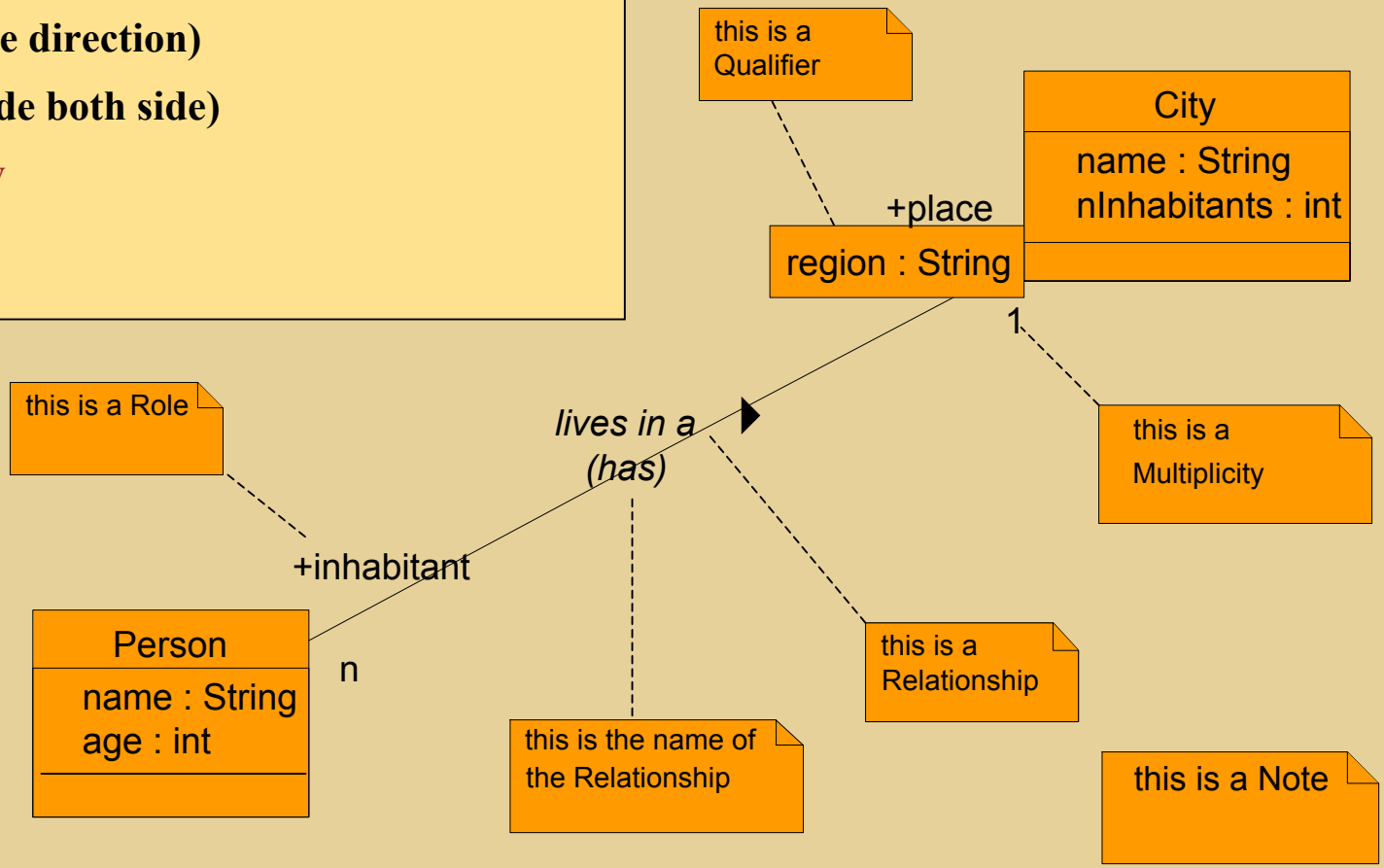


# Relationship

**The graphical notation for a relationship is straight line. A relationship has (can have):**

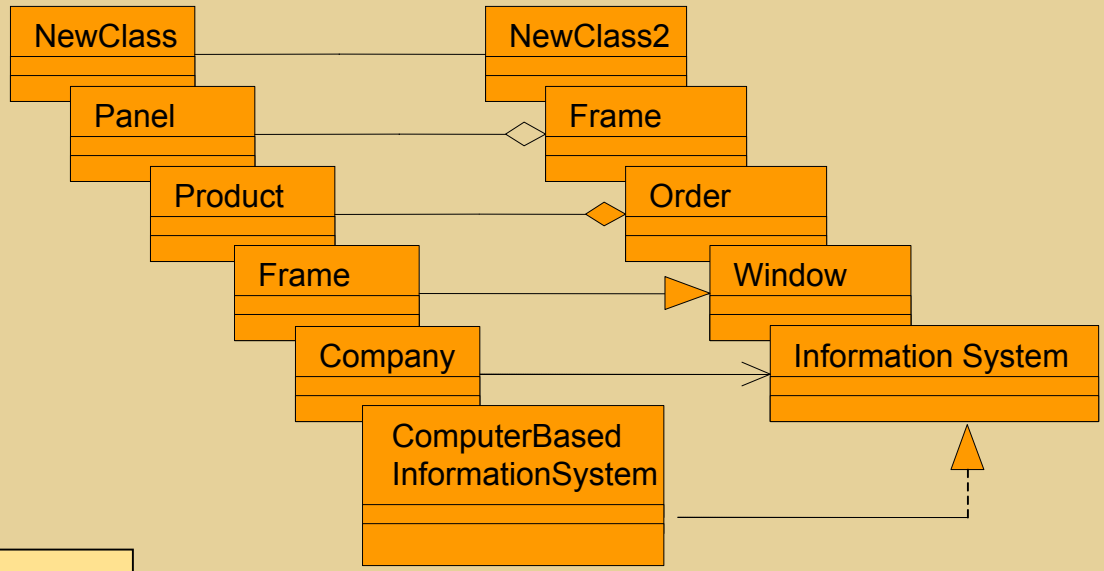
- **name (name direction)**
- **role (one side both side)**
- **multiplicity**
- **qualifier**



# Relationship

**A relationship is an "abstract" concept. When we draw diagrams we use special types of relationships :**

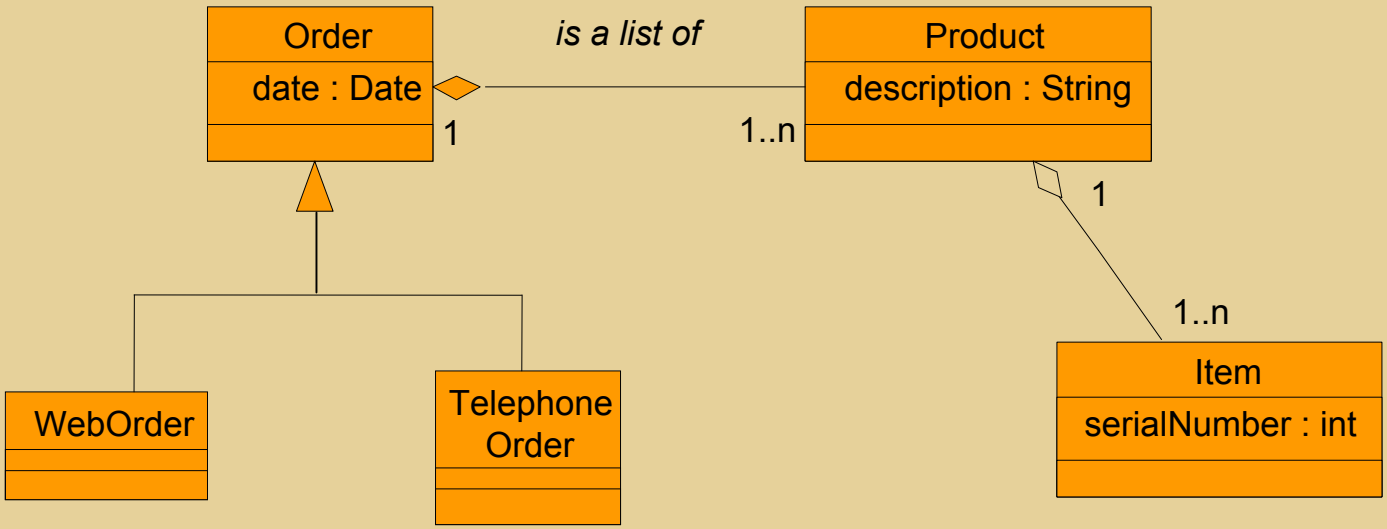
- **Association**
- **Aggregation**
- **Composition**
- **Generalization**
- **Dependency**
- **Realization**




**A relationship can have a direction**

# Class diagram

**A simple example**

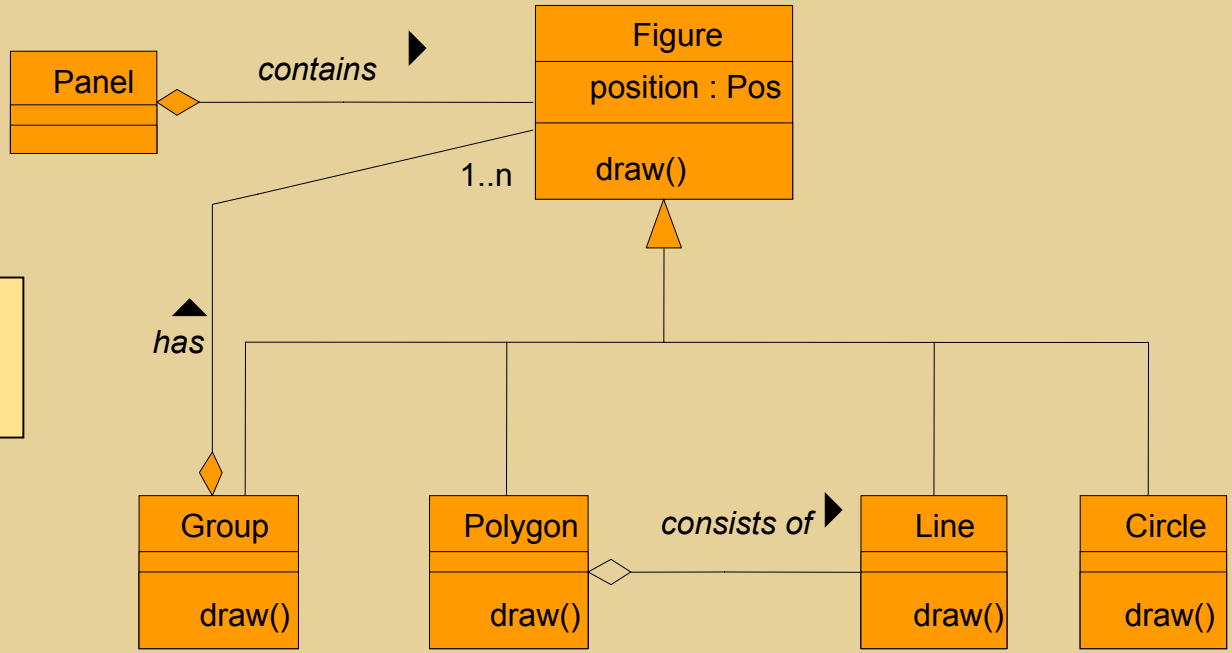


# Object diagram

**An object diagram expresses possible objects combination of a specific class diagram. It is typically used to exemplify a class diagram.**

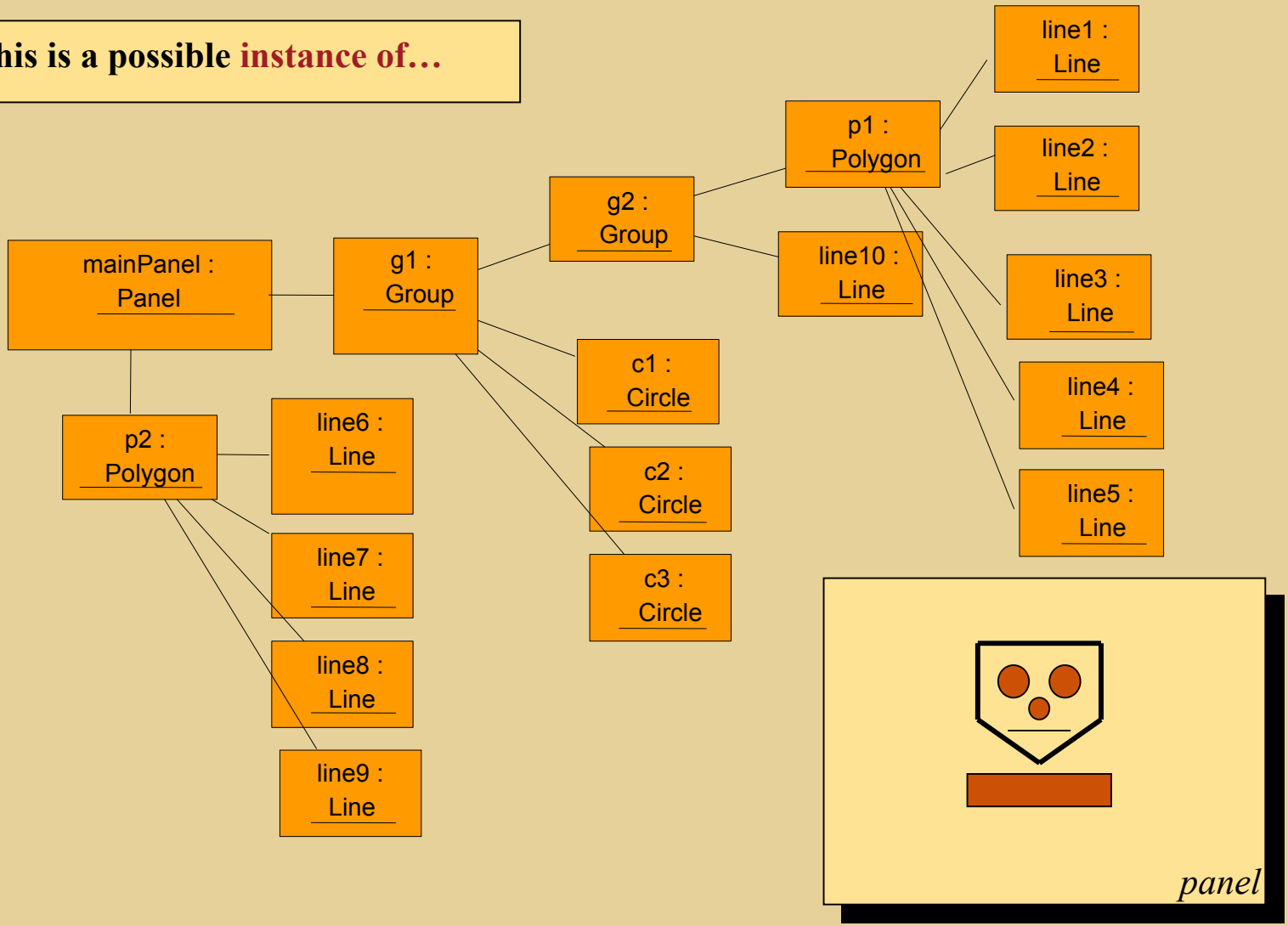
**The graphical notation for an object is the same as a class. The name of the object is followed by the name of the class and it is underlined.**

**This is the class diagram...**



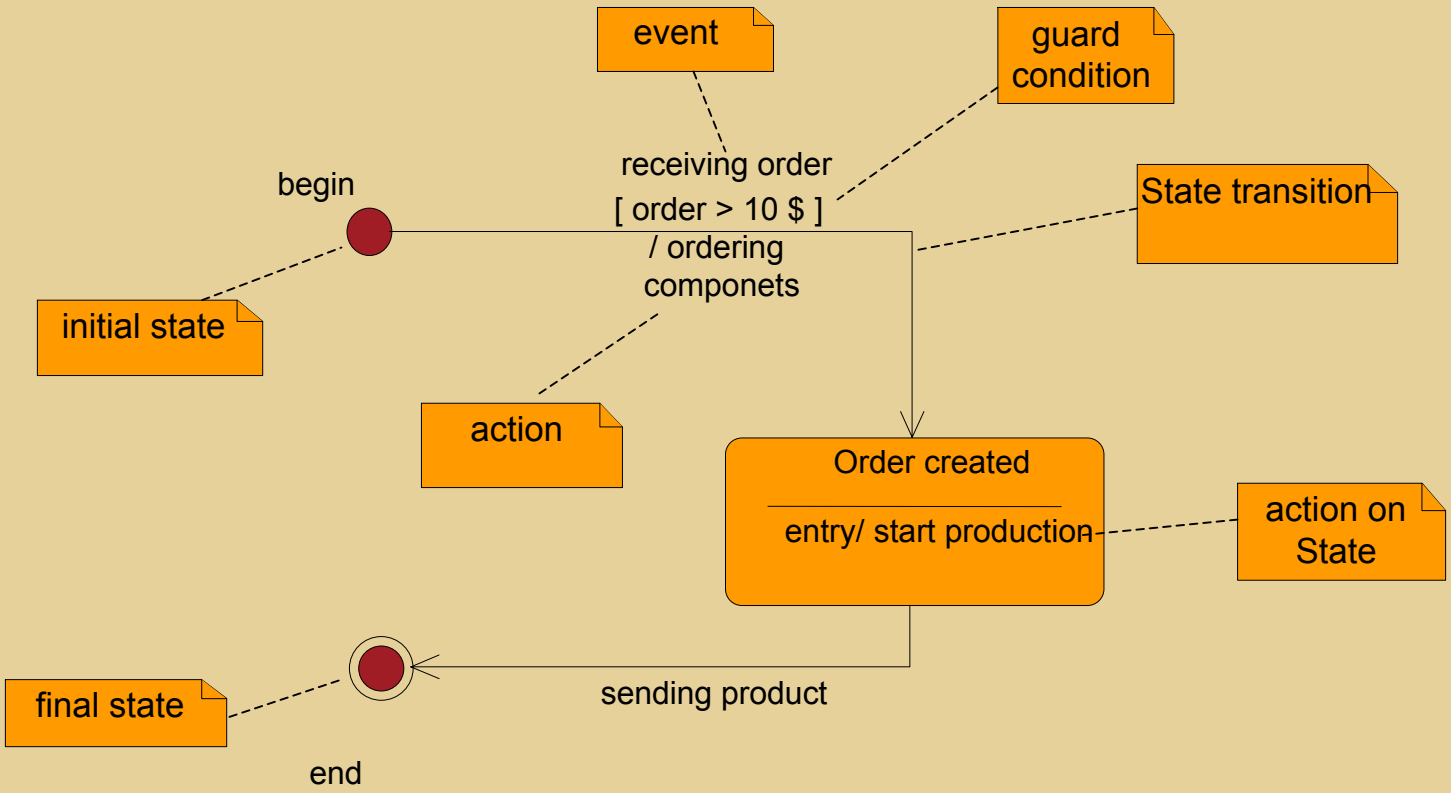
# Object diagram

**An this is a possible instance of…**



# Statechart diagram

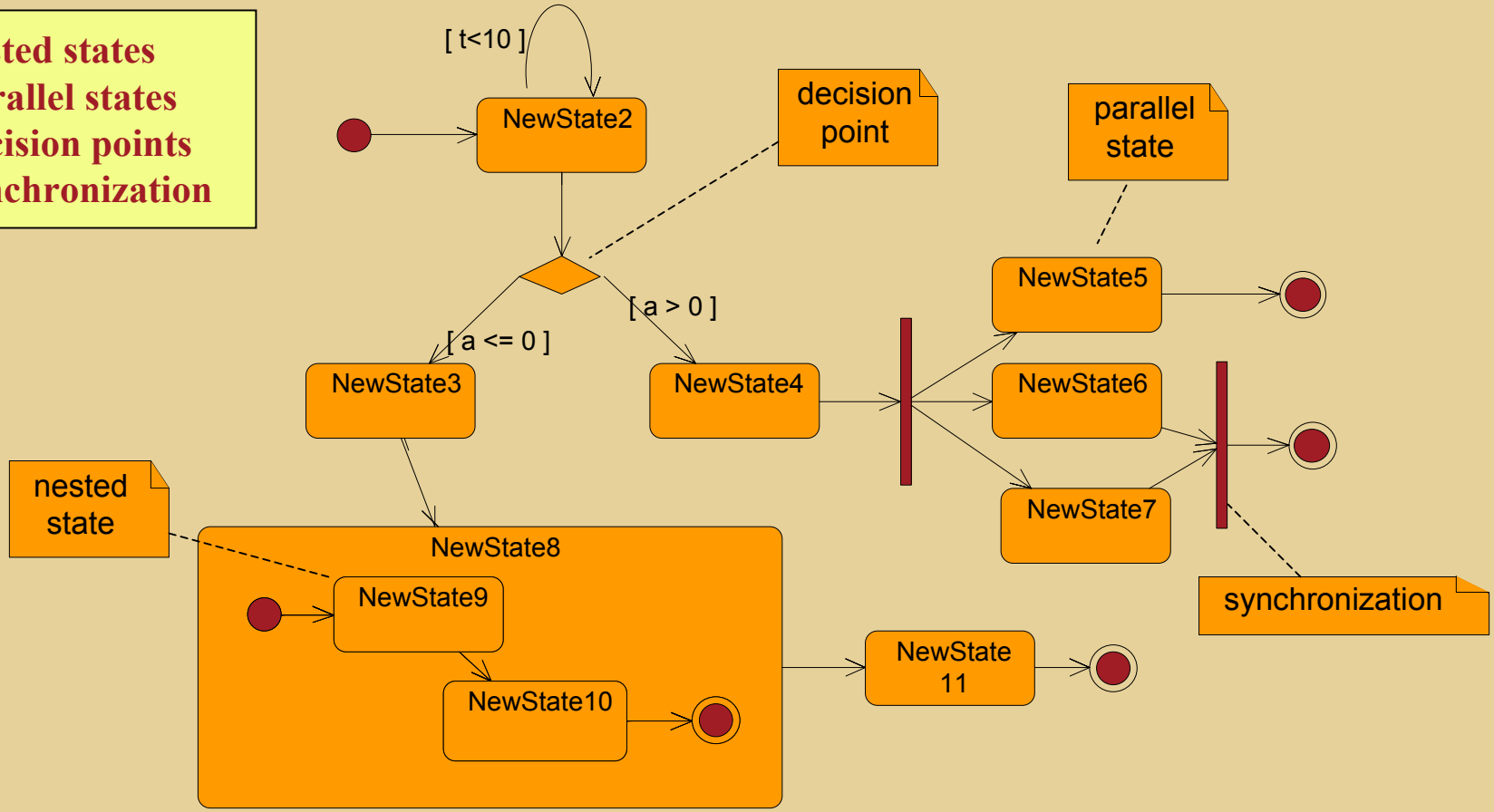
**A statechart diagram expresses possible states of a class (or a system).**
**It indicates what states an object can have and how different events affects those states over time.**



# Statechart diagram

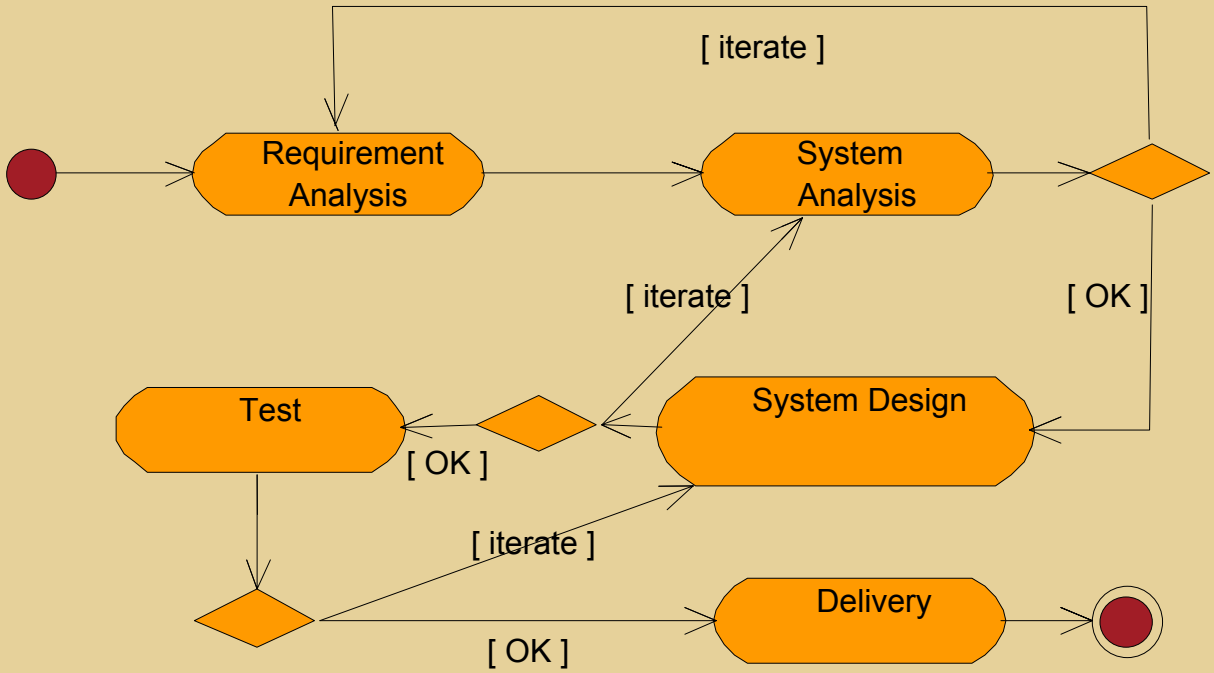
**It is also possible to model in a statechart diagram:**

- nested states
- parallel states
- decision points
- synchronization



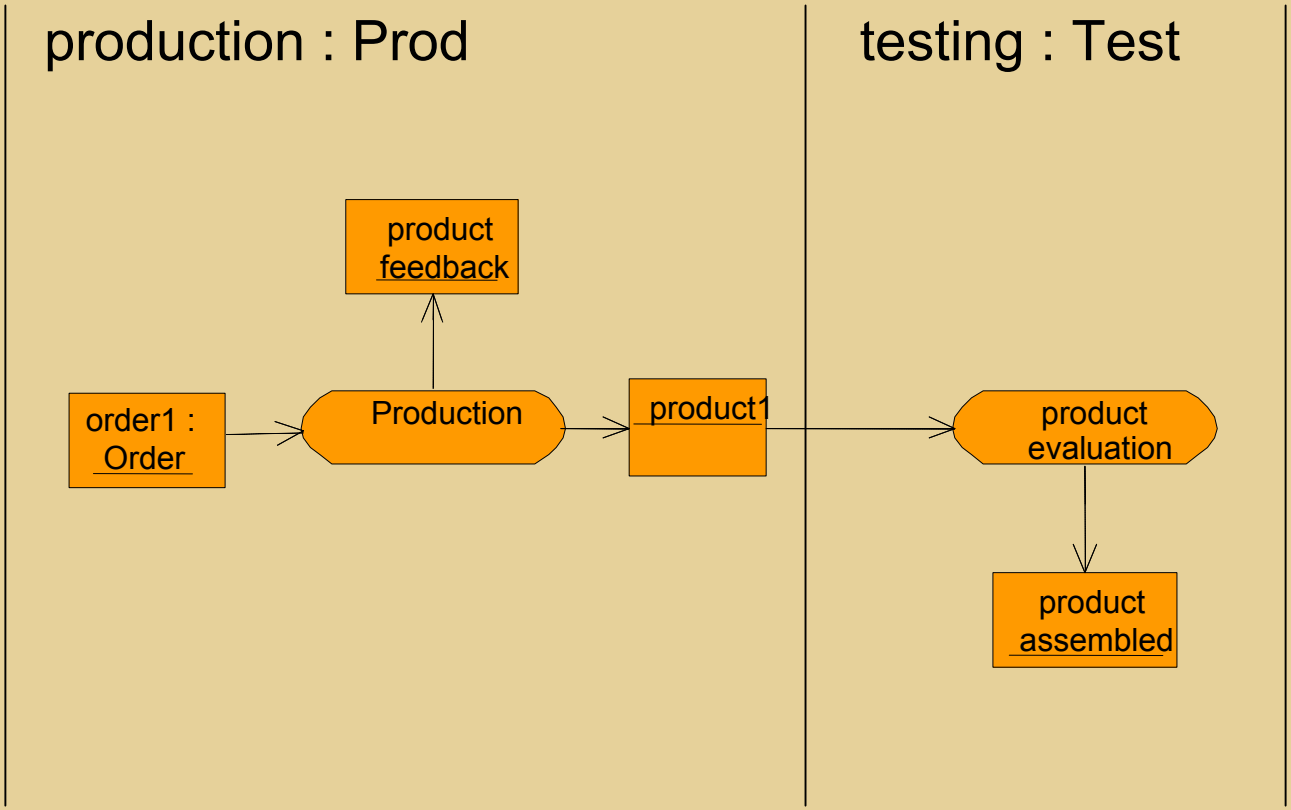
# Activity diagram

**An activity diagram describes activities (actions) taking place in a system. The graphical notation (and syntax) is similar to the one used for a statechart diagram.**



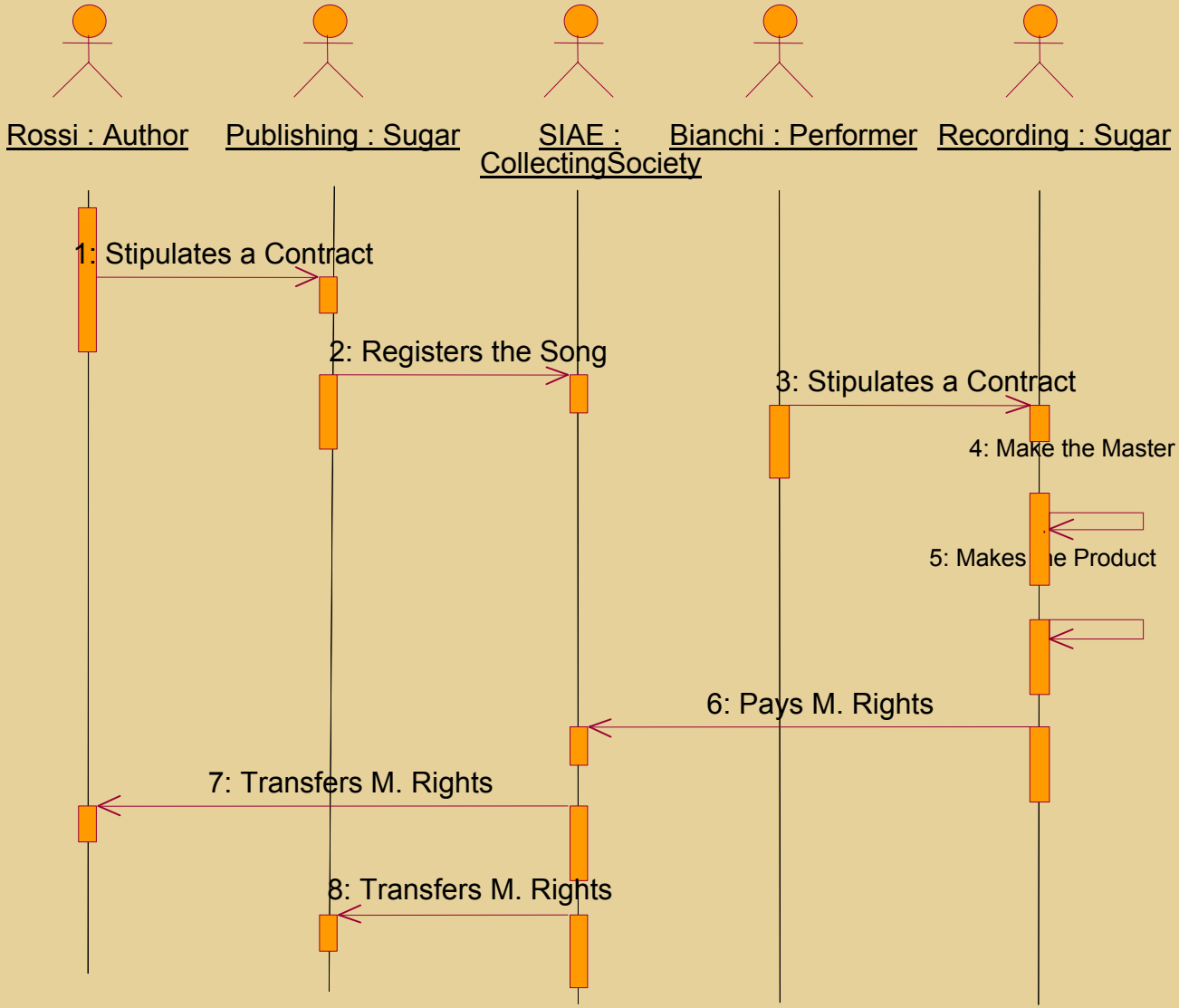
# Activity diagram

**With activity diagrams it is possible to model the flow of objects and the objects responsibilities of the actions (swimlanes) .**



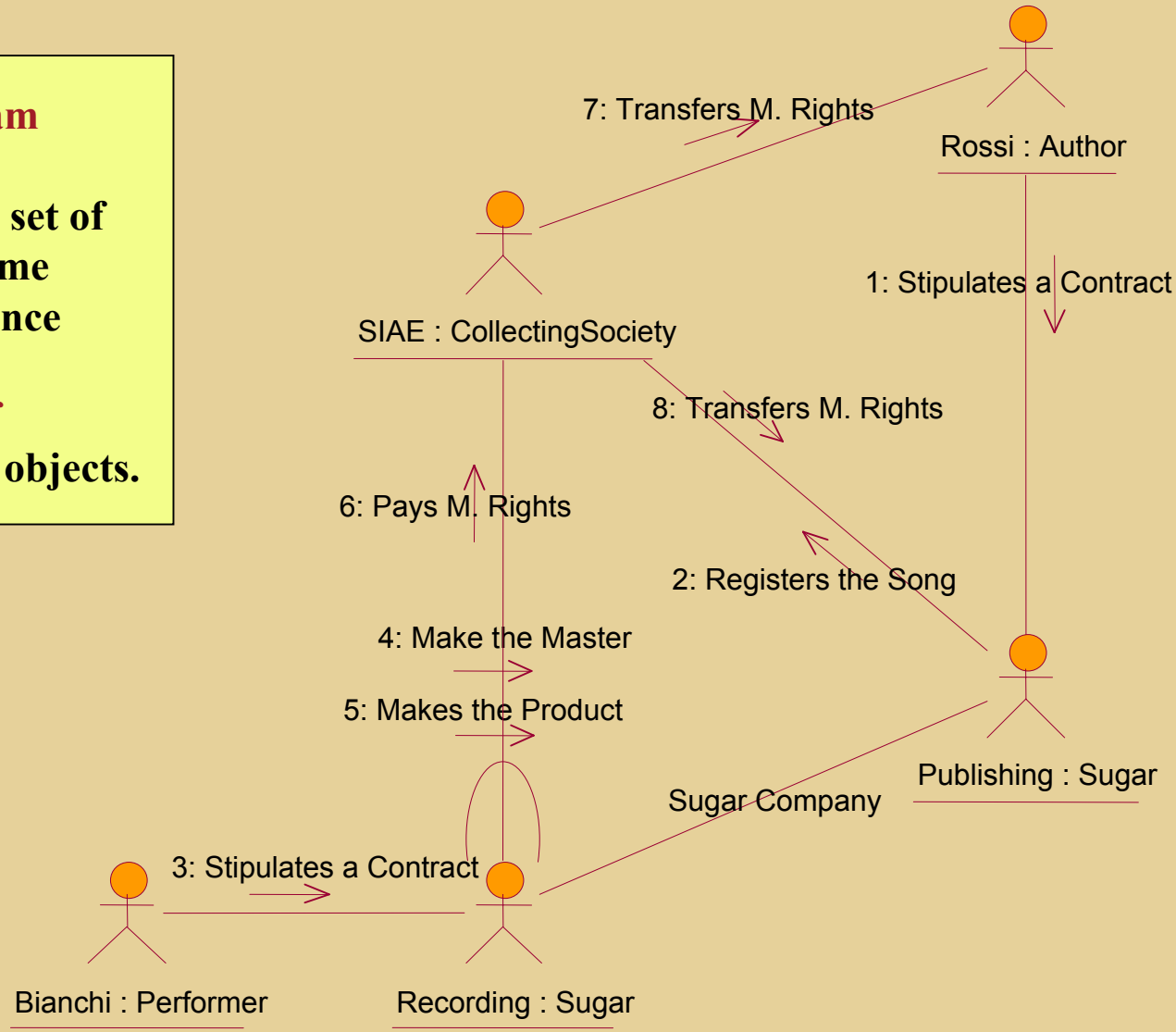
# Sequence diagram

**A sequence diagram shows one or several sequences of messages sent among a set of objects. It depicts the order and time of the messages.**



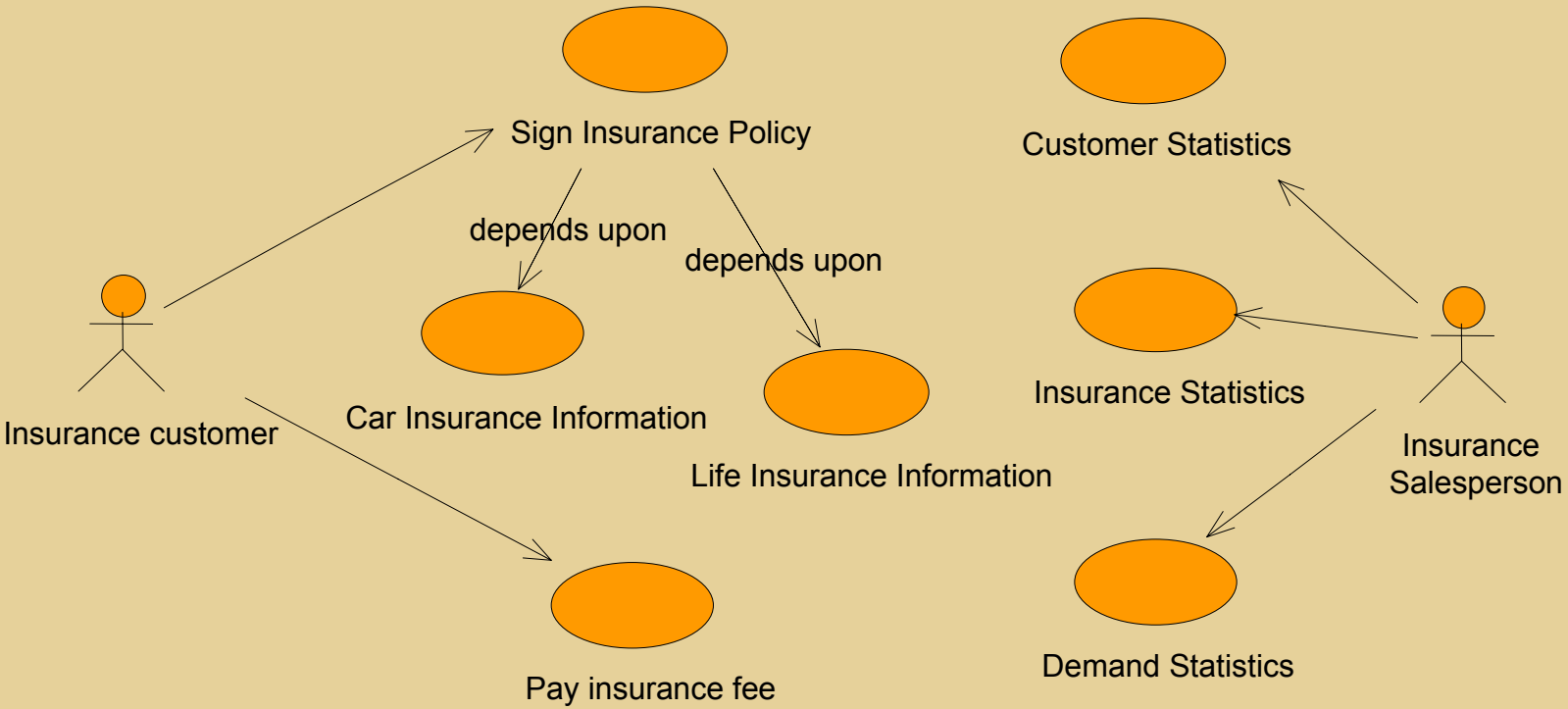
# Collaboration diagram

**A collaboration diagram describes a complete collaboration among a set of objects. It holds the same information of a sequence diagram but it can emphasizes the type of collaboration between objects.**



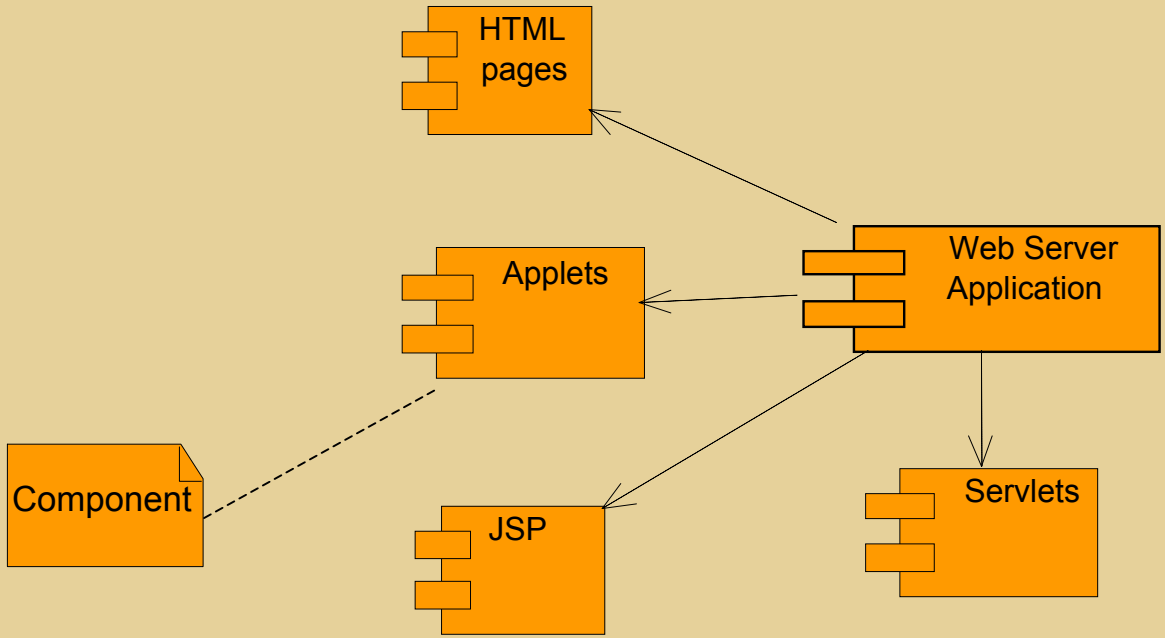
# Use Case diagram

**Illustrate the relationship between use cases, where each use case captures the functional requirements of a system.**
**The basic elements that are used in a use case diagrams are actors and use cases.**



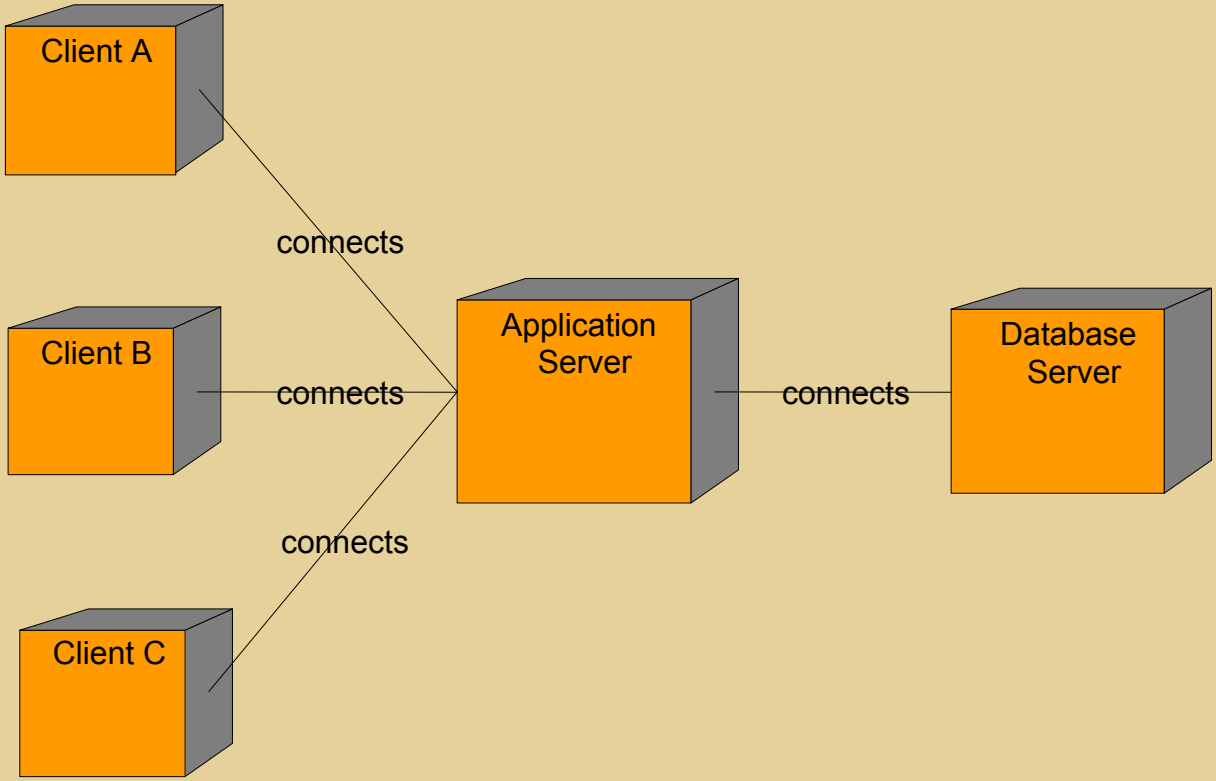
# Component diagram

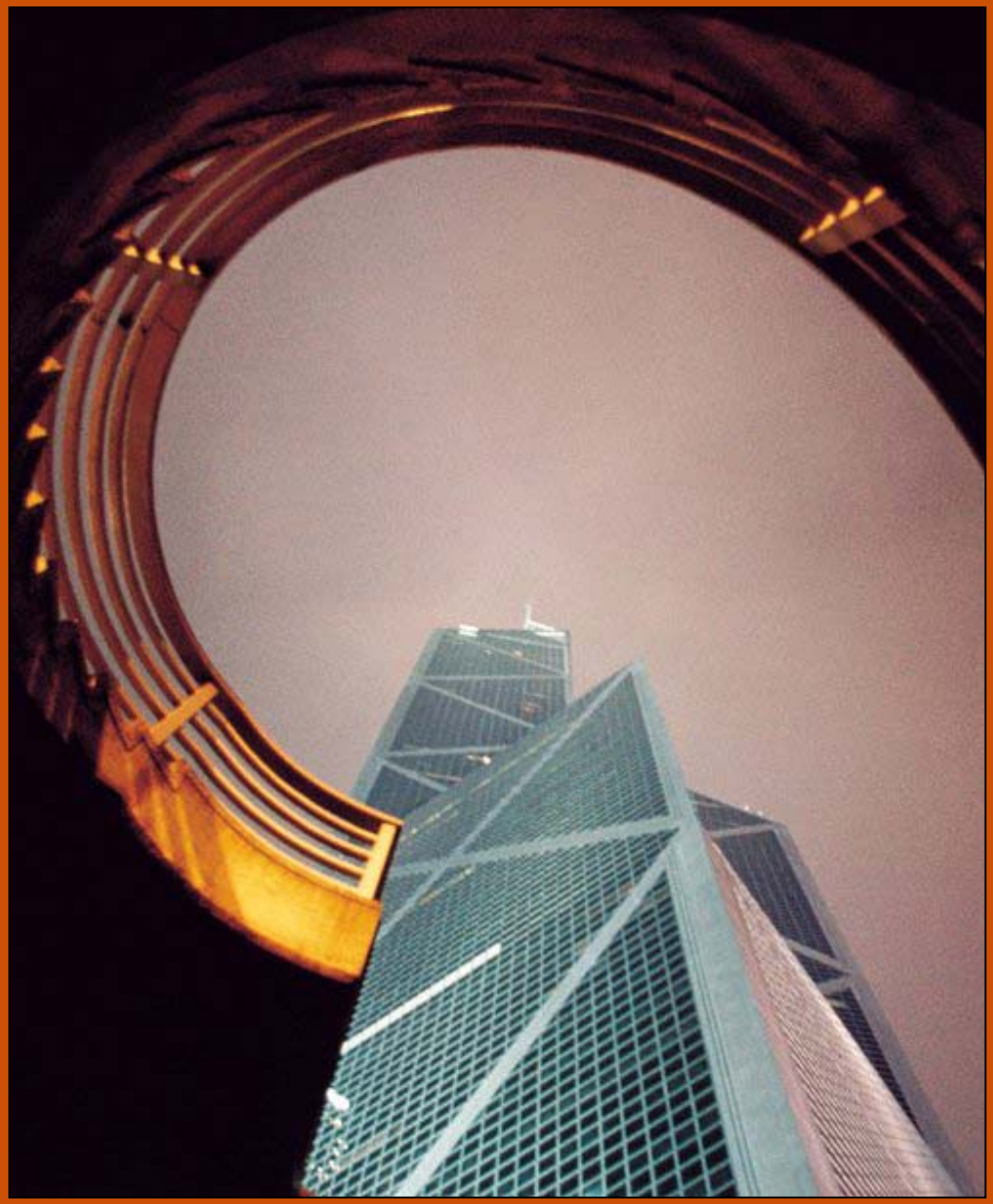
**A special case of class diagram used to describe components within a software system. Components in UML are physical, such as source code files, libraries, applets, JSP, or executable programs.**



# Deployment diagram

**It is used to describe hardware within a software system.**
**The basic block elements that you can use are nodes, that are further specialized in devices and processors.**





# *Extending the language*

**A “good” language should provide the vocabulary and rules to create a well-formed model.**

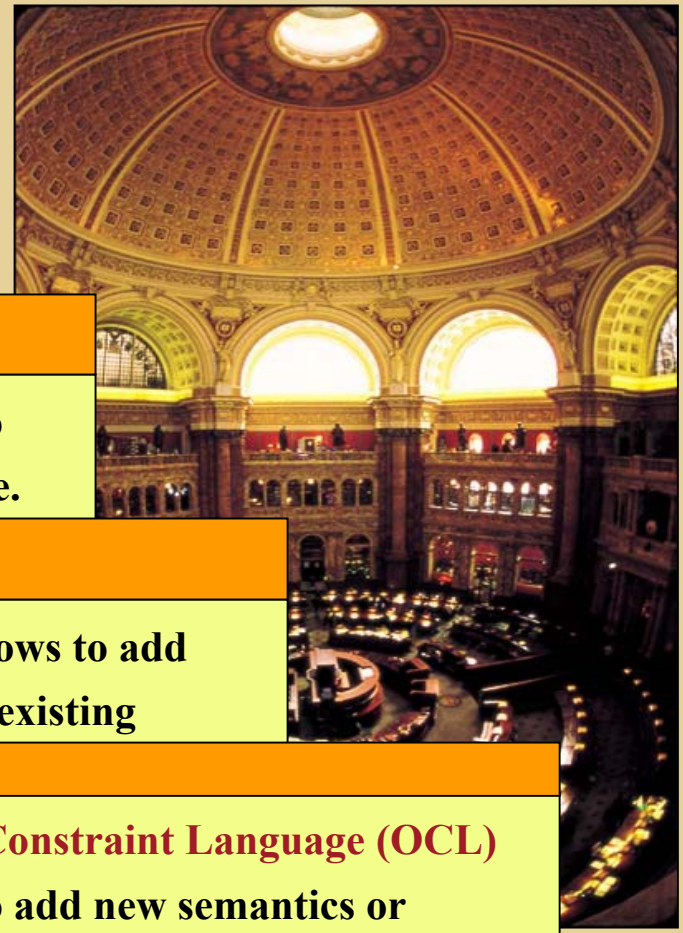
**The possibility to extend a language in a formal way gives you the potentiality to express everything : from new concepts to new rules. Since software solutions are always evolving the importance to have an evolving language is mandatory.**
**UML is such a language.**

# Extending the language

## Extending mechanisms

UML gives the possibility to extends the language in three ways:

- defining new basic building elements
- defining new characteristic of existing elements
- defining new rules that relate elements



## Stereotypes

A stereotype allows to define a new metatype.

## Tagged values

A tagged value allows to add new properties to existing blocks.

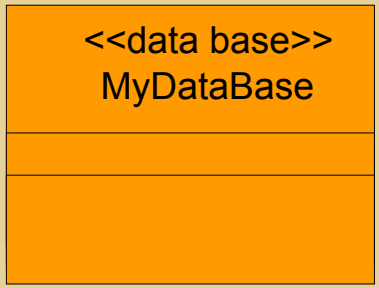
## OCL

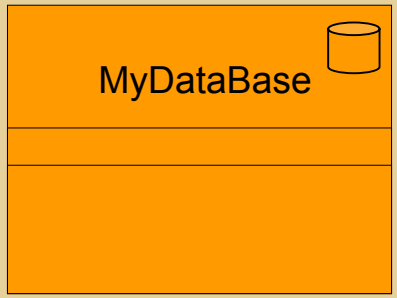
Object Constraint Language (OCL) allows to add new semantics or change existing rules

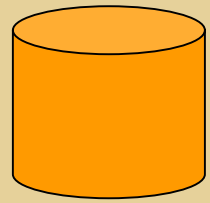
# Stereotypes

**There are three methods for showing and modeling stereotypes:**

- **showing the stereotype name enclosed within guillemets**
- **using an icon inside the original symbol**
- **using only the icon as a building block of UML**

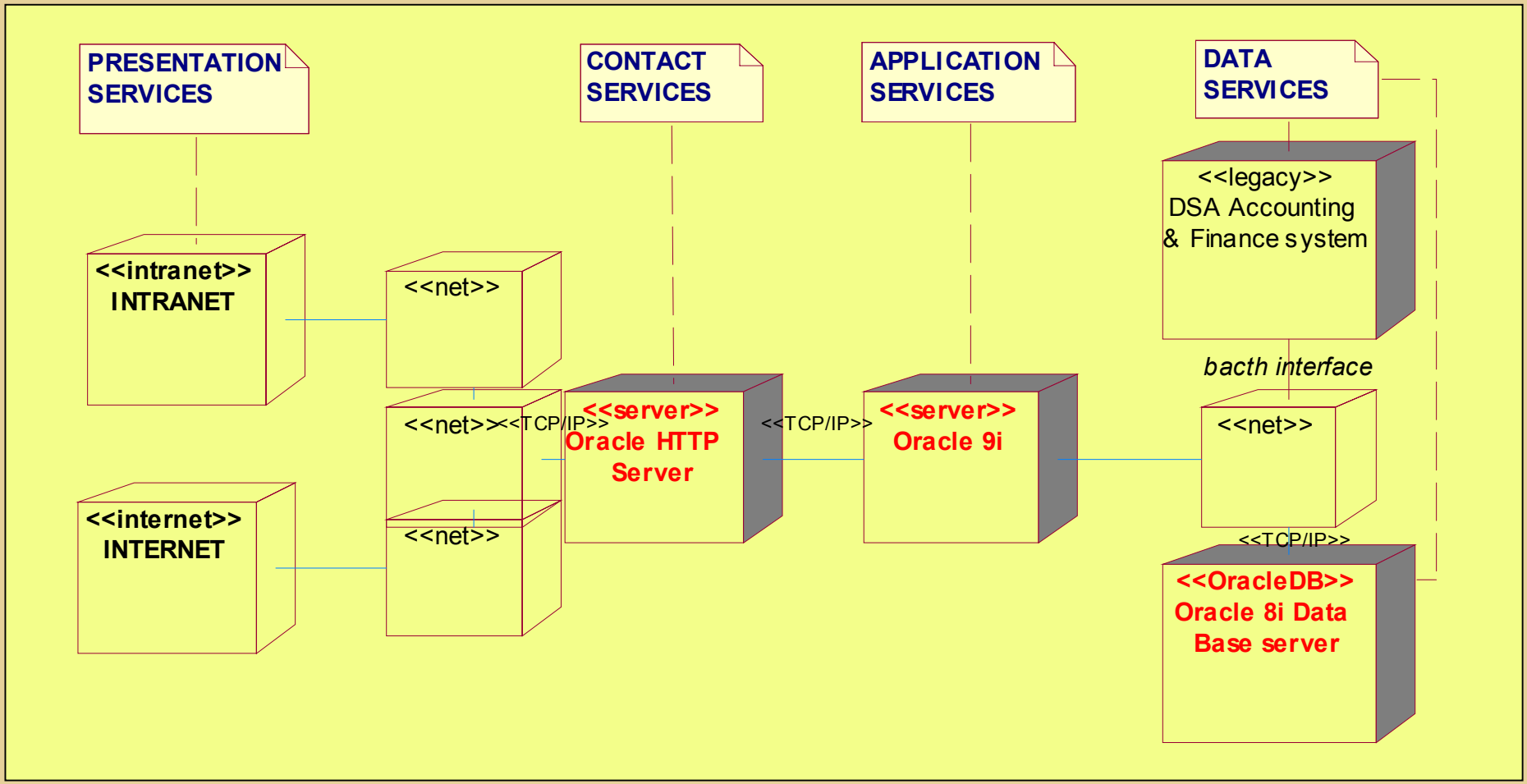


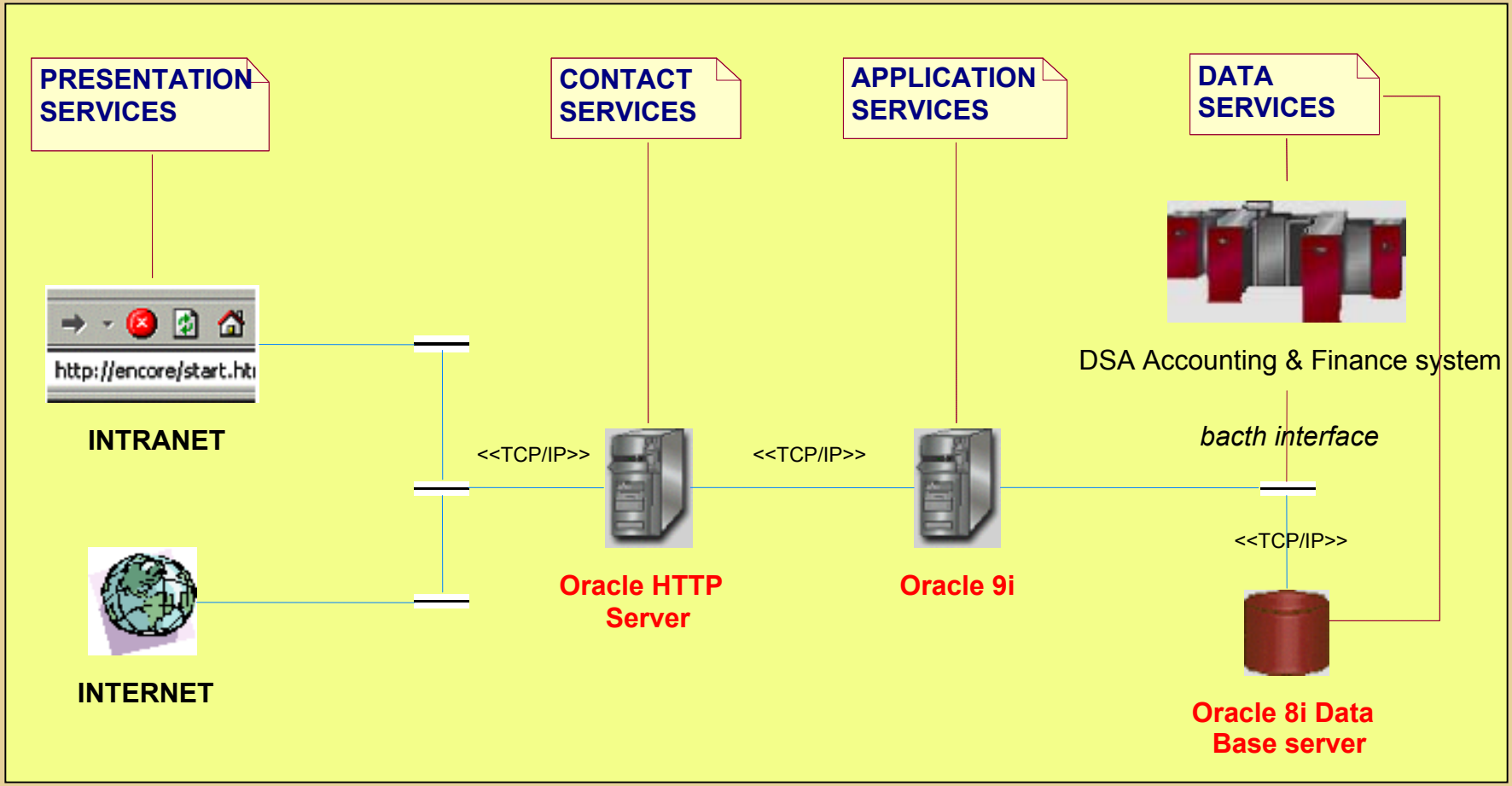



# Stereotypes

**A more complex example...**

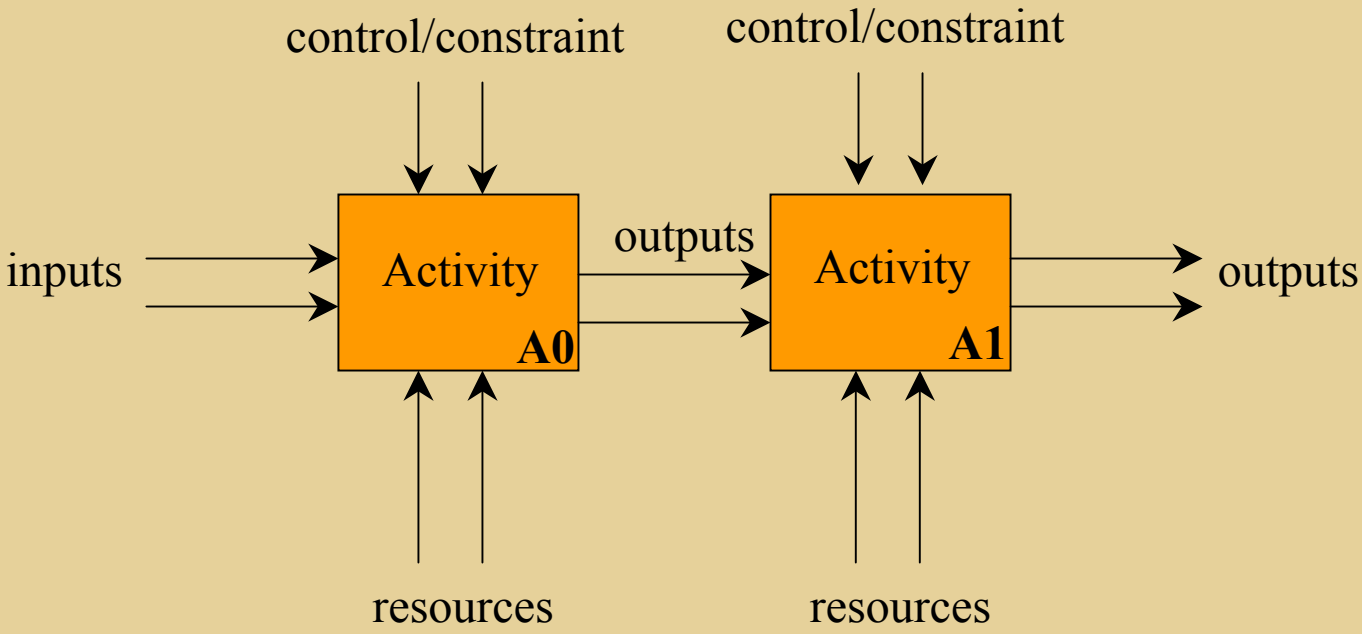


# Stereotypes



# From IDEF0 to UML

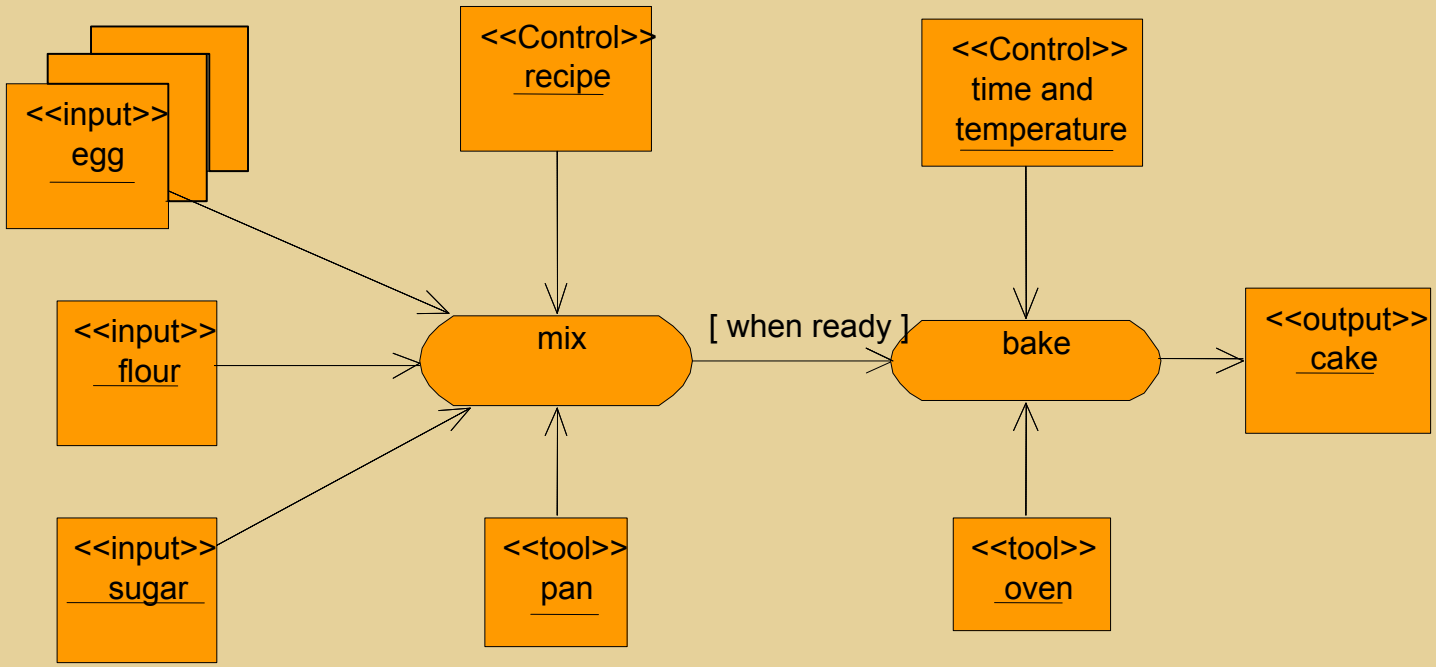
**IDEF0 is a well know notation to represent business process.**




**Each activity can be "exploded" in sub activities with the same basic structure.**

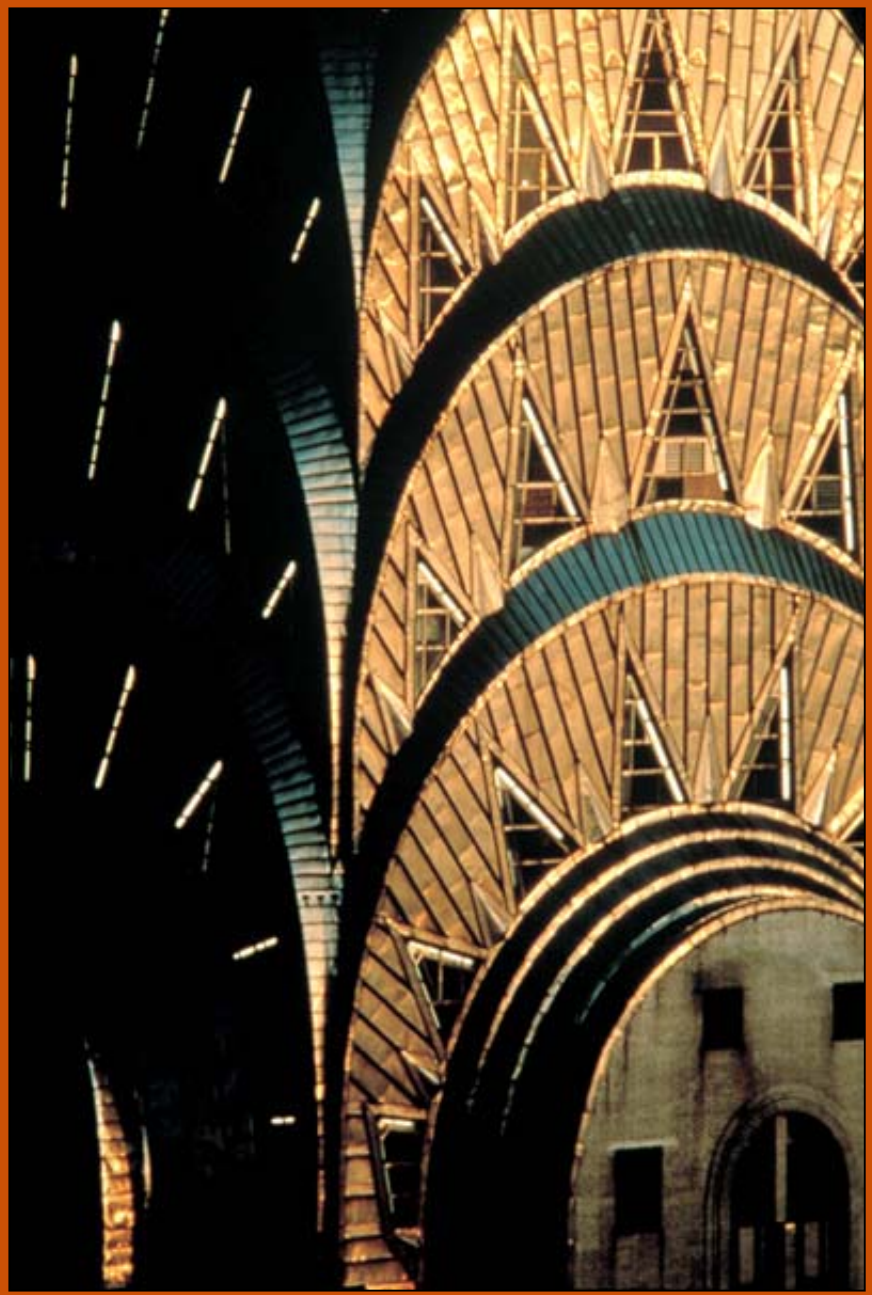
# From IDEF0 to UML

**It is possible to represent an IDEF0 diagram using an activity diagram and some stereotypes.**



# Where you can use UML

- **Display the boundary of a system & its major functions using use cases and actors**
- **Illustrate use case realizations with interaction and sequence diagrams**
- **Represent a static structure of a system using class diagrams**
- **Model the behavior of objects with state transition diagrams**
- **Reveal the physical implementation architecture with component & deployment diagrams**
- **Extend your functionality with stereotypes**



# *How to use UML: Architecture*

**Architecture is the set of significant decision about:**

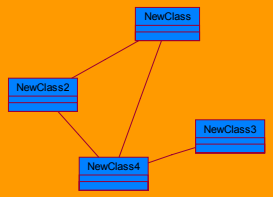
- **The organization of a software system**
- **The selection of the structural elements and their interface by which the system is composed**
- **Their behavior, as specified in the collaboration among those elements**
- **Their usage and functionality**

**To better understand the system it is important to view it from a number of different prospective. To define prospective we model the system architecture.**

# Modeling System's Architecture

Analysts/Designer

**Logical view**

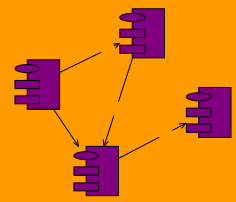


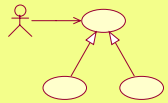

*Structure*

Programmers

**Implementation view**

*Software Management*



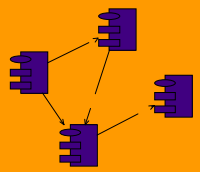


End User

**Use Case view**

*functionality*

System integrators



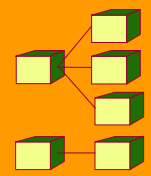
**Process view**

*Performance*
*Scalability*
*Throughput*

System engineering

**Deployment view**

*System topology*
*Delivery*
*Installation*
*Communication*

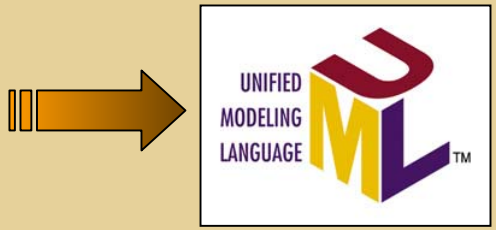


**4+1 View Architecture**

# Use Case view

## What/Where

**The use case view of a system encompasses the use cases that describe the behavior of the system as seen by its end user, analyst and tester. This view doesn't really specify the organization of a software system. Rather, it exists to specify the forces that shape the system's architecture.**



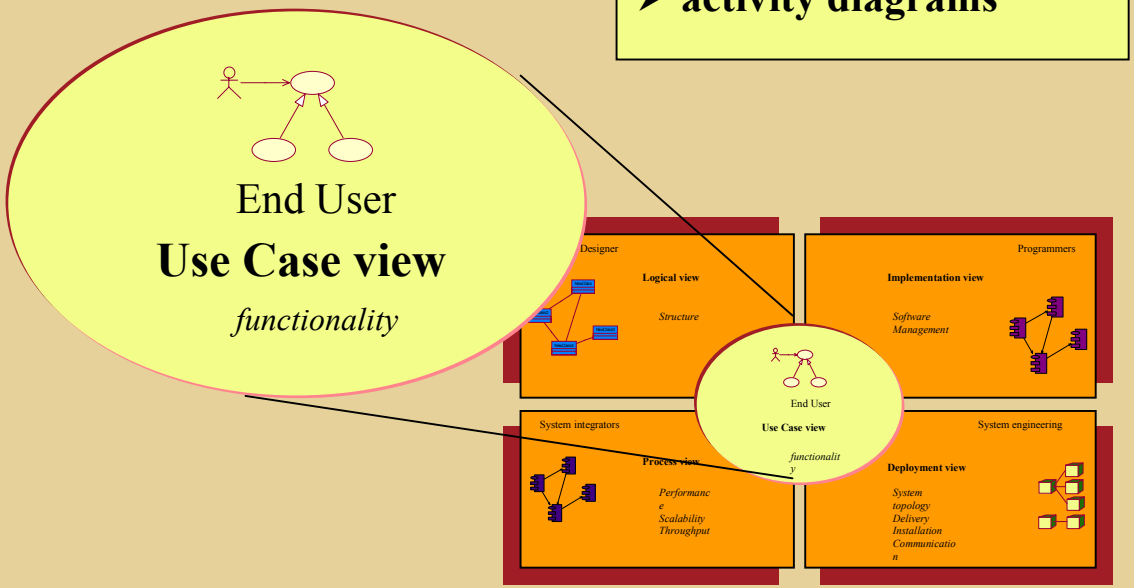

## Diagrams

***Static aspect:***

- **use case diagrams**

***Dynamic aspect:***

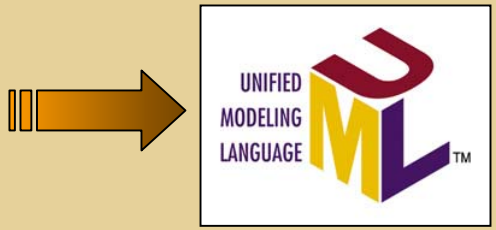
- **interaction diagrams**
- **statechart diagrams**
- **activity diagrams**



# Logical view

**What/Where**

**The logical view of a system encompasses the classes, interface, and collaborations that form the vocabulary of the problem and its solutions. This view primarily supports the functional requirements of the system, meaning the services that the system should provide to its end users.**



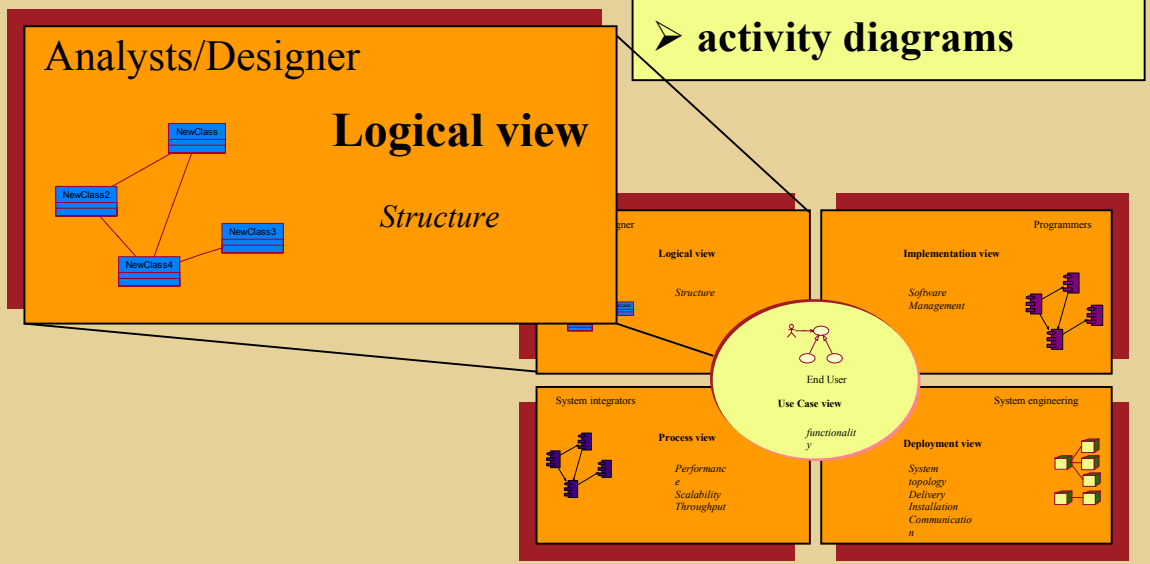

**Diagrams**

***Static aspect:***

- **class diagrams**
- **object diagrams**

***Dynamic aspect:***

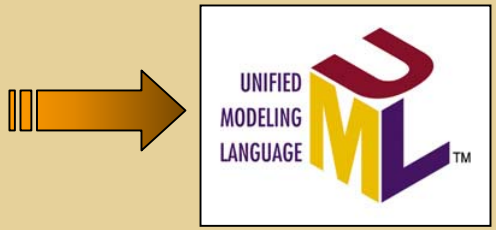
- **interaction diagrams**
- **statechart diagrams**
- **activity diagrams**



# Process view

**What/Where**

**The process view of a system encompasses the threads and processes that form the system's concurrency and synchronization mechanisms. This view primary addresses the performance, scalability, and throughput of the system.**



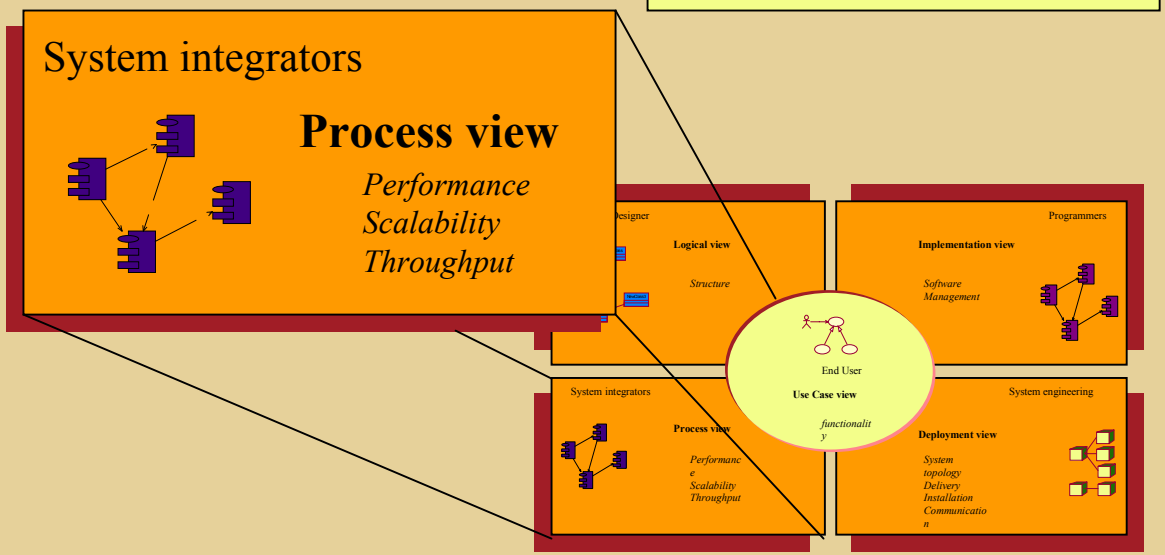

**Diagrams**

***Static aspect:***

- **component diagram**

***Dynamic aspect:***

- **interaction diagrams**
- **statechart diagrams**
- **activity diagrams**



# Implementation view

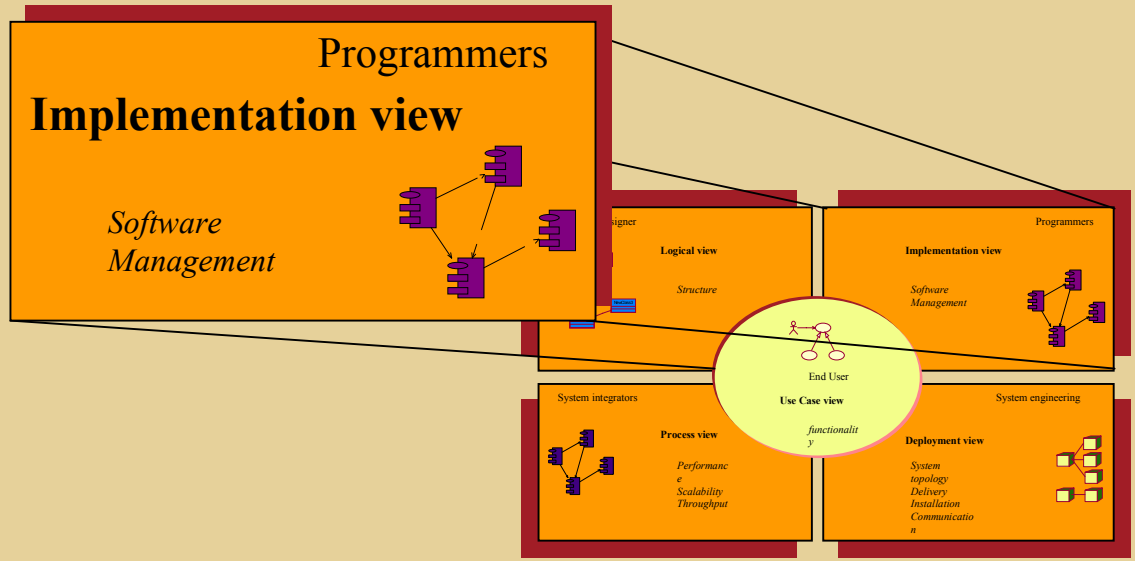
**What/Where**

**The implementation view of a system encompasses the components and files that are used to assemble and release the physical system. This view primarily addresses the configuration management of the system release, made up of somewhat independent components and files that can be assembled in various way to produce a running system.**

**Diagrams**

***Dynamic aspect:***

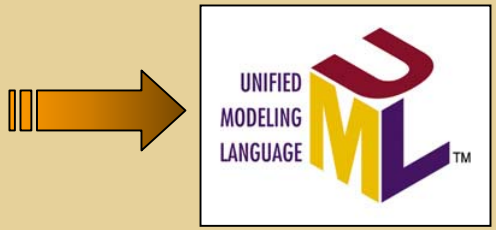
- **interaction diagrams**
- **statechart diagrams**
- **activity diagrams**



# Deployment view

**What/Where**

**The implementation view of a system encompasses the nodes that form the system's hardware topology on which the system execute. This view primary addresses the distribution, delivery, and installation of the parts that make up the physical system.**



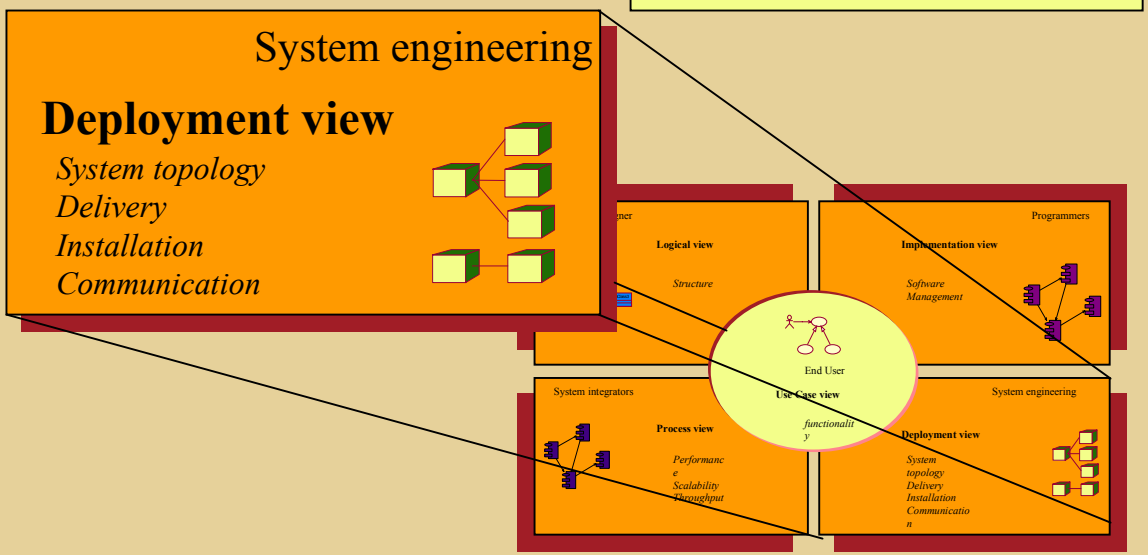

**Diagrams**

***Static aspect:***

- **deployment diagram**

***Dynamic aspect:***

- **interaction diagrams**
- **statechart diagrams**
- **activity diagrams**

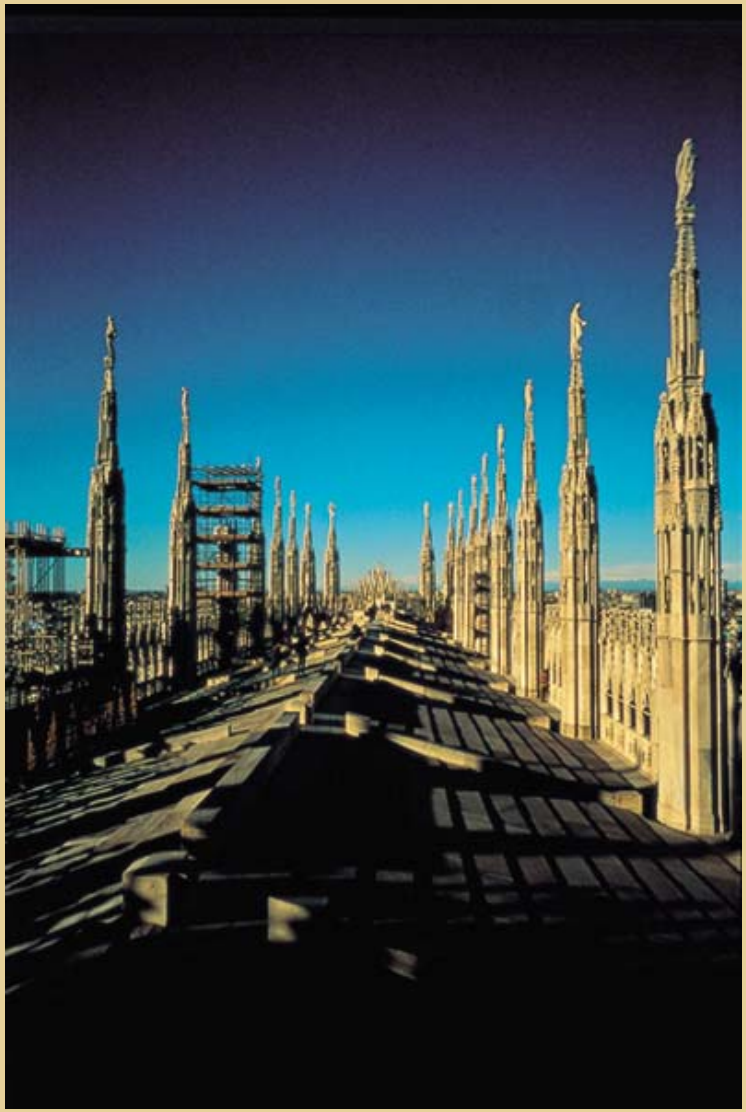


# Building the model...

**Views and language**

- Each of the five views can stand alone so that different stakeholders can focus on the issues of the system's architecture that most concern them.
- These five views also interact with one another.
- The UML permits you to express every one of these five views and their interactions

# Conclusions



- **UML is a Language that can help anyone (business analyst, developers, consultants…) to Model almost anything (processes, functional requirements, software solutions, …) in a Unified matter.**

- **A set of "core" diagrams have been defined and standardized by OMG (ISO in 2001 ?), but new diagrams, properties and rules can be defined by extending the language.**