SOFTWARE ENGINEERING MODELS AND MODELING

ANTTI KNUTAS (D.SC.)

Course Q&A

Please ask any question (or post them in Echo360 lecture Q&A area)

Course Q&A

I'll address the first question here: What events and attendance options we have?

Interactive events

- Lectures, in Lappeenranta (you can replace these by watching videos)
- 2x long seminars in Lahti
- Exercises, on both campuses (plus online)

There's a tiny point bonus from coming to work with us on these.

Materials

- Online lectures
- Coursebook
- Other links

How we'll proceed today

- I'll introduce the topic with a short, 30 to 45min lecture
- Then, we'll proceed to work on analyzing the course project
- Lastly, groups are welcome to post their analysis on the course project assignment *before the end of day today* (for a small point reward)
- There will be more in-depth videos posted online (on UML and modeling)

The small point rewards from interactive lectures, weekly exercises, and seminars overlap => it is enough to do one, or part from each

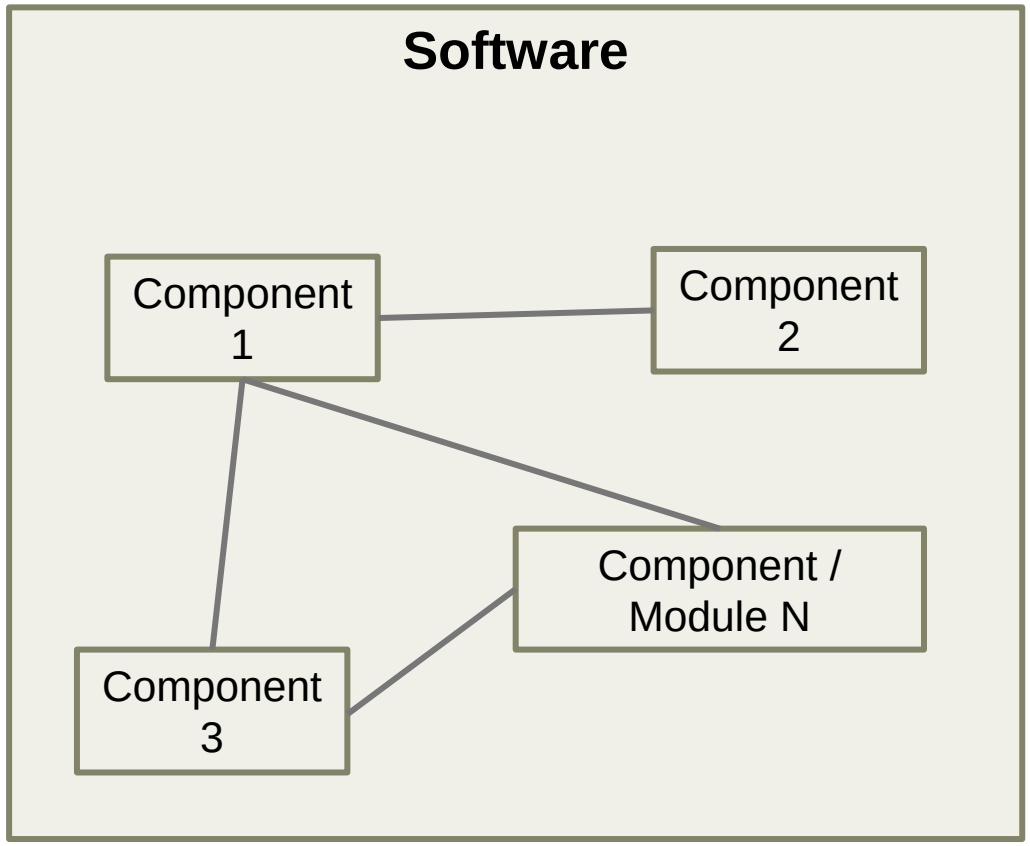
Weekly assignments are the main "big effort" before the course project. (10%)

LUT University

MODELING SOFTWARE WITH OBJECTS

LECTURE 3

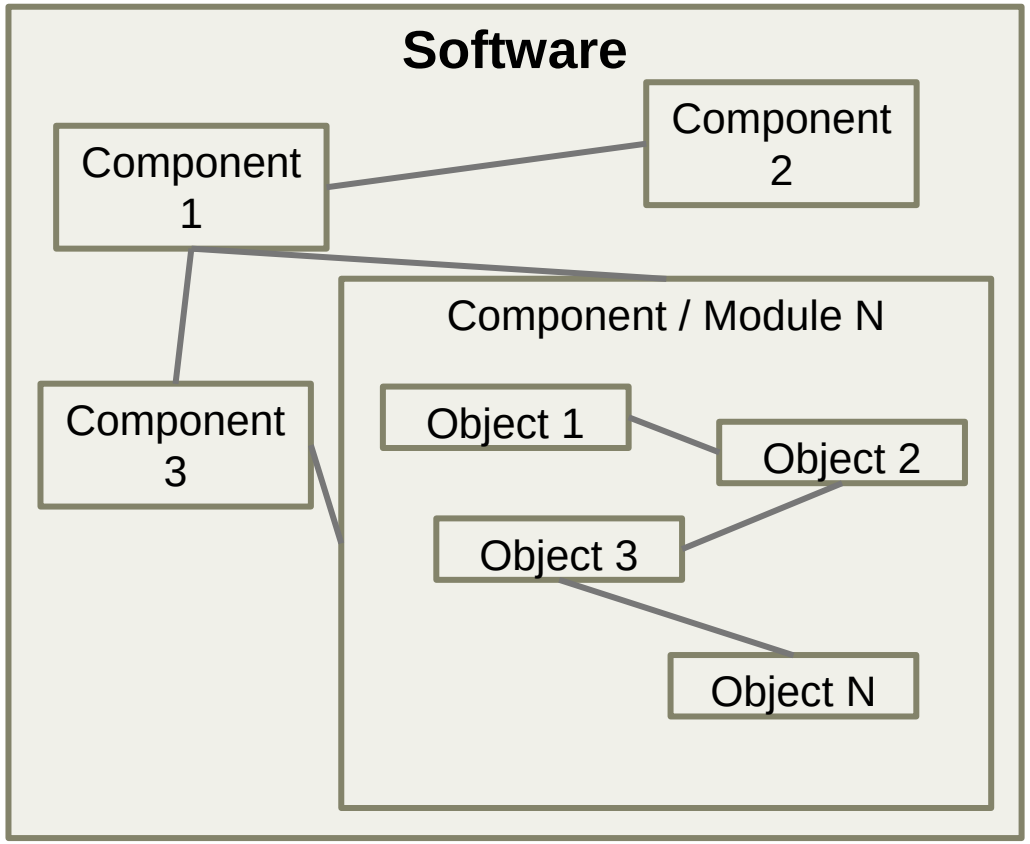
WHAT IS SOFTWARE?



Software: A compilation of interconnected components (also called modules) which exchange data to get things done.

Component (also module): A part of software system, usually responsible for managing one activity or role such as GUI drawing, event listener, networking etc…

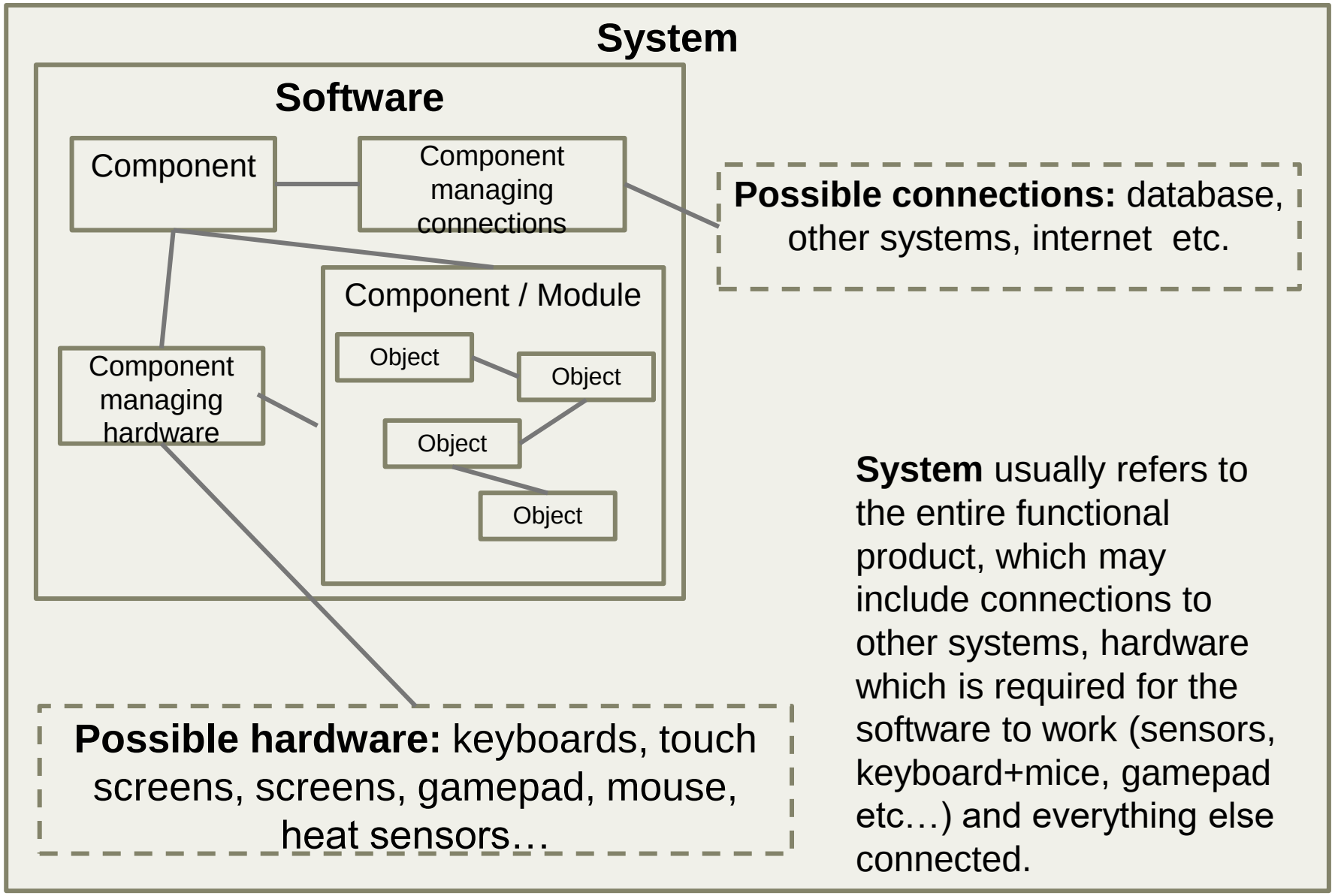
WHAT ARE OBJECTS?



Components (or modules) are also formed from a group of objects, which exchange information to enable the component to work**.**

Objects are formed based on defined classes, which are written in source code, and created by programming work to do **routines** and manage **messages**.

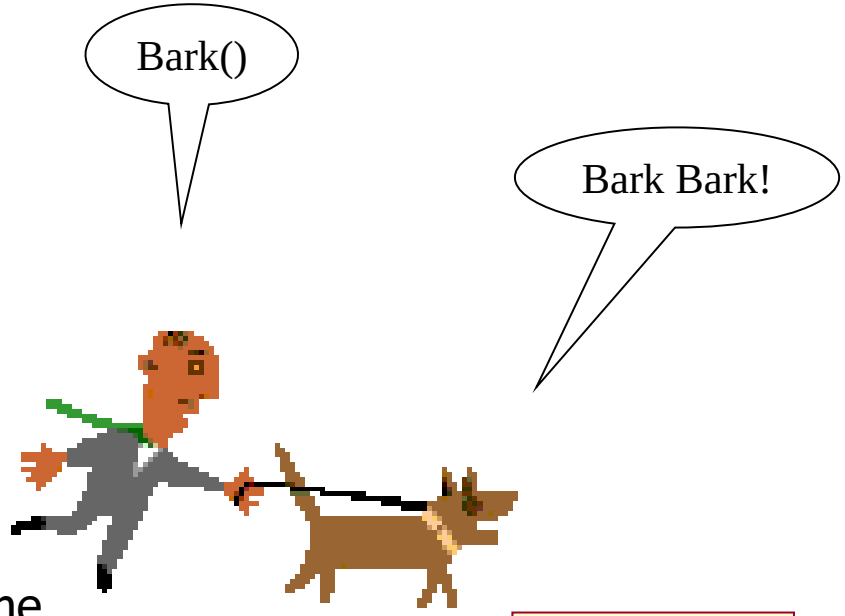
WHAT IS SYSTEM?



Review: object oriented concepts and thinking

- What is an object and what is a class?
- Messages
- Inheritance
- Polymorphism
- Abstraction
- Uses for objects
- Benefits and drawbacks of object orientation
- Object oriented technologies

Class or object?



Class
Attributes, state variables
Services, methods

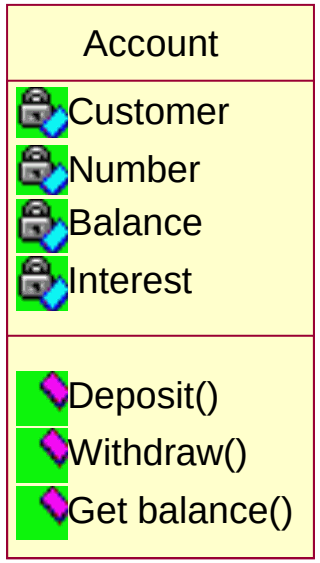
- Class
  - A set of similar objects that have same attributes and behavior
- An object has
  - identity
  - privacy (it acts independently)
  - class
  - state (attribute values ~ member variables)
  - behavior (services ~ operations ~ methods)

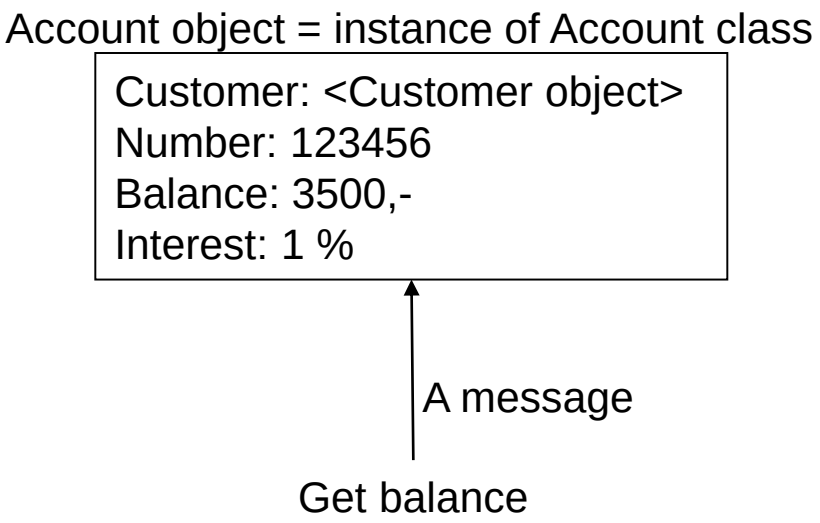
Fido : Dog

Name: Fido
Age: 3 years

Dog
Name Age
Bark()

Objects react to messages





- A message causes an operation execution in an object
  - Operations are stored in class definition
  - Operations are common to all class instances (objects)
  - Each object has its own state = attribute values
  - Method = operation implementation

Encapsulation and information hiding

- Information hiding
  - Hiding of *design decisions* in a computer program that are most likely to change, thus protecting other parts of the program from change if the design decision is changed (Parnas, 1972)
  - A module should implement and hide only one design decision and reveal as little as possible about its inner workings and data
- Related concepts
  - Cohesion
  - Coupling

Cohesion

- The extent to which an individual module is self-contained and performs a single well-defined function
- High cohesion
  - The module has strongly related and focused responsibilities
- Low cohesion
  - The responsibilities are varied and use unrelated sets of data

Coupling

- The degree of interdependence of modules
- High coupling
  - A module modifies or relies on the internal workings of another module
  - Change in a module requires a change in the dependent module
- Low coupling
  - A module interacts with another module through a stable interface and does not need to be concerned with the other module's internal implementation
  - Change in a module does not require a change in the other module

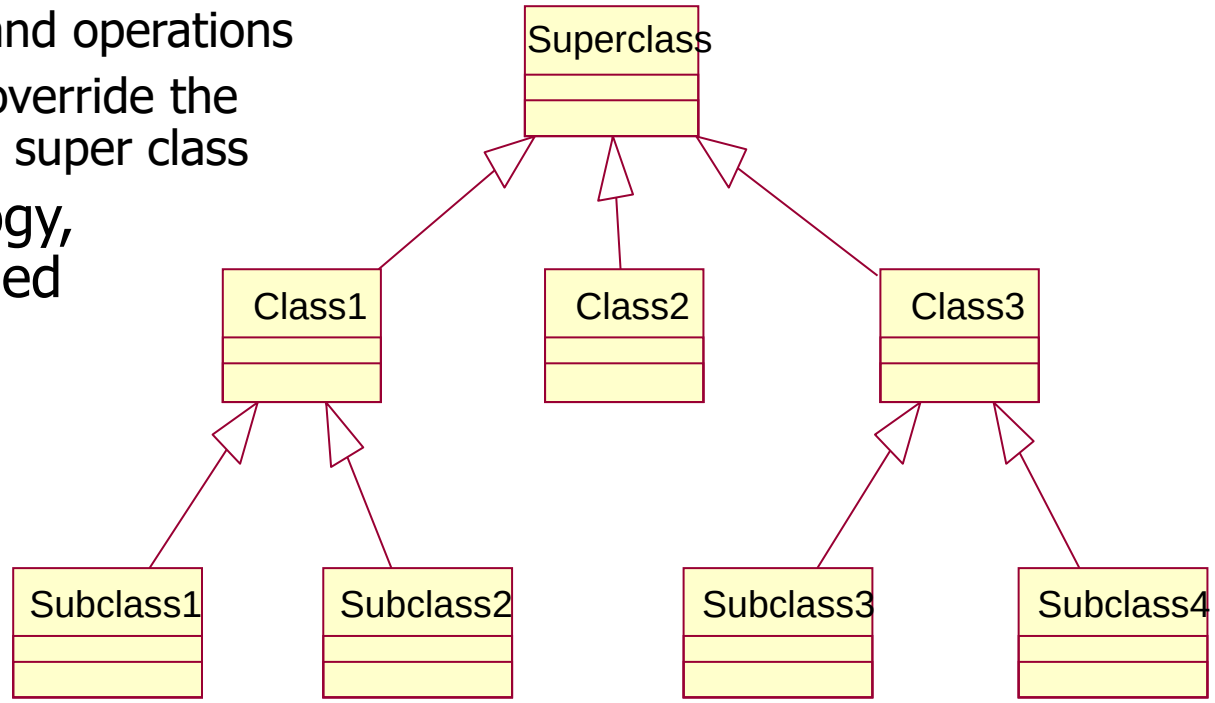
Encapsulation

- A well-designed module should have highest possible cohesion and lowest possible coupling
- Encapsulation
  - An object oriented term for information hiding
  - Binding of data and methods into an object
  - The internal structure of an object is hidden from other objects
  - The object has an interface that it reveals to other objects – everything else is hidden
  - Aims at low coupling and gives possibilities for high cohesion

Inheritance/generalization

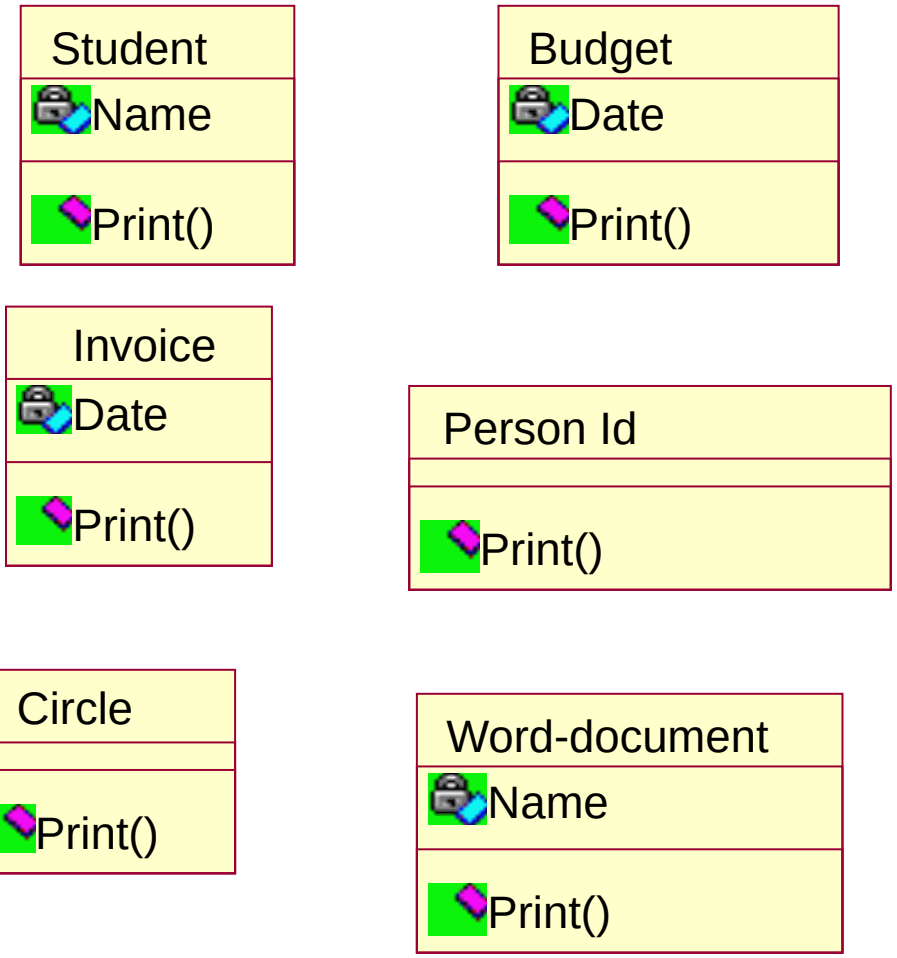
- A subclass inherits all properties of a super class: attributes and operations
- In addition, a subclass has its own attributes and operations
- A subclass can override the operations of its super class

- In UML terminology, inheritance is called generalization



Properties are polymorphic

- Polymorphism
  - The process of using an operator or function in different ways for different given inputs
  - A message does not need to know how the operation will be performed = low coupling
  - Objects of several classes can react to the same message
  - Objects of several classes may have the same attribute



Classes are abstractions

	Real world	Model
Abstract	Concepts	Classes Relationships
Concrete	Events Things Phenomena	Objects/instances Attribute values

- In object oriented design, models are made from the problem domain (often the real world)
  - Business models, data structures, conceptual models, etc

DESIGN WITH OBJECTS

So why did this Software Engineering –course turned into an intro to Object-Oriented programming?

Because objects are a tool which provides us the functionality to design without implementation!

There is no inherent shape, form or look for software component, class or object, so we need a language to express our designs.

Abstraction

- Concentration on essentials
  - Leave out inessentials
  - Purposeful simplification of things
- Forming of general concepts by simplifying things
- Modeling
- Main ways of abstraction
  - Classification – individualization
    - ”Fiat 500, reg.no GLL-86” is an instance of ”Car”
  - Generalization – specialization
    - A ”Car” can be specialized to a ”Sports car”
  - Aggregation (combination) – separation
    - A ”Car” is composed of four ”Wheels”, ”Body”, and ”Engine”

An object has an interface and it encapsulates its properties

- The interface encapsulates object properties
- Information hiding: the data is not accessed directly but through the defined operations in the interface
- Hiding of complexity from the class user
- Enables better change management and reuse
  - Under the interface, the implementation is free

Benefits of object-orientation

- Use of natural human ways of thinking
  - Real world vs. model
- Encapsulation helps change management
- Reuse becomes easier through inheritance and encapsulation
- Iterative development becomes possible
  - Design first the interfaces, then implement the functionality iteratively
- Use of same basic concepts through the development life-cycle
  - From requirements to implementation one speaks about classes and objects

OO technologies: OO programming languages

- Pure OO languages
  - All variables except basic data types (int, float, char, etc.) are objects
  - Typically there is one class hierarch with the root class Object
  - Java, C# (.NET framework) Smalltalk, Eiffel
  - Garbage collection: automatic memory management
  - Late binding
- Hybrid languages
  - Often extensions to more traditional languages
  - It is possible to program both in OO way and procedurally
  - No automatic memory management
  - C++, Python, Visual Basic, Object Pascal (Delphi), Modula, Ada

UML

UNIFIED MODELING LANGUAGE

Object-oriented analysis and design / history

- In the 1980's it was noted that there is a gap between different parts and phases of the development lifecycle
  - A conceptual database model and a hierarchical functional model were often designed separately
    - Or: only the other was really designed
  - The database model and functional model did not correspond to each other
    - The functional model relied on wrong data structures
    - The database model did not meet the requirements of the functionality
  - Gap between requirements specifications and designs

History continued

- It was believed that with object oriented concepts these gaps would be smaller
  - Class and object are quite natural concepts to humans (concepts, things, objects, events, etc.)
  - Service thinking: what services are offered by an object?
  - Message thinking: what messages does an object understand and what it reacts to?
  - Abstraction
  - Distributed applications and their interfaces brough a natural environment for object-oriented analysis and design
    - Service interfaces
    - Independently functioning objects

Benefits of object-oriented analysis and design

- Usually, the following ones are seen as the benefits
  - The same basic concepts are used in all phases
    - Object, class, service, message
  - One description language that becomes more detailed during the life-cycle
    - Class diagrams, UML
  - It is easy to use different service layers that can be later distributed
    - E.g. User interface, application logic, database services
  - Enables the hiding of complexity under the designed interface
    - Concentration on problem solving
    - Easy-to-use interfaces

UML

- UML
  - Unified Modelling Language
  - Three OO legends, James Rumbaugh, Grady Booch and Ivar Jacobsen joined together (Rational Software Corporation) and developed the UML description language
  - UML has become a global industry standard in the field of OO analysis and design
    - All major companies committed (also in Finland)
    - An open standard defined by the OMG (Object Management Group)
    - http://www.omg.org/uml
  - The development started in 1995, version 2 was accepted October 2004

Characteristics of the UML

- According to the original authors
  - UML supports the whole life cycle from requirements to deployment
  - UML fits to most application areas (embedded systems, administrative systems, etc.)
- Good tool support
  - E.g. Rational software architect (former Rose)
  - StarUML, ArgoUML, … (free)
- Many process models that suit to UML
  - E.g. RUP – Rational Unified Process

WHY?

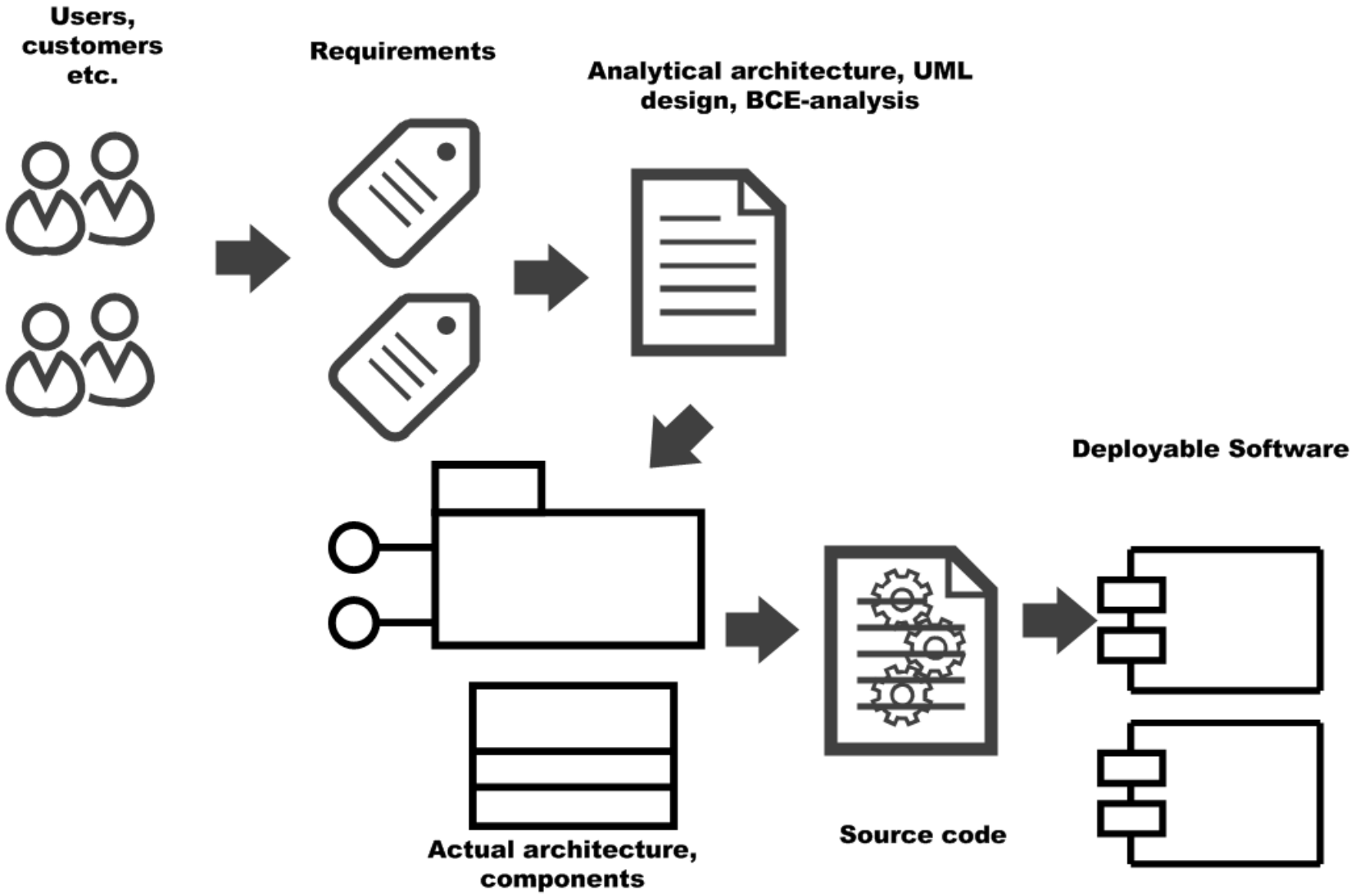
UML is not a development language, it is a design tool which enables knowledge transfer and visual presentation of ideas!

1. **Use cases to define how the system should operate.**
2. **Class diagram to define how the internal parts should be defined.**
3. **Deployment diagram to define the environment in which the system will operate.**

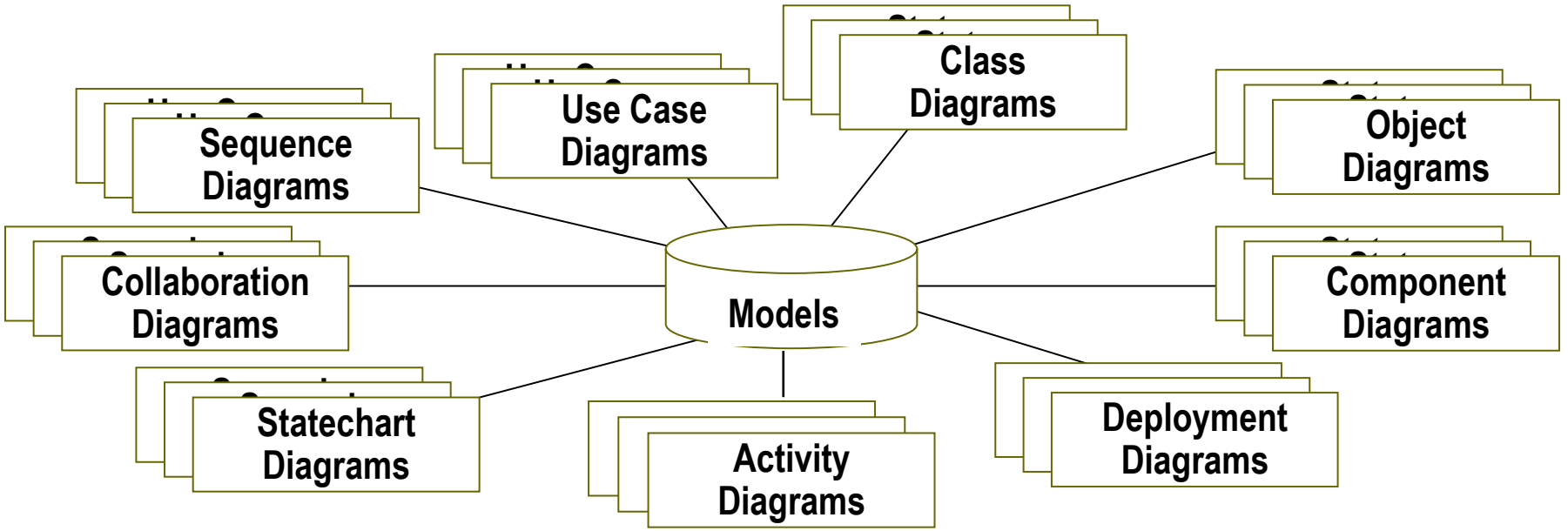
Additionally (if needed)

4. **Sequence diagram to define client-server communication.**
5. **State machines to define embedded system behavior.**

FROM REQUIREMENTS TO IMPLEMENTATION



Modeling techniques in UML



- Above: UML 1.0 techniques
- Short introduction to the techniques follows

UML DIAGRAM TYPES

UNIFIED MODELING LANGUAGE

UML 2 Chart types

Structure diagrams

- **Class diagram**
- **Component diagram**
- **Composite structure diagram** (added in UML 2.x)
- **Deployment diagram**
- **Object diagram**
- **Package diagram**

Behavior diagrams

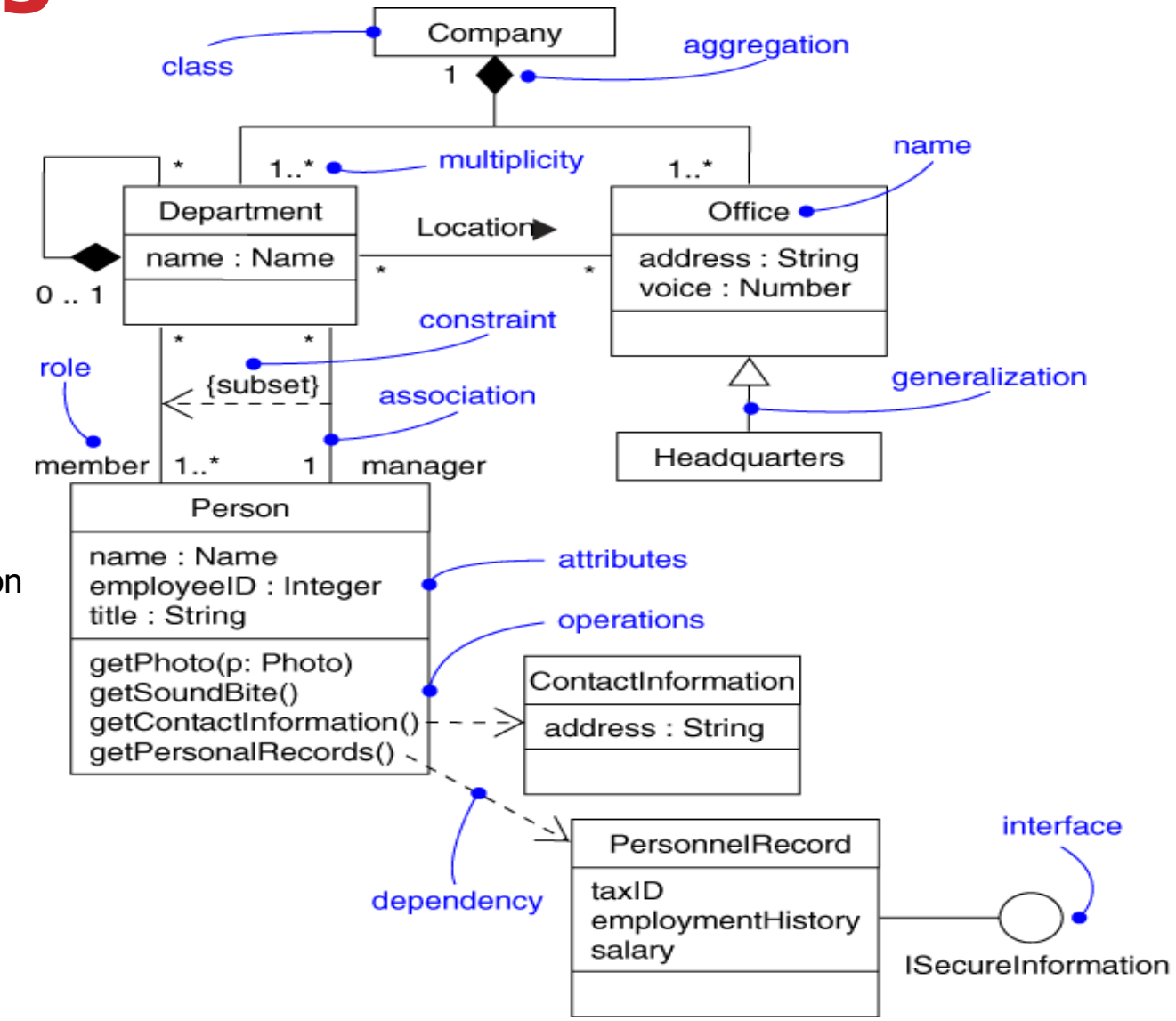
- **Activity diagram**
- **State Machine diagram** (*was Statechart diag. in UML1*)
- **Use case diagram**

Interaction diagrams

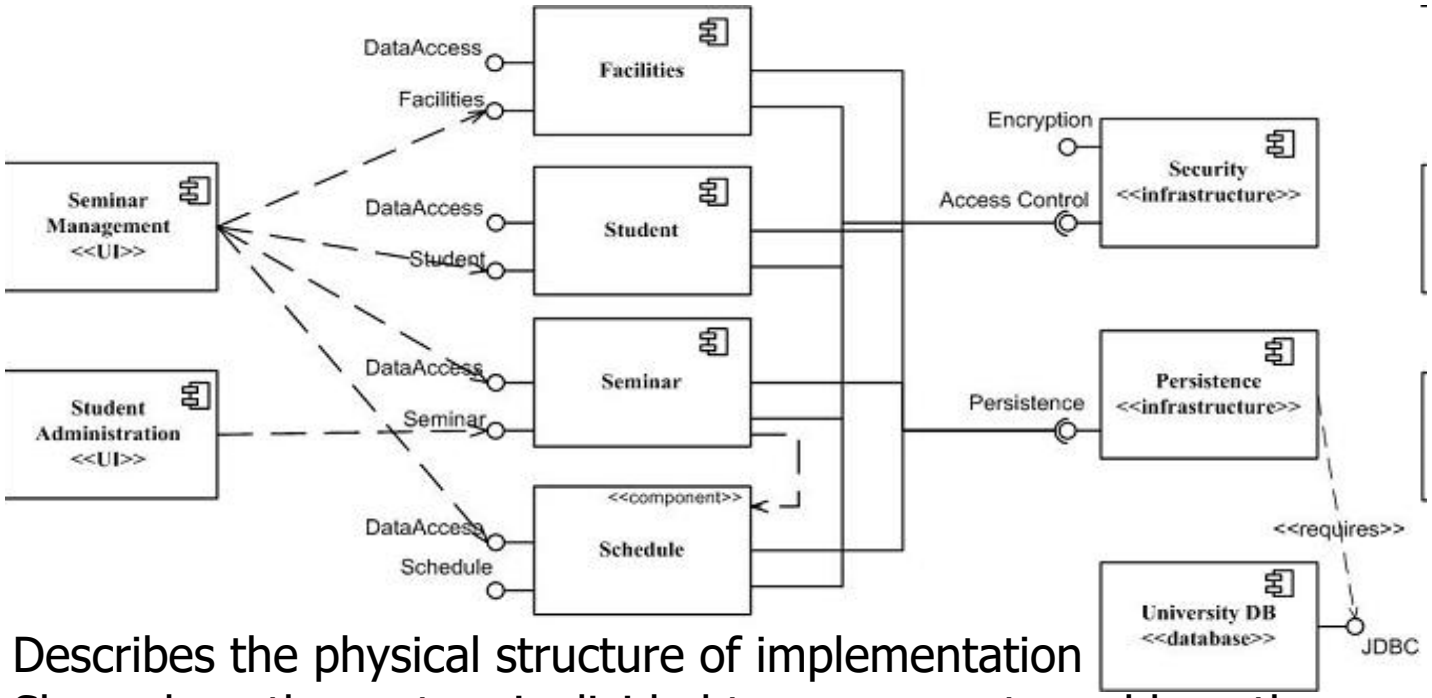
- **Communication diagram** (was *Collaboration diag. in UML1)*
- **Interaction overview diagram** (added in UML 2.x)
- **Sequence diagram**
- **Timing diagram** (added in UML 2.x)

Class Diagram

- Describes the concepts of the system
- Becomes more detailed during the lifecycle
- Purpose is to
  - Name and model the concepts of the system
  - Define collaboration between concepts
  - Define database structure

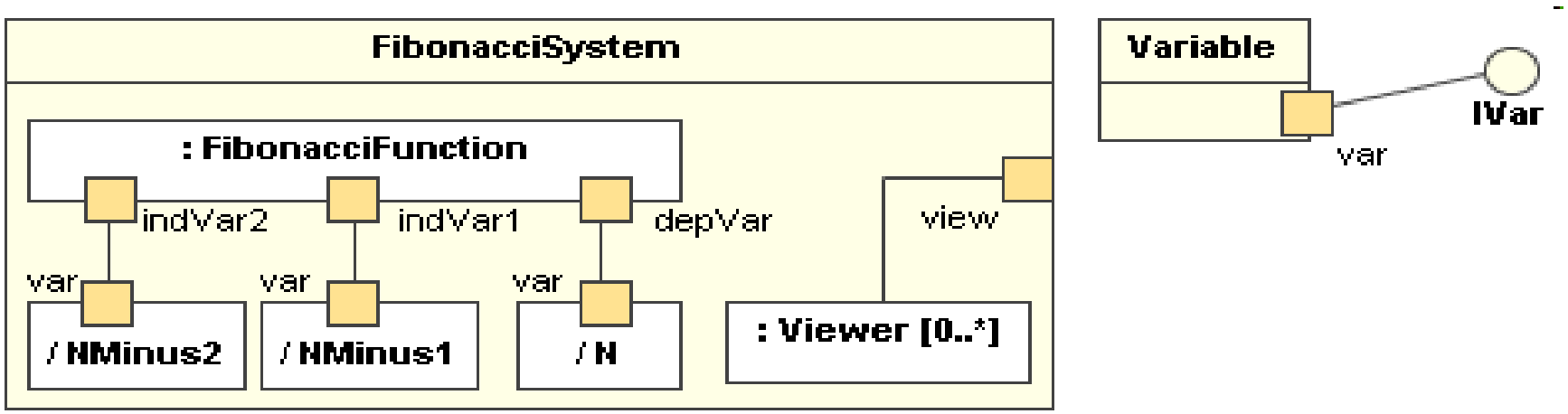


Component Diagram



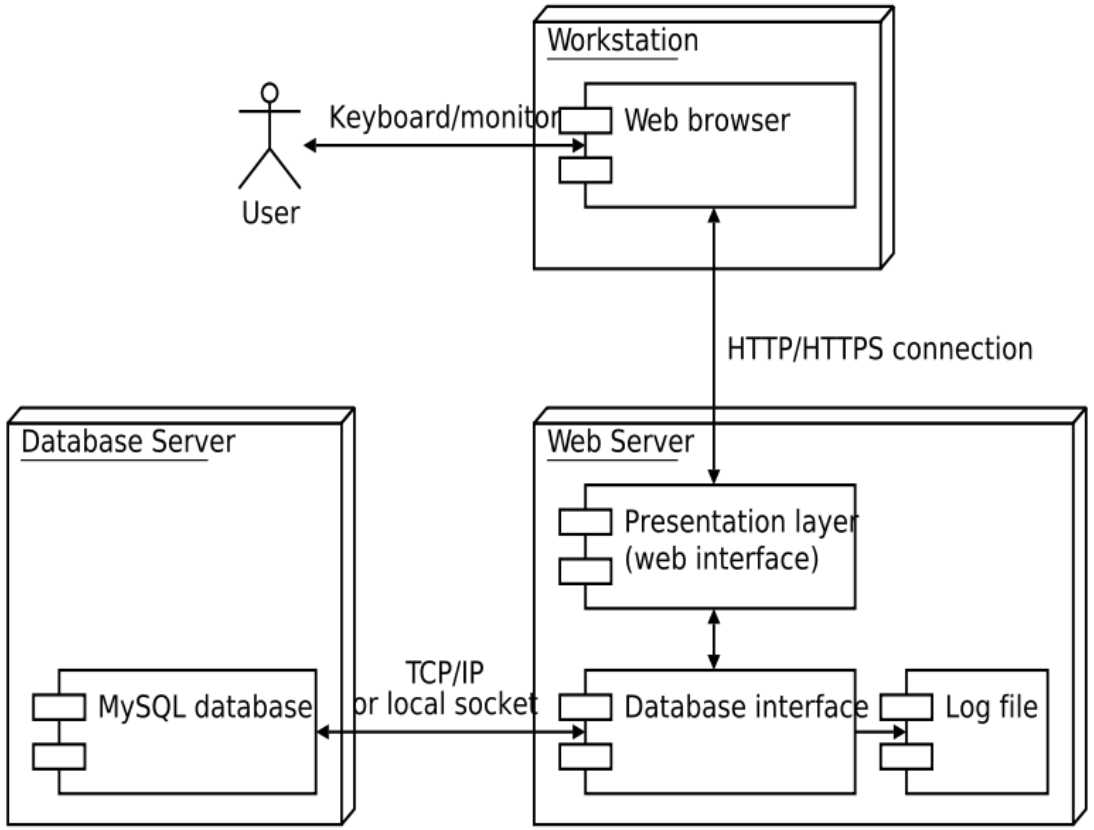
- Describes the physical structure of implementation
- Shows how the system is divided to components and how they are dependent on each other
- Part of architecture description
- Purpose is to organize the source code, build executables and describe the physical database

Composite structure diagram



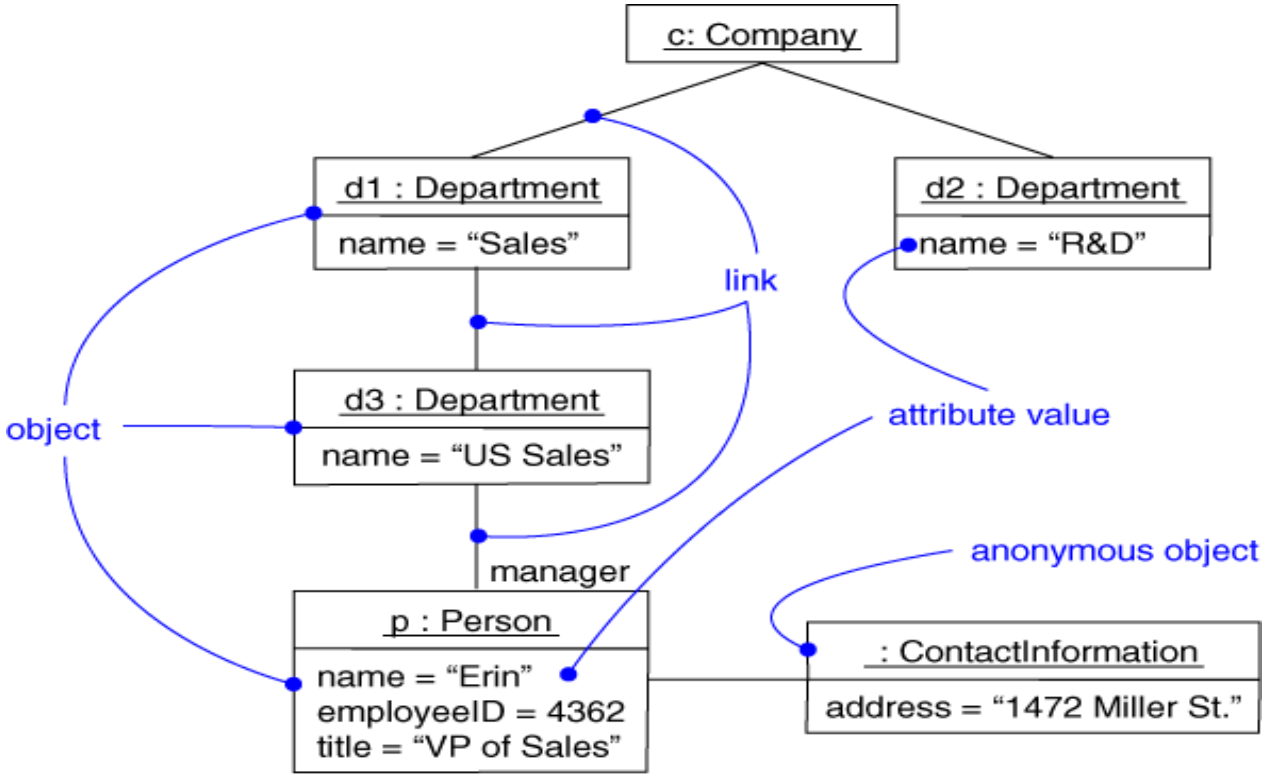
- Models classes, their interfaces and internal structures with a hierarchical notation
- Shows also collaborations between classes
- Typically used for architecture design
- Not a widely used notation – came with UML 2.0

Deployment Diagram



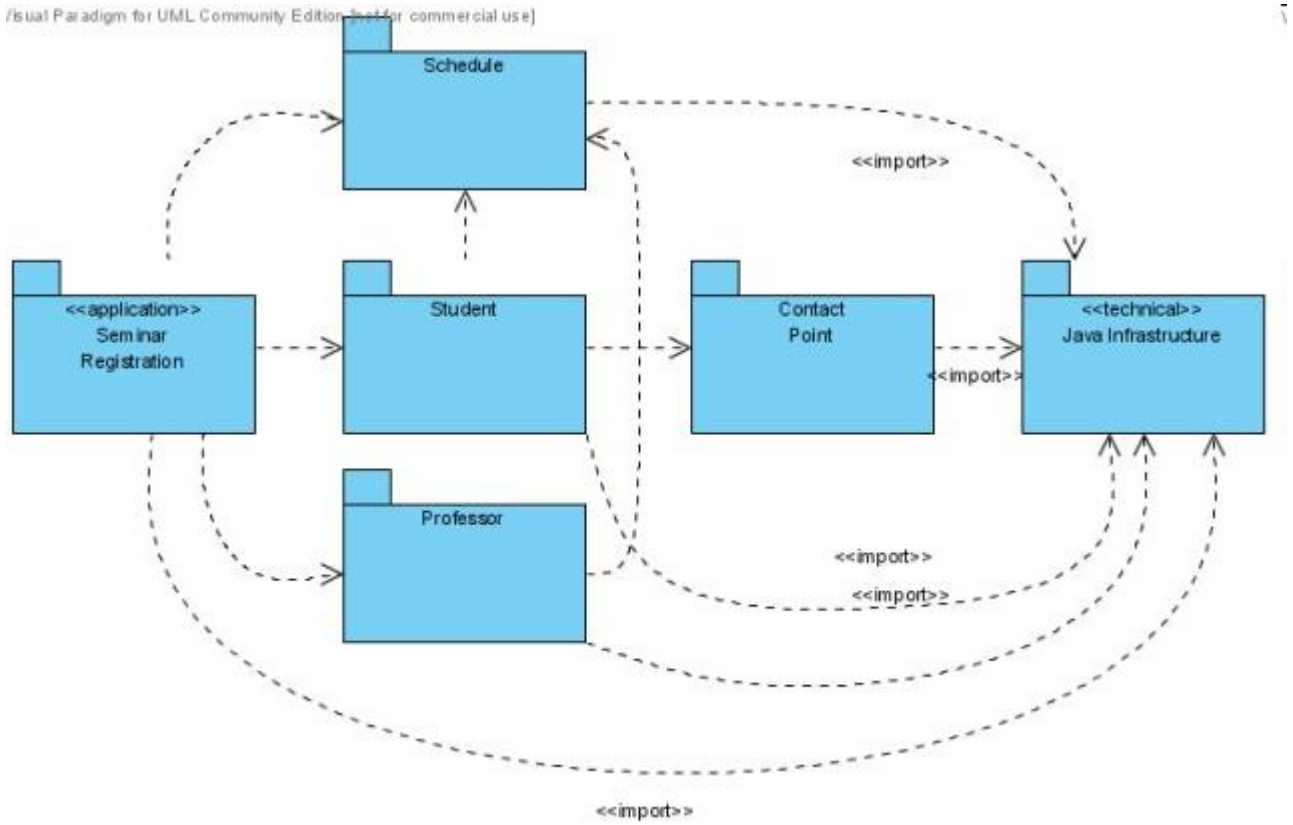
- Describes system's hardware topology
- Part of architecture description
- Purpose is to describe distribution of hardware components and identify the bottlenecks of performance

Object Diagram



- Shows object instances and their links
- Used to description of examples about data structures

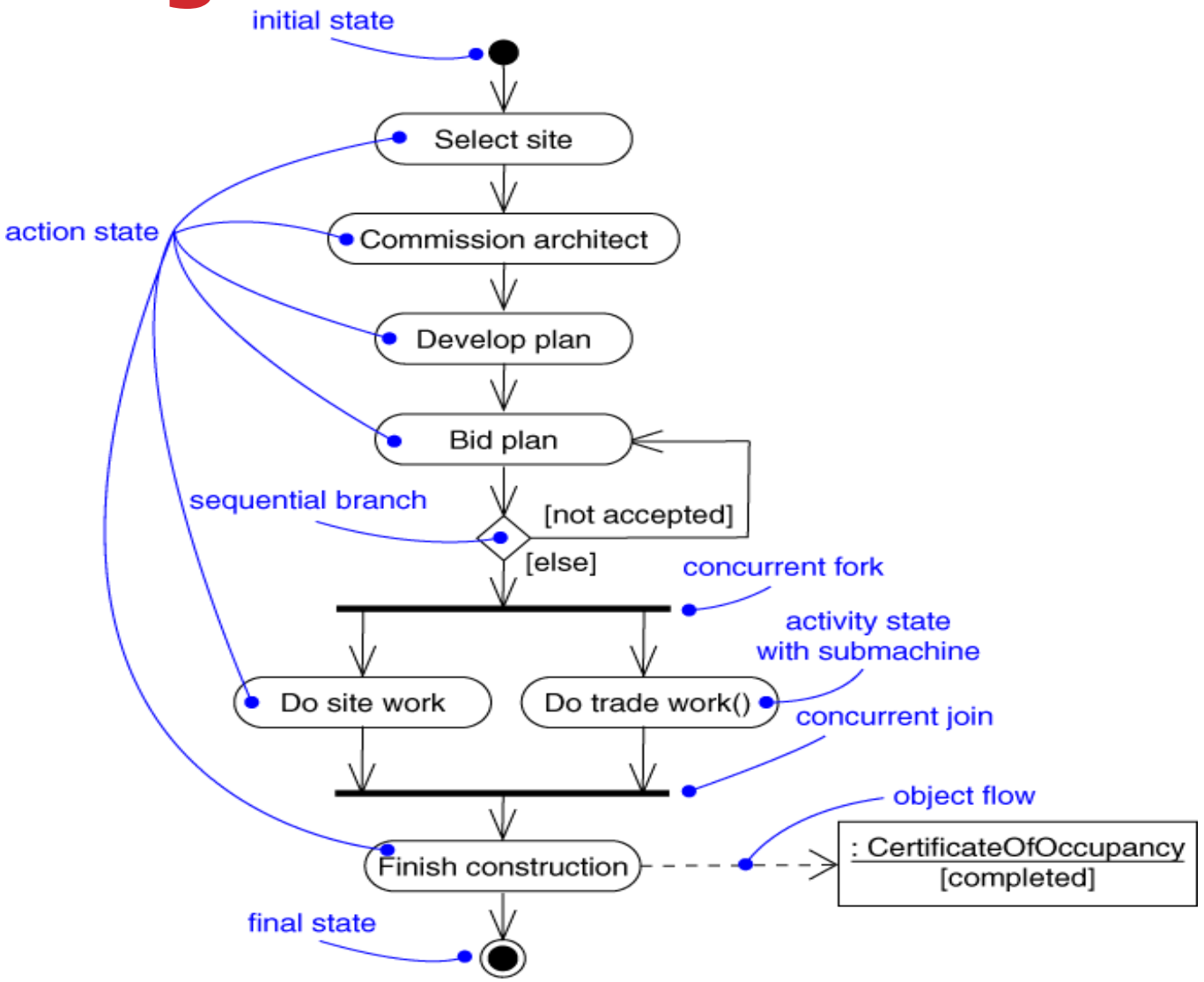
Package Diagram



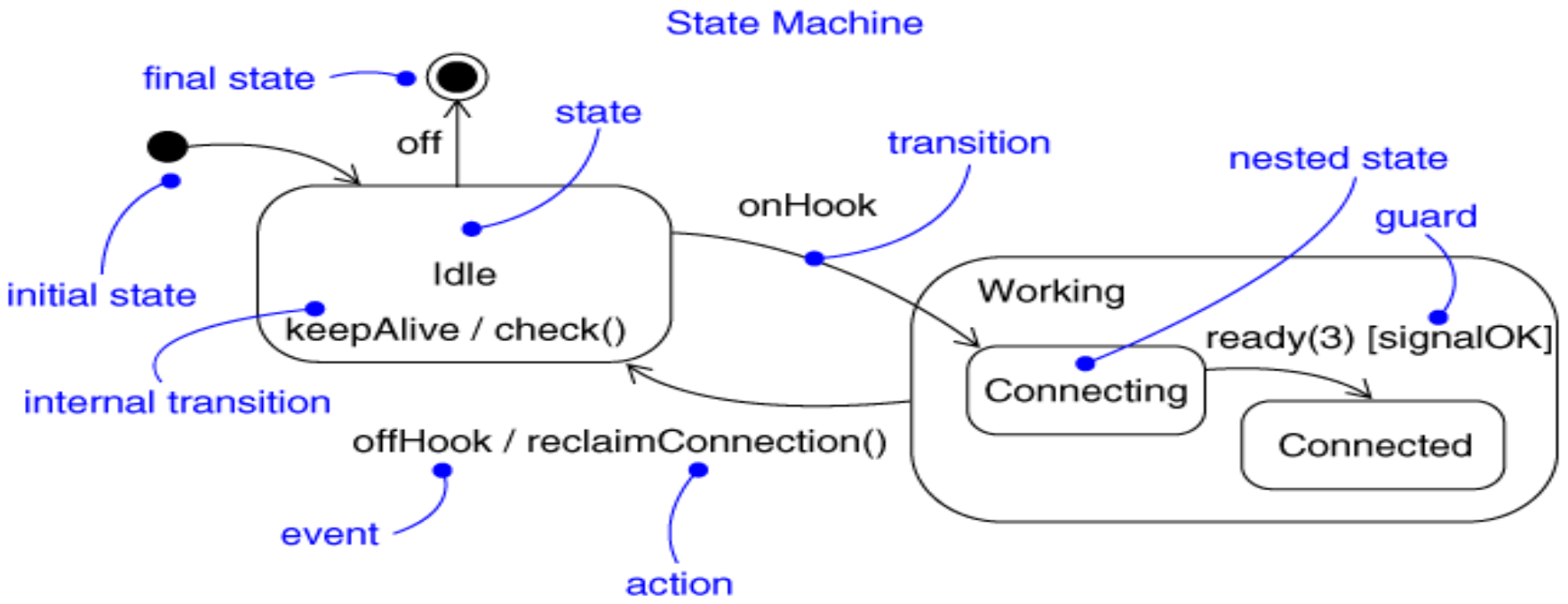
- Shows dependencies between larger blocks of items called packages
- Used in architecture design and specification

Activity Diagram

- Models the dynamic behavior in an event-oriented manner
- Models business workflows

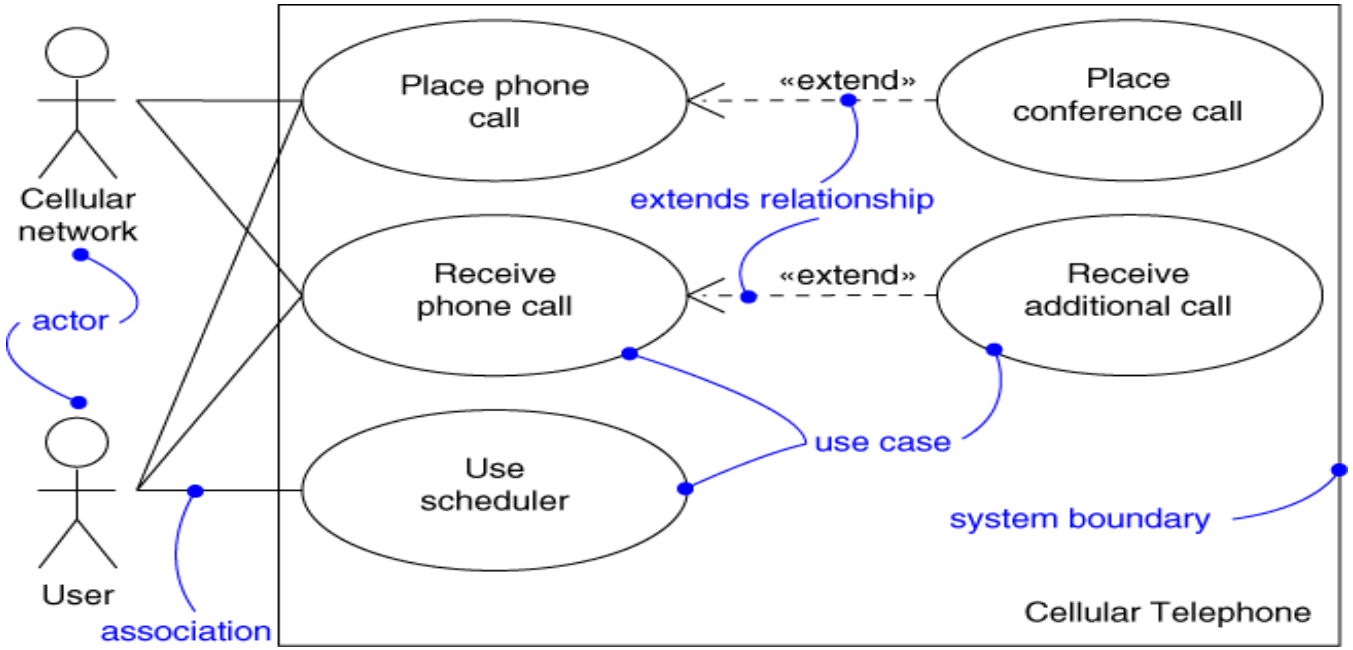


Statechart Diagram/State Machine Diagram



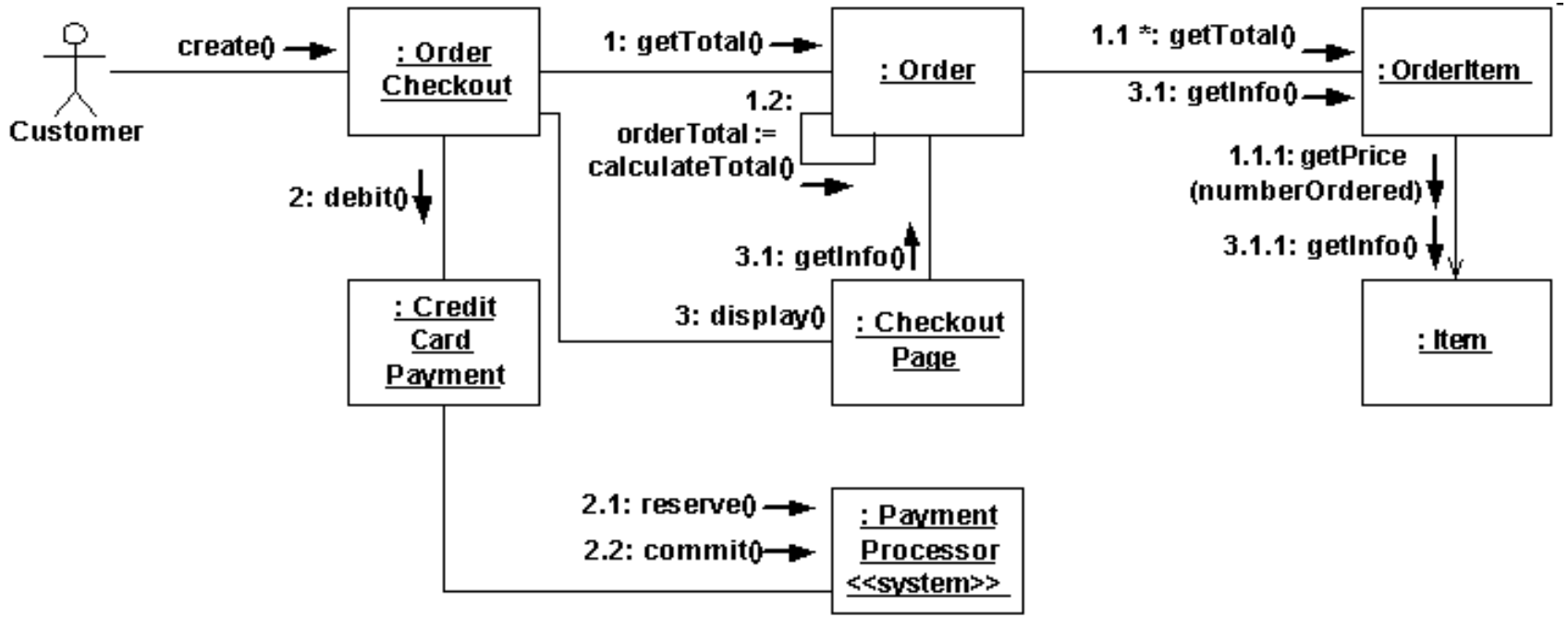
- Describes the dynamic behavior in an event-oriented manner
- Models object life-cycles
- Models reactive objects (such as user interfaces, hardware, etc.)

Use Case Diagram



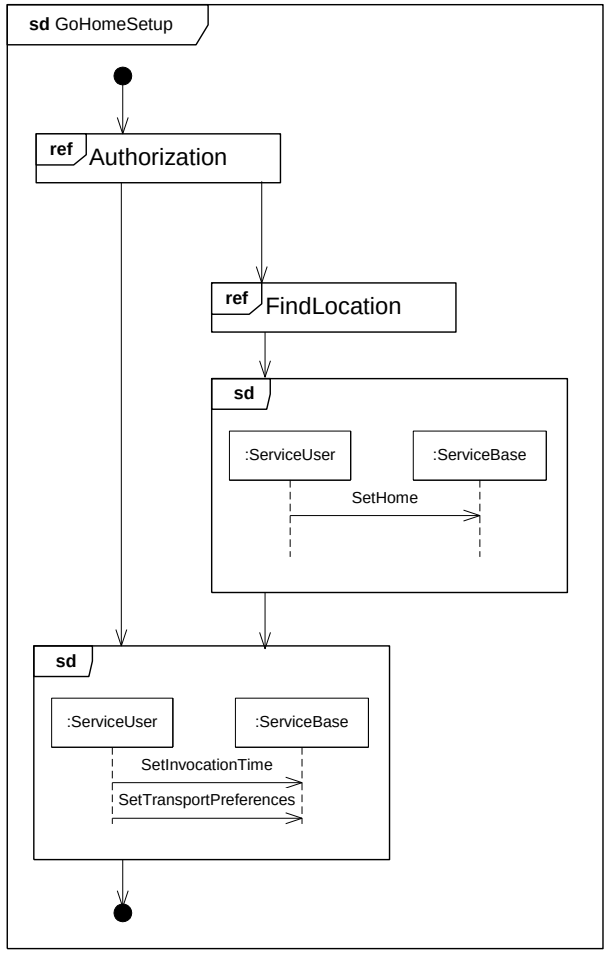
- Functionality from the user's point of view
- Made early in the lifecycle
- Purpose is to define system's boundaries, requirements and high-level architecture

Communication Diagram



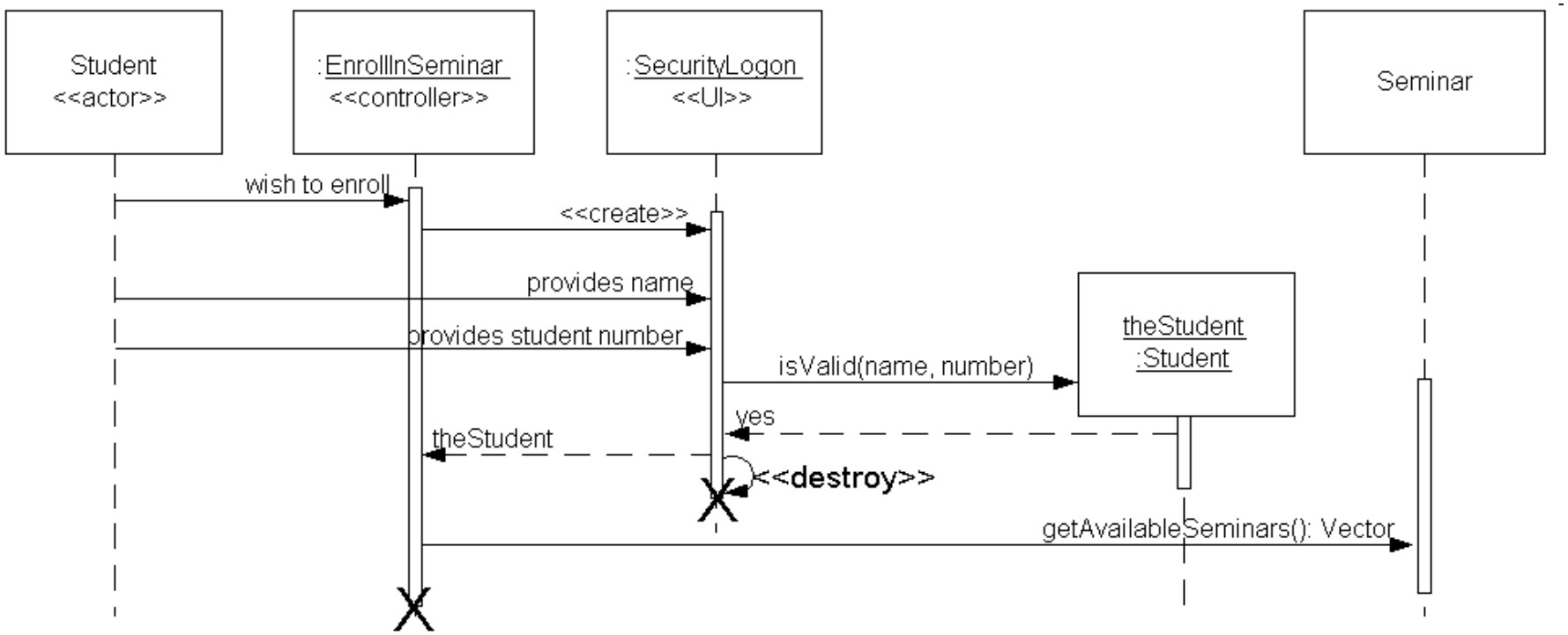
- **Collaboration diagram** in UML 1.x
- Describes dynamic behavior in a message-oriented manner
- Models the interaction between objects
- Describes a typical scenario

Interaction Overview Diagram



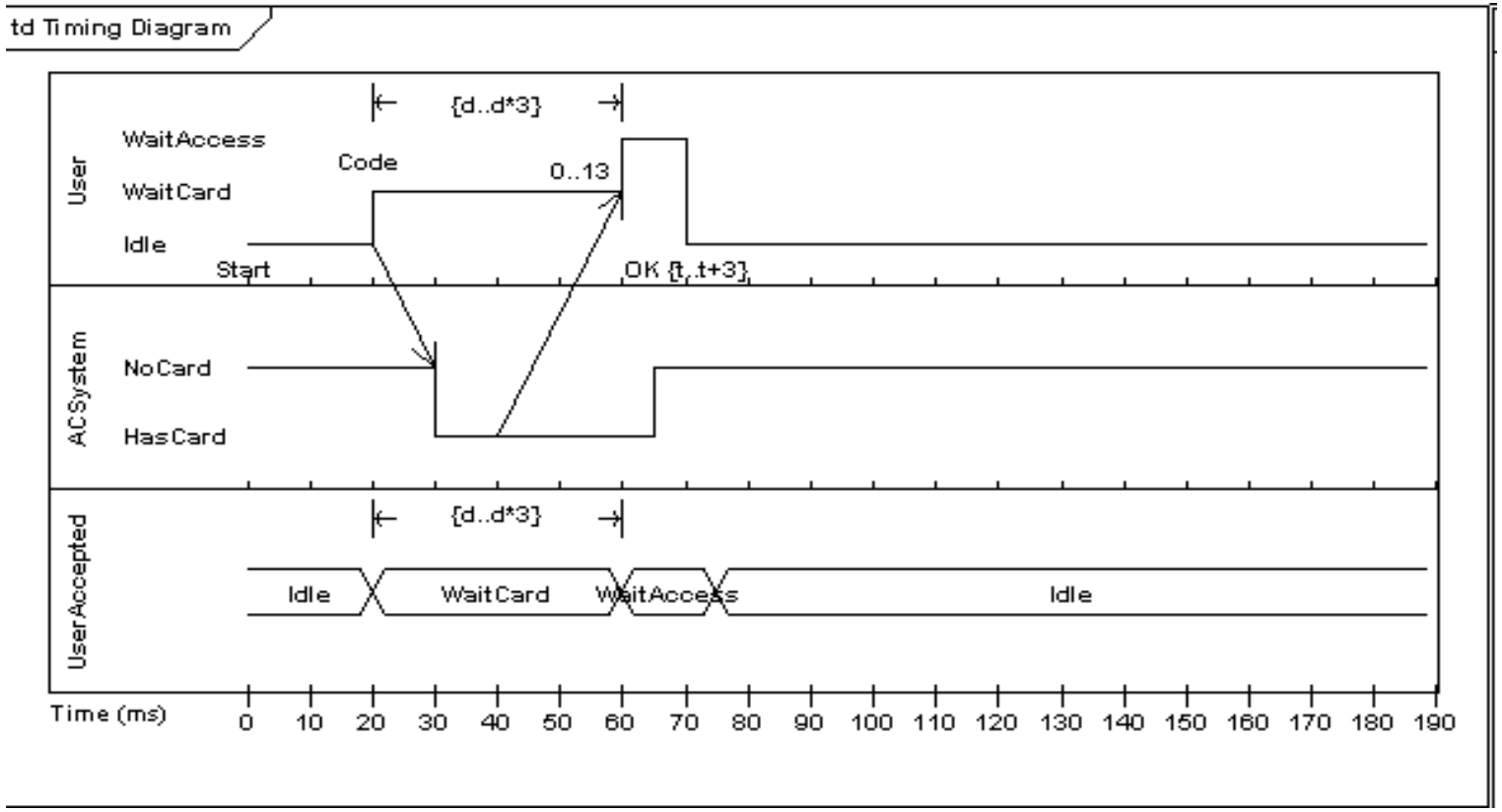
- A form of Activity Diagram with diagram references
- Added in UML 2
- Rarely used

Sequence Diagram



- Describes the dynamic behavior in a time-oriented manner
- Models program behavior in a typical scenario
- UML2 has added graphics for iteration and conditions

Timing Diagram



- A diagram for real-time design
- State changes, timing constraints, value changes

4+1 VIEW INTO ARCHITECTURE