UNIVERSITÀ DEGLI STUDI DI BERGAMO

Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione

System Design in UML

Corso di laurea Magistrale in Ingegneria Informatica

PROGETTAZIONE, ALGORITMI E COMPUTABILITÀ
(38090-MOD1)

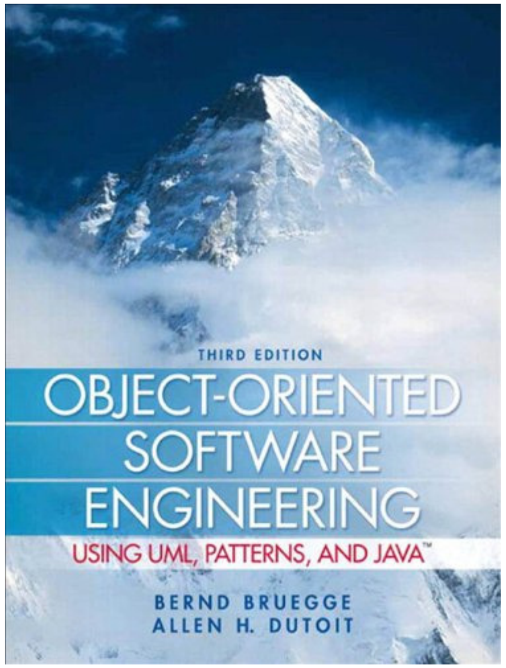
RELATORE
Prof.ssa Patrizia Scandurra

SEDE
DIGIP

DATA
04-10-2021

Main reference

Object-Oriented Software Engineering Using UML, Patterns, and Java™, Third Edition, Bernd Bruegge & Allen H. Dutoit, Chapter 5-6


THIRD EDITION
OBJECT-ORIENTED SOFTWARE ENGINEERING
USING UML, PATTERNS, AND JAVA™
BERND BRUEGGE
ALLEN H. DUTOIT

Design is difficult. Why?

Analysis: Focuses on the **application domain**

Design: Focuses on the **solution domain**

- The solution domain is changing very rapidly
    - Halftime knowledge in software engineering: About 3-5 years
    - Cost of hardware rapidly sinking
- Design knowledge is a moving target

Design window: **Time** in which **design decisions** have to be made

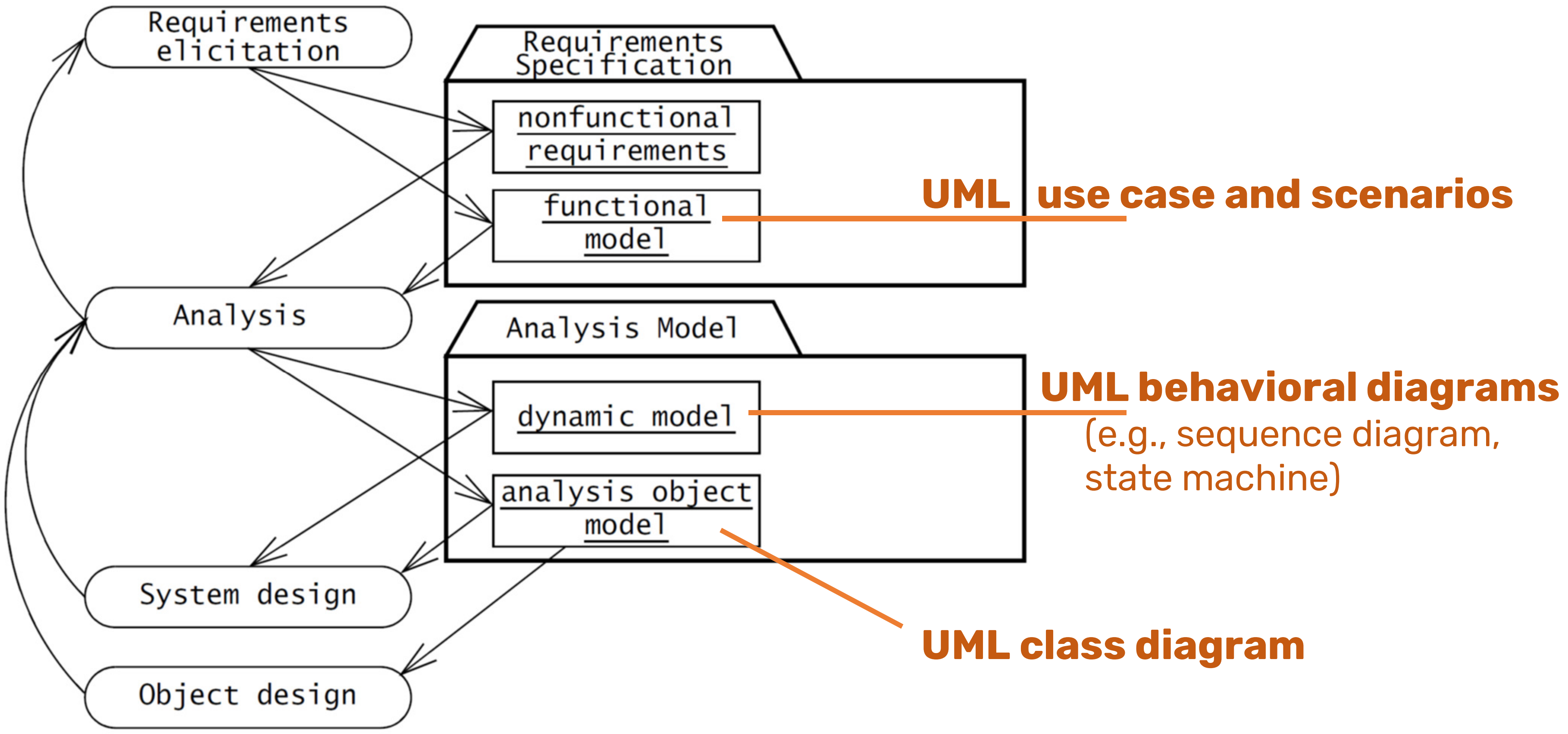
From requirements to analysis to system design to implementation

Design/implementation activities at a glance:

- Requirements and functional model
- **Analysis model**: Object model and Dynamic model
- **System Design** (*Early software architecture*)
- Design Patterns applied
- Implementation
- Static analysis and dynamic analysis (testing)

These activities must be executed as part of the iterations of the AMDD process

An overview of software engineering activities and their products



Analysis

From requirements to analysis to system design to implementation

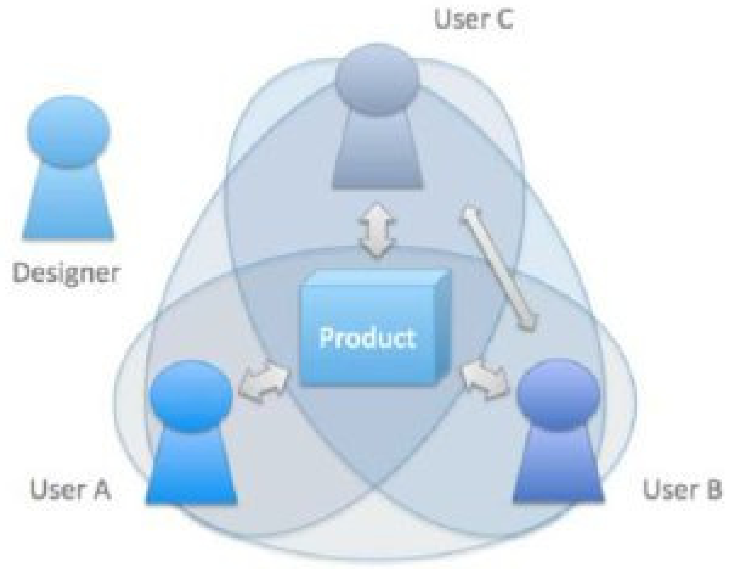
Design/implementation activities at a glance:

- Requirements and functional model
- Analysis model: Object model and Dynamic model
- System Design (*Early software architecture*)
- Design Patterns applied
- Implementation
- Static analysis and dynamic analysis (testing)

These activities must be executed as part of the iterations of the AMDD process

Analysis Concepts

The analysis model represents the system under development from **the user's point of view**



Main analysis concepts:

- **Analysis Object Models** and **Dynamic Models**
- **Entity**, **Boundary**, and **Control objects**
- **Generalization** and **Specialization**

Analysis Object Models and Dynamic Models

The **analysis object model** focuses on the individual concepts that are manipulated by the system, their properties and their relationships

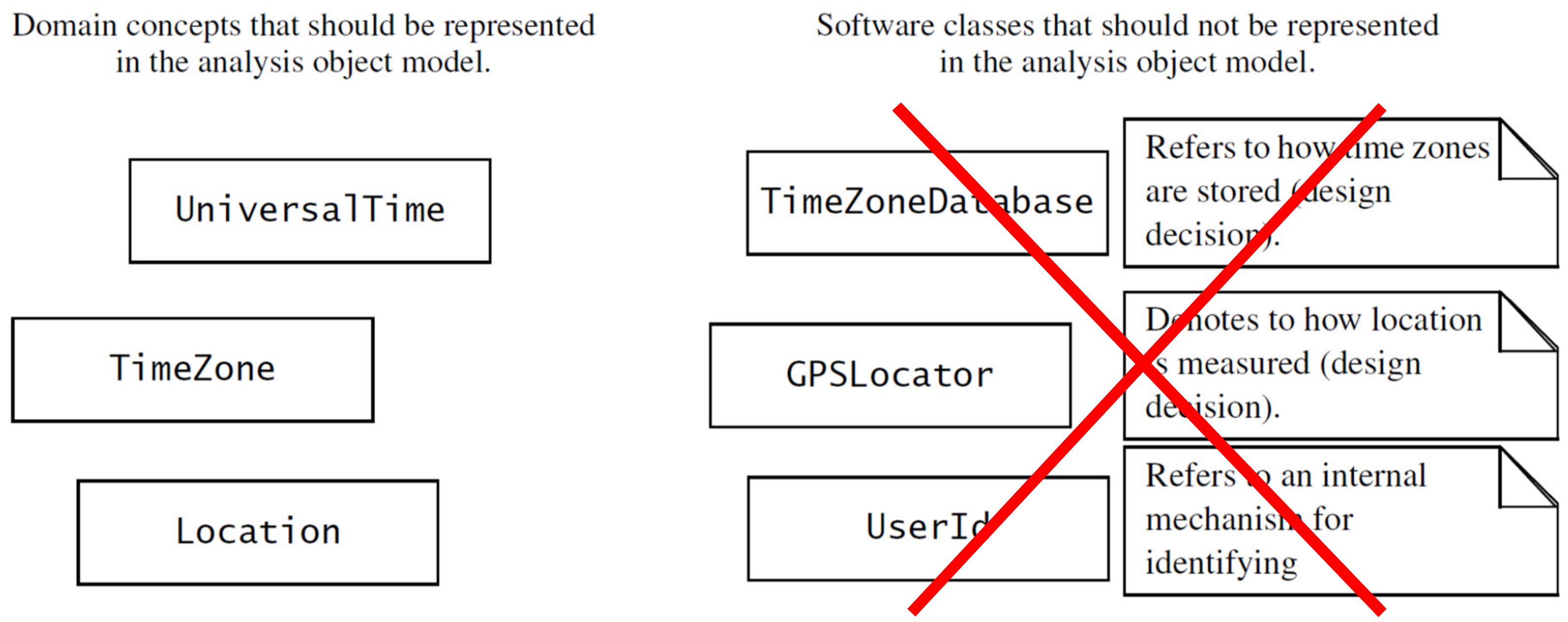
- Depicted with UML class diagrams including classes, attributes, and operations

The **dynamic model** focuses on the behavior of the system

- Depicted with UML sequence diagrams and with state machines

User-level concepts, not actual software classes or components!

Examples and counter-examples of classes in the analysis object model of a *SatWatch* system


Domain concepts that should be represented in the analysis object model.
UniversalTime
TimeZone
Location
Software classes that should not be represented in the analysis object model.
TimeZoneDatabase
Refers to how time zones are stored (design decision).
GPSLocator
Denotes to how location is measured (design decision).
UserId
Refers to an internal mechanism for identifying

It is a **visual dictionary** of the main concepts visible to the user!

There are different types of Objects

The **three object-type approach**: [Jacobson et al.,1999]

- Having three types of objects leads to models that are more *resilient* to change

Entity Objects

- Represent the **persistent information** tracked by the system (application domain objects, also called "Business objects")

Boundary Objects

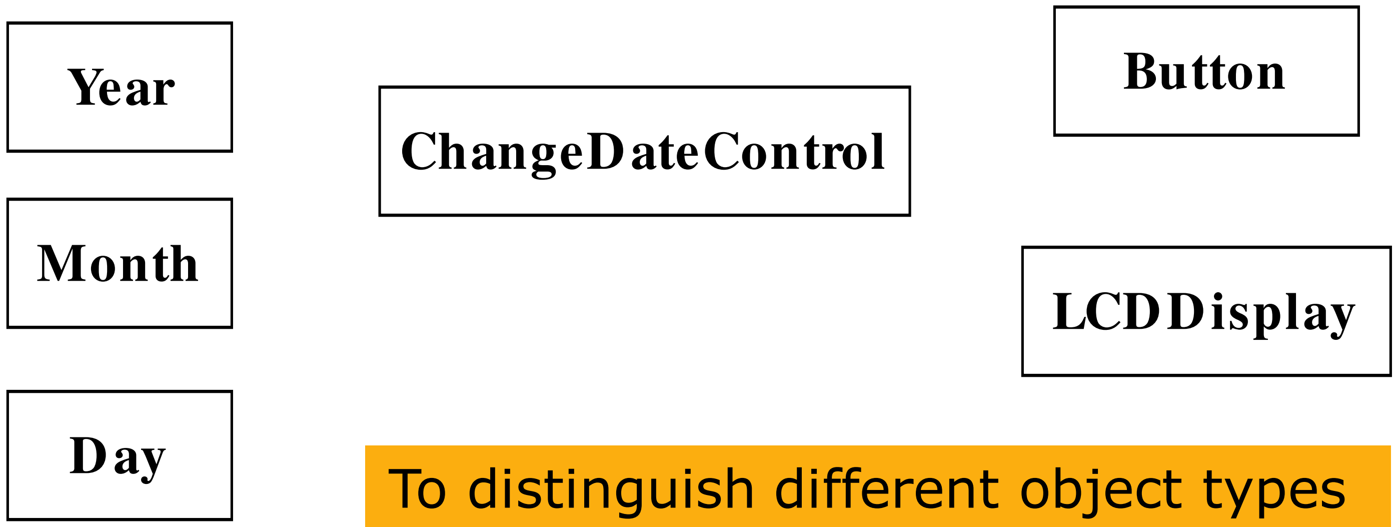
- Represent the **interaction between the user and the system**

Control Objects

- Represent the **control tasks** performed by the system

Example: 2BWatch Modeling

Always use the clients or end users terminology for naming objects



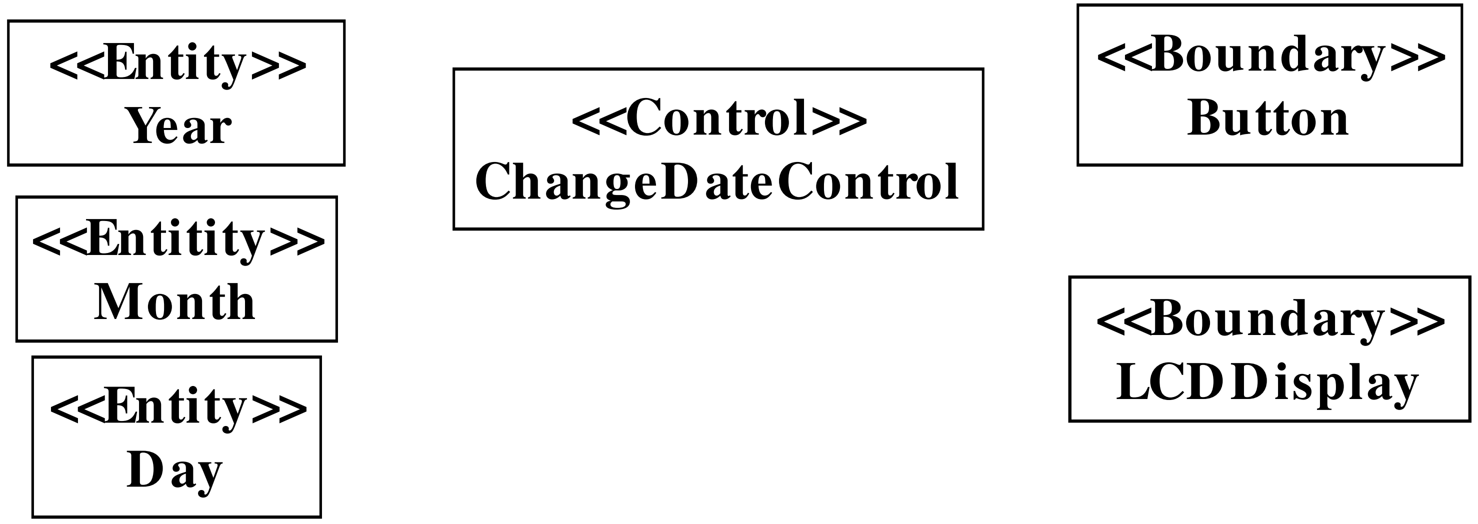
To distinguish different object types in a model we can use the **UML stereotype mechanism**

Entity Objects Control Object Boundary Objects

Naming Object Types in UML

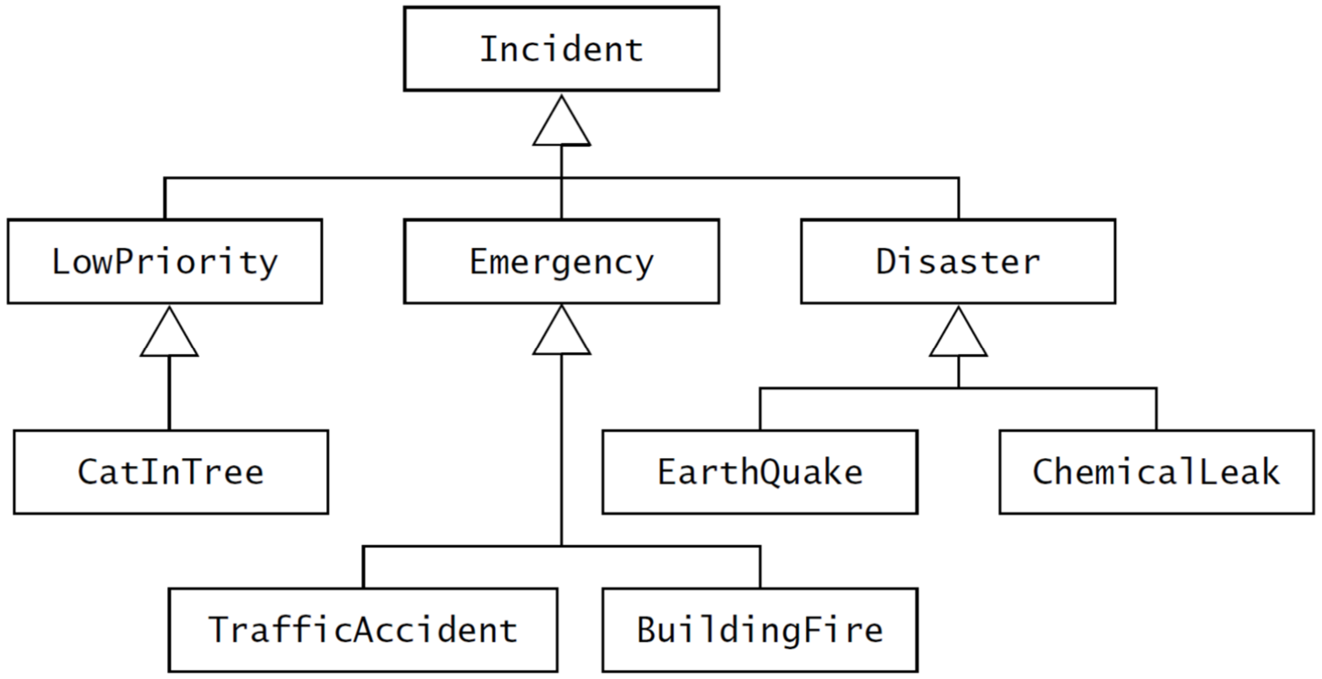
UML provides the stereotype mechanism to introduce new types of modeling elements

- Notation: **<<String>> Name**



Entity Object Control Object Boundary Object

Generalization and Specialization



Generalization and specialization result in the specification of inheritance relationships between concepts

- Generalization identifies abstract concepts from lower-level ones
- Specialization identifies more specific concepts from a high-level one

Analysis: from Use Cases to Objects

Pick a use case and **look at flow of events**

Do a **textual analysis** (noun-verb analysis)

- Nouns are candidates for objects/classes
- Verbs are candidates for operations
- This is also called Abbott’s Technique (1983)

After objects/classes are found, identify their types

- Identify real world entities that the system needs to keep track of (FieldOfficer: Entity Object)
- Identify real world procedures that the system needs to keep track of (EmergencyPlanController: Control Object)
- Identify interface artifacts (PoliceStation: Boundary Object)

(A foo) example for using the Technique

Flow of Events:

The customer enters the store to buy a toy.

It has to be a toy that his daughter likes and it must cost less than 50 Euro.

He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

An assistant helps him.

The suitability of the game depends on the age of the child.

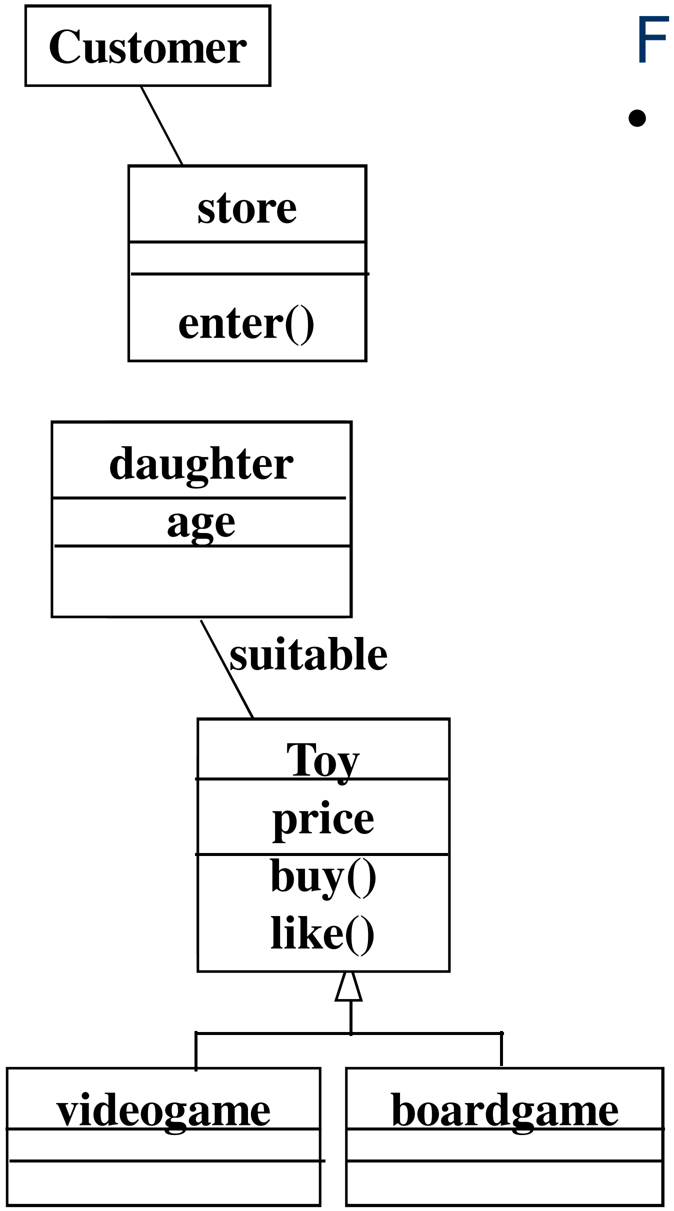
His daughter is only 3 years old.

The assistant recommends another type of toy, namely the boardgame "Monopoly".

Mapping parts of speech to model components (Abbot's Technique)

Example	*Part of speech*	*UML model component*
"Monopoly"	Proper noun	object
Toy	Improper noun	class
Buy, recommend	Doing verb	operation
is-a	being verb	inheritance
has an	having verb	aggregation
must be	modal verb	constraint
dangerous	adjective	attribute
enter	transitive verb	operation
depends on	intransitive verb	constraint, class, association

Generating a Class Diagram from Flow of Events



Flow of events:

- The customer enters the store to buy a toy. It has to be a toy that his daughter likes and it must cost less than 50 Euro. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

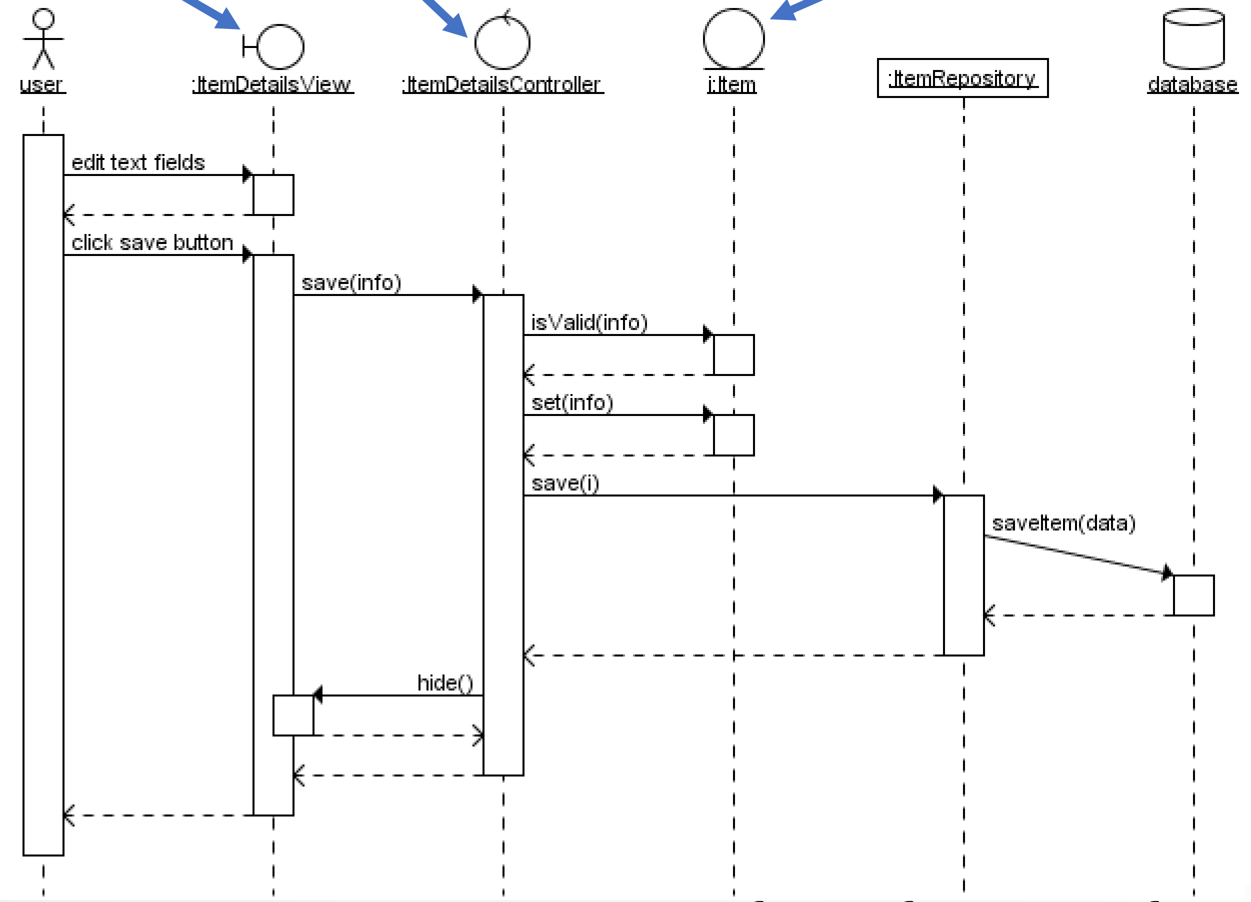
  An assistant helps him. The suitability of the game depends on the age of the child. His daughter is only 3 years old. The assistant recommends another type of toy, namely a boardgame. The customer buy the game and leaves the store

Identifying Entity-Boundary-Control objects from flow of events in a use case

Boundary objects represent the system interface with the actors

Controller objects are responsible for coordinating boundary and entity objects

Entity objects: real-world entities that the system needs to track



Modeling the flow of events of a use case with a sequence diagram can help!

Ways to find Objects

Syntactical investigation with Abbot‘s technique:

- Flow of events in use cases
- Problem statement

Use other knowledge sources:

- Application knowledge: End users and experts know the abstractions of the application domain
- Solution knowledge: Abstractions in the solution domain
- General world knowledge: Your generic knowledge and intuition

Order of Analysis Activities for Object Identification

1. Formulate a few scenarios with help from an end user or application domain expert
2. Extract the use cases from the scenarios, with the help of an application domain expert

3. Then proceed in parallel with the following:
   - Analyze the flow of events in each use case using Abbot's textual analysis technique
   - Generate the UML class diagram

Steps in Generating Class Diagrams

1. Class identification (textual analysis, domain expert)
2. Identification of attributes and operations (sometimes before the classes are found!)
3. Identification of associations between classes
4. Identification of multiplicities
5. Identification of roles
6. Identification of inheritance

Analysis Dynamic Models

Dynamic models enable us

- to represent more precisely complex behaviors involving many actors
- to identify missing use cases

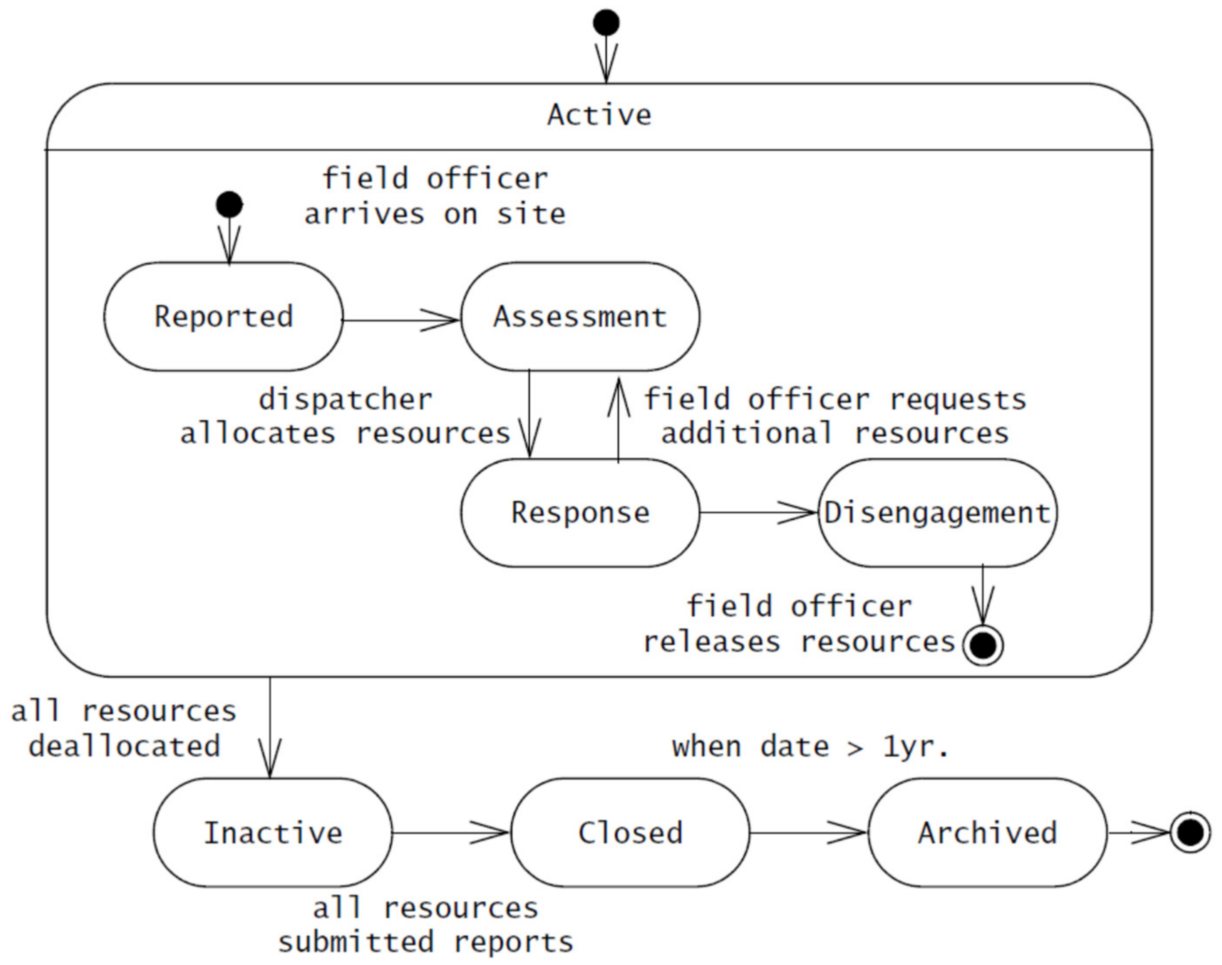
Sequence diagrams represent the behavior of the system from the perspective of a single use case

- are used to distribute behavior across objects and to identify operations

State machine diagrams represent behavior from the perspective of a single object

- Control objects with state-dependent behavior are worth considering

Example: UML state machine for Incident


Active
field officer arrives on site
Reported
Assessment
dispatcher allocates resources
field officer requests additional resources
Response
Disengagement
field officer releases resources
all resources deallocated
when date > 1yr.
Inactive
Closed
Archived
all resources submitted reports

Analysis Model vs. Design model

The **analysis model** is constructed during the analysis phase

- Main stake holders: end user, customer, analyst
- The class diagrams contains only **application domain classes**
- No implementation *specification*

The **design model** (also called **specification model**) is created during the system design phase

- Main stake holders: class specifiers, class implementors, class users and class extenders
- The class diagrams contain application domain as well as **solution domain classes**

Developers have different Views on Class Diagrams

According to the development activity, a developer plays different roles:

- Analyst
- System Designer
- Object Designer
- Implementor

Each of these roles has a different view about the class diagram (the object model)

The View of the Analyst

The analyst is interested

- in **application domain classes**: The associations between classes are relationships between abstractions in the application domain
- operations and attributes of the application classes

The analyst uses inheritance in the model to reflect the taxonomies in the application domain

- Taxonomy: An is-a-hierarchy of abstractions in an application domain

The analyst is **not interested in**

- **the exact signature of operations**
- **solution domain classes**

The View of the Designer

The designer focuses on the solution of the problem, that is, the **solution domain**

The associations between classes are references (pointers) between classes in the application or solution domain

An important design task is the **specification of interfaces**:

- The interface **of classes** and the interface **of subsystems**

Subsystems originate from *modules* (term often used during analysis):

- Module: a collection of classes
- Subsystem: a collection of *components*/classes with an interface

Goals of the Designer

The most important design goals for the designer are design usability and design reusability

Design usability: the interfaces are usable from as many classes as possible within in the system

Design reusability: The interfaces are designed in a way, that they can also be reused by other (future) software systems

=> Class libraries

=> Frameworks

=> Design patterns

The View of the Implementor

Component/class implementor

- Must realize the interface of a component/class in a programming language
- Interested in appropriate data structures (for the attributes) and algorithms (for the operations)

Component/class extender

- Interested in how to extend a component/class to solve a new problem or to adapt to a change in the application domain

Component/class user

- Interested in the signatures of the component/class operations and conditions, under which they can be invoked
- Not interested in the implementation of the component/class

System design

From requirements to analysis to system design to implementation

Design/implementation activities at a glance:

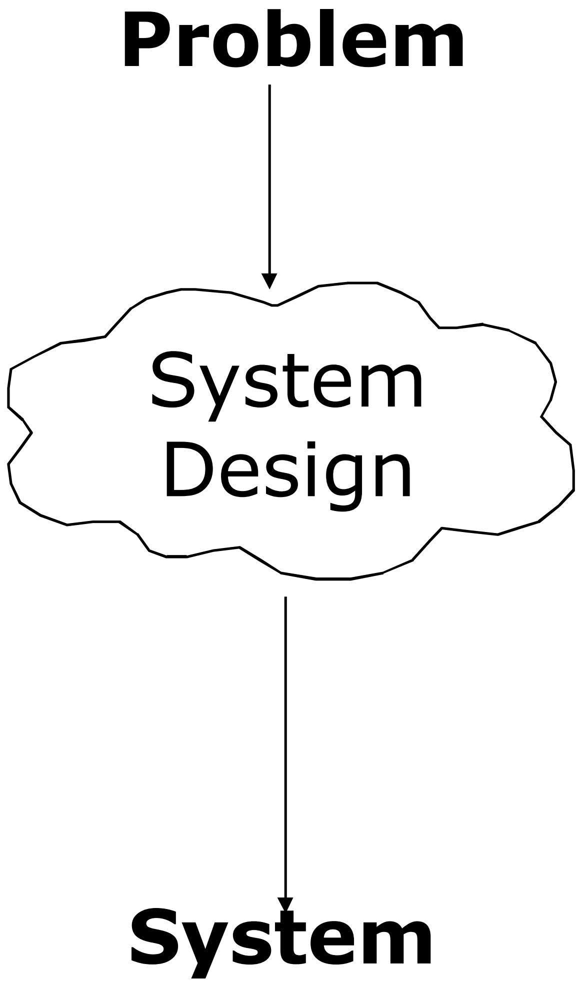
- Requirements and functional model
- Analysis model: Object model and Dynamic model
- System Design (*Early Software Architecture*)
- Design Patterns applied
- Implementation
- Static analysis and dynamic analysis (testing)

These activities must be executed as part of the iterations of the AMDD process

The Scope of System Design

Bridge the gap

- How?
- Use **Divide & Conquer**:
  1) Identify design goals
  2) Model the new system design as a set of subsystems
  3) Address the major design goals
     - Concurrency, HW/SW allocation, data management, resource access, design alternatives, QoSs, etc.



New UML Diagram Types

Subsystems are modeled in UML with a ***package***

- In particular, UML component diagrams with *ball-and-socket notation* are used during system and object design

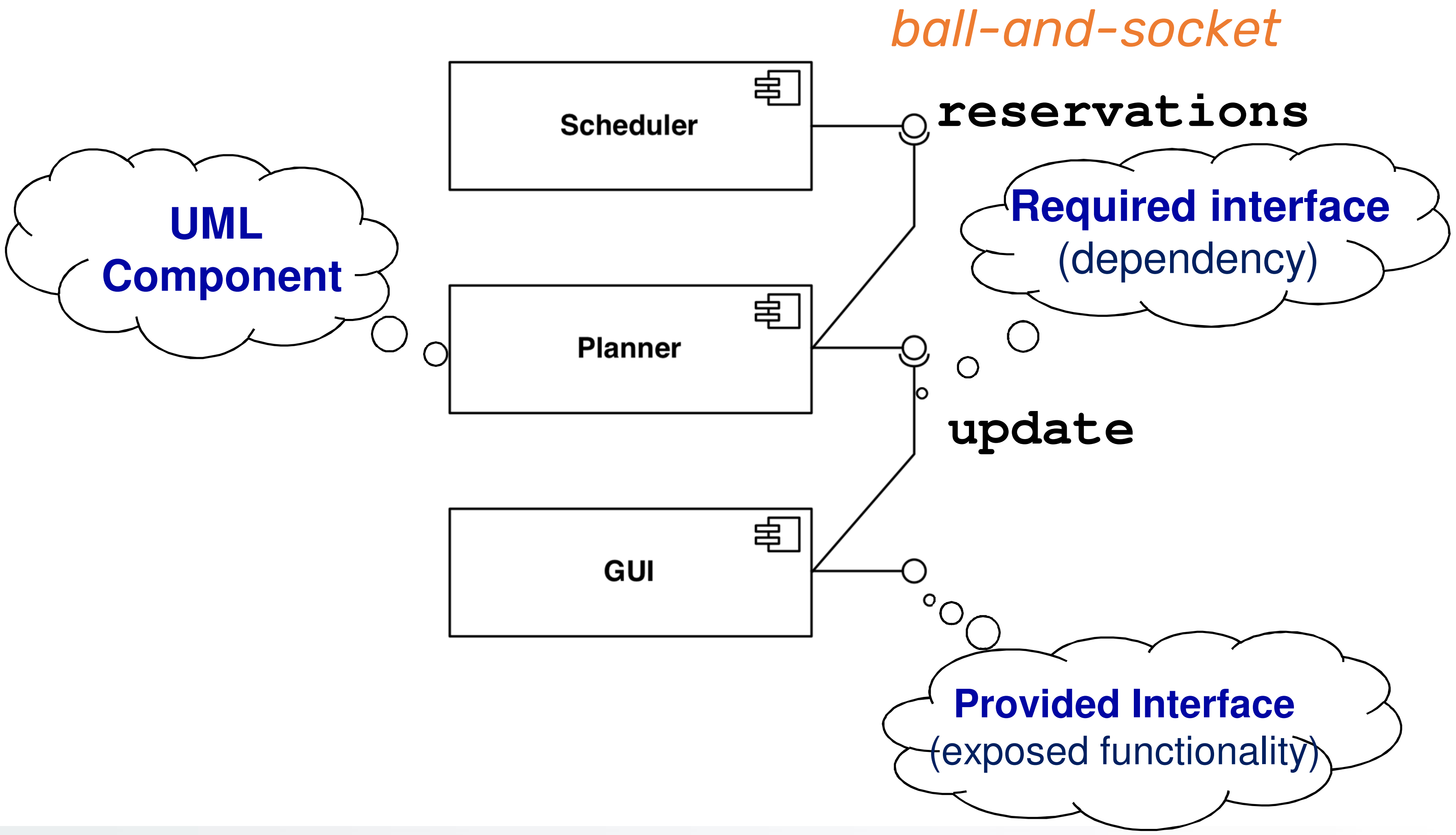
Deployment Diagram:

- Illustrates the distribution of components at run-time
- Deployment diagrams use computational nodes and connections to depict the physical resources in the system

Component Diagram:

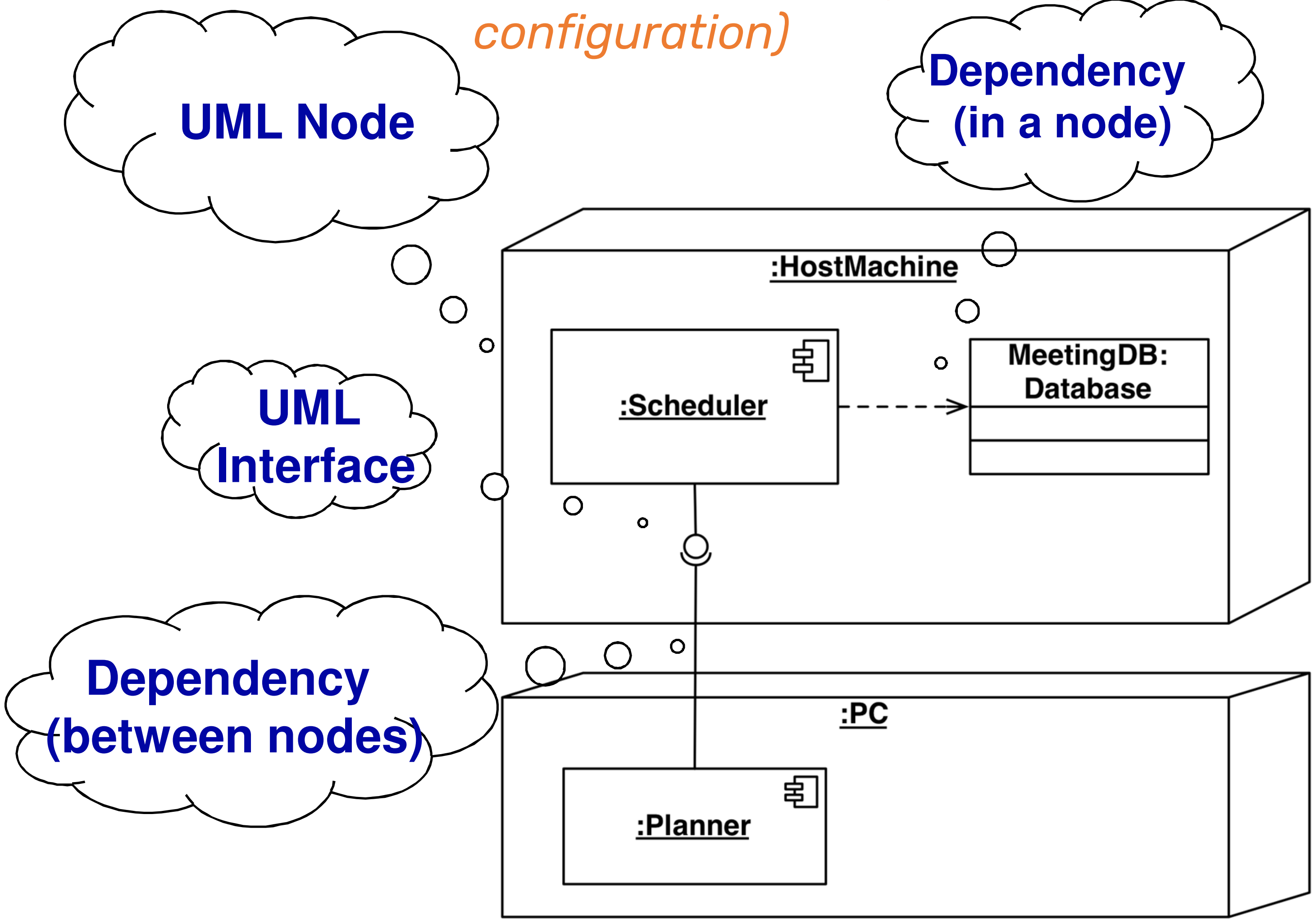
- Illustrates dependencies between components at design time, compilation time and runtime

Component Diagram Example


ball-and-socket
reservations
Scheduler
UML Component
Required interface (dependency)
Planner
update
GUI
Provided Interface (exposed functionality)

Deployment Diagram Example

Instance-level (a specific system configuration)



Architectural Style vs Architecture

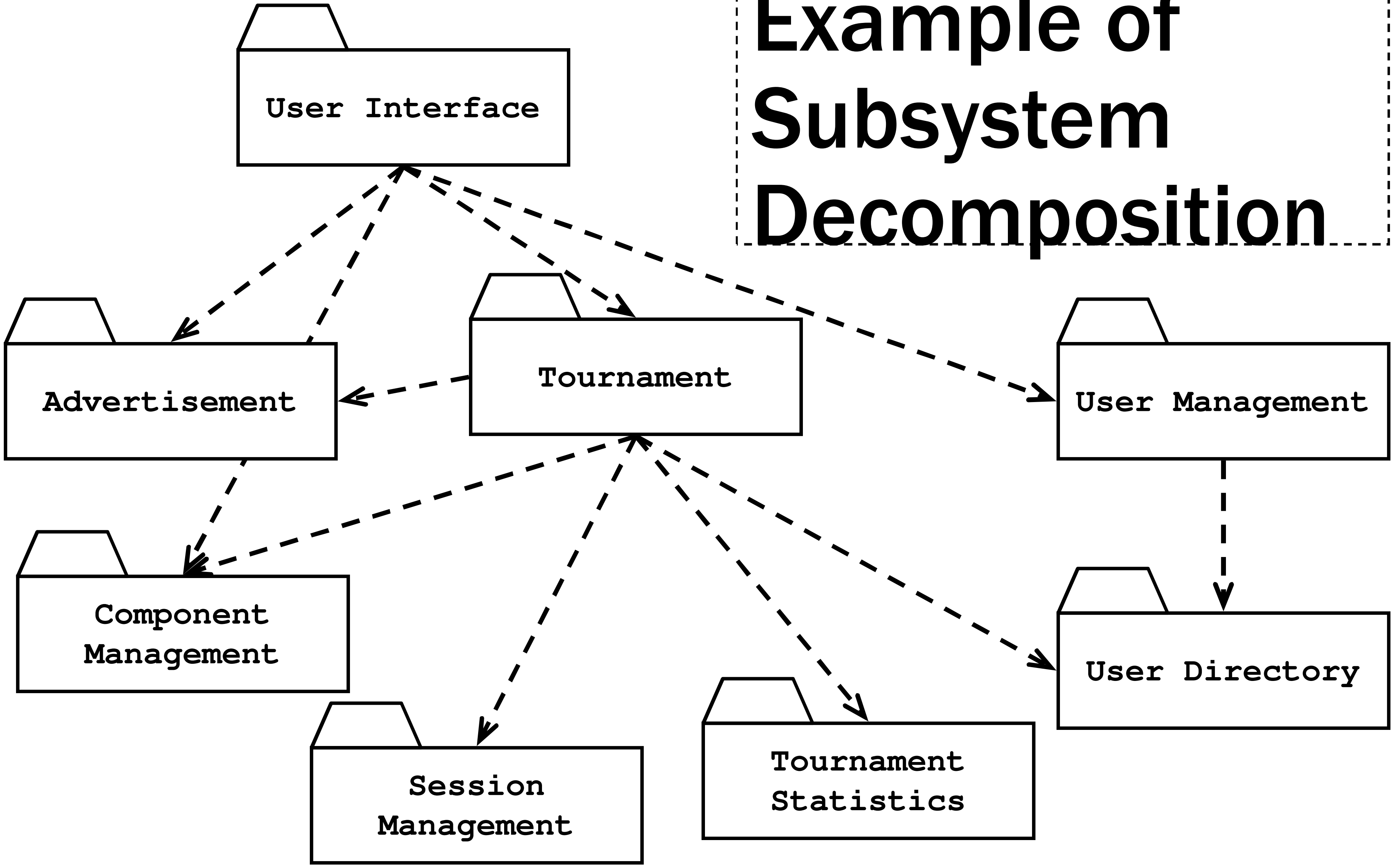
Subsystem decomposition: Identification of subsystems, services, and their association to each other (hierarchical, peer-to-peer, etc)

Architectural Style: A pattern for a subsystem decomposition

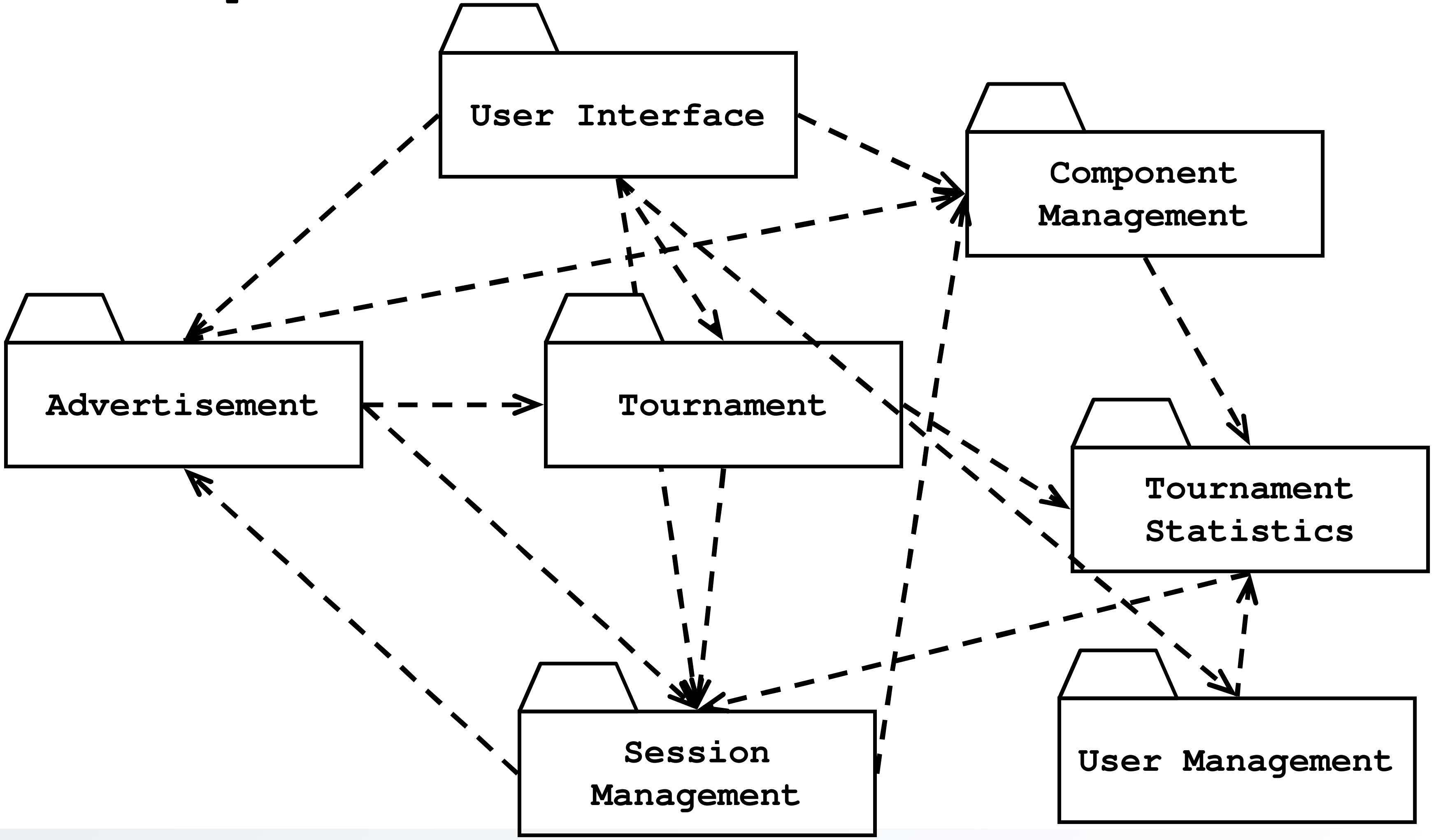
- Client/Server
- Peer-To-Peer
- Repository
- Model/View/Controller
- Three-tier, Four-tier Architecture
- Service-Oriented Architecture (SOA)
- Pipes and Filters
- etc.

Software Architecture: Instance of an architectural style

Example of Subsystem Decomposition



Example of a Bad Subsystem Decomposition

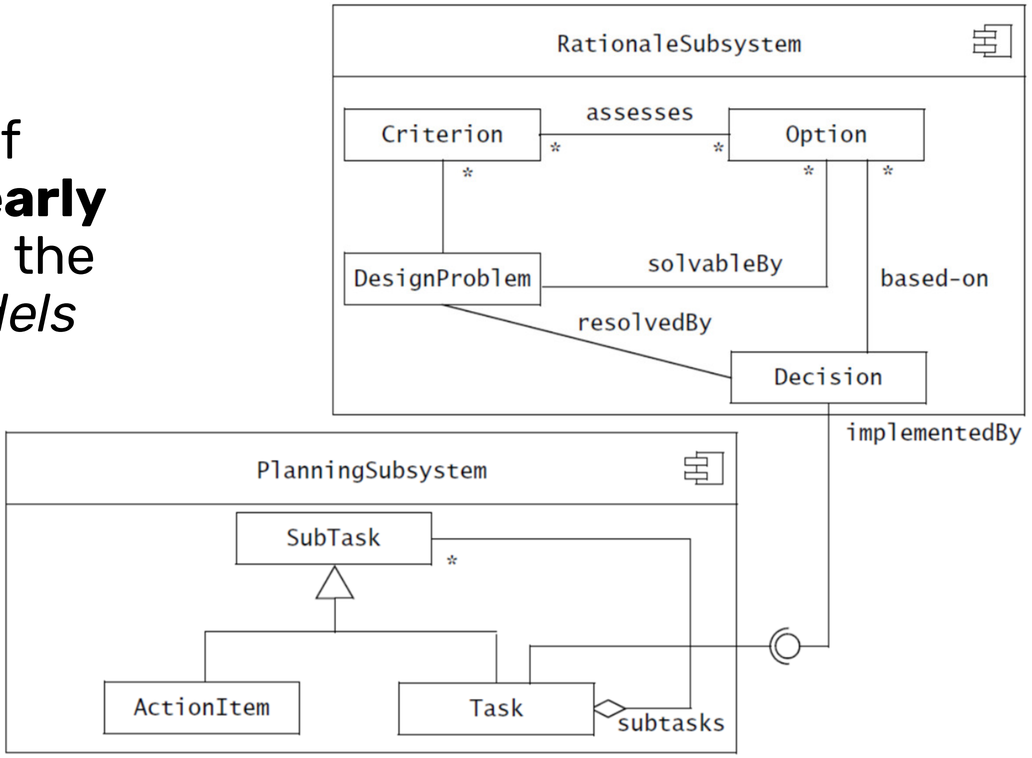


From use cases and analysis to system design: best practice

- **Good design principle**: The System as set of *Interface Objects*
  - publish the service (= Set of public operations) provided by the subsystem

- **Design heuristics**: a set of guidelines for identifying **early design components** from the *use case* and *analysis models*

Best practice for *agile modeling*


RationaleSubsystem
Criterion
assesses
Option
*
*
*
*
*
DesignProblem
solvableBy
based-on
resolvedBy
Decision
implementedBy
PlanningSubsystem
SubTask
*
ActionItem
Task
subtasks