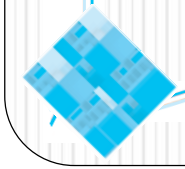
Architectural Design

King Saud University
College of Computer and Information Sciences
Department of Computer Science

Dr. S. HAMMAMI



Objectives

- To Establish the overall structure of a software system
- To introduce architectural design and to discuss its importance
- To describe types of architectural model that may be used

Architectural Design

What is Software architecture?

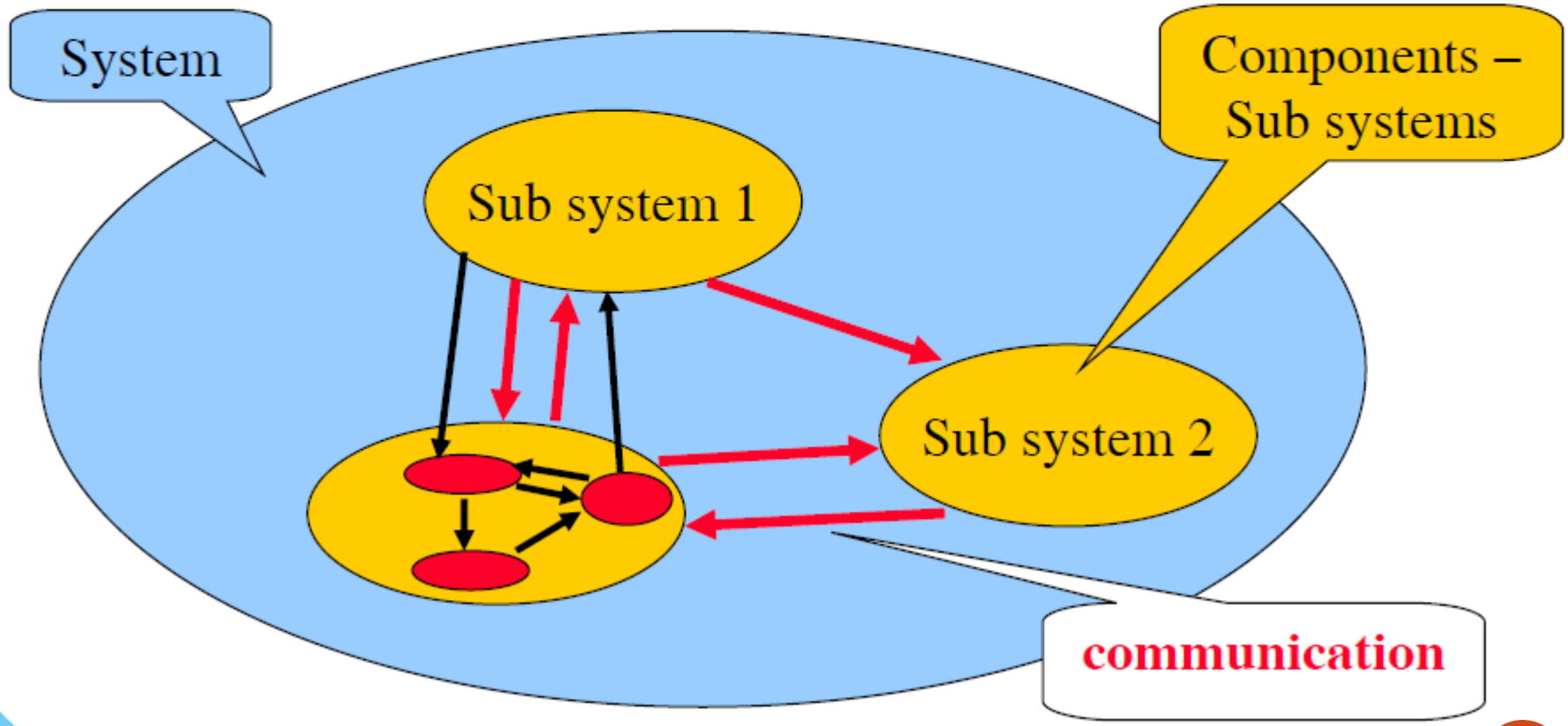
Architectural design is the design process for:

- **identifying the sub-systems** making up a system, and
- the framework for **sub-system control and communication**

- The output of this design process is a description of the *software architecture*

Architectural design

- **Identify system components** and their **communications**



Architectural design process

- System structuring
  - The system is **decomposed into several principal sub-systems and communications** between these sub-systems are identified
- Control modelling
  - A **model of the control relationships** between the different parts of the system is established
- Modular decomposition
  - The identified **sub-systems are decomposed into modules**

Architectural design process: System structuring

- Concerned with decomposing the system into **interacting sub-systems**
- The architectural design is normally expressed as a **block diagram** presenting **an overview of the system structure**
- More specific models showing how sub-systems share data, are distributed and interface with each other may also be developed

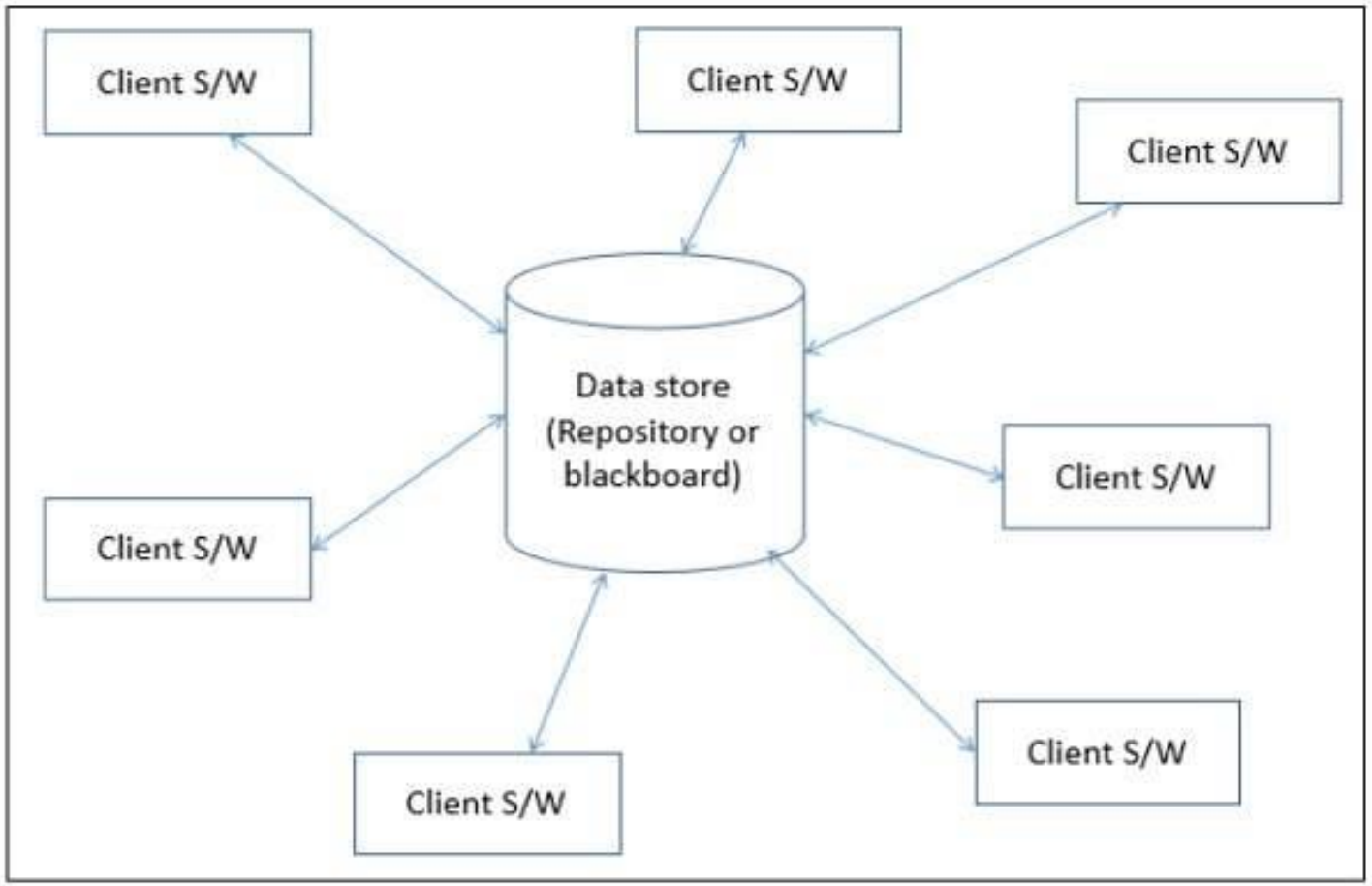
Architectural Design: System Organisation

- Reflects the basic strategy that is used to structure the system
- Three architectural styles are widely used:
  - Shared data repository
  - Client-server (services and servers)
  - Abstract machine or layered style

The repository model

- Sub-systems must exchange data. This may be done in two ways:
  - Shared data is held in a central database or repository and may be accessed by all sub-systems
  - Each sub-system maintains its own database and passes data explicitly to other sub-systems
- When large amounts of data are to be shared, the repository model of sharing is most commonly used

Example: repository model for

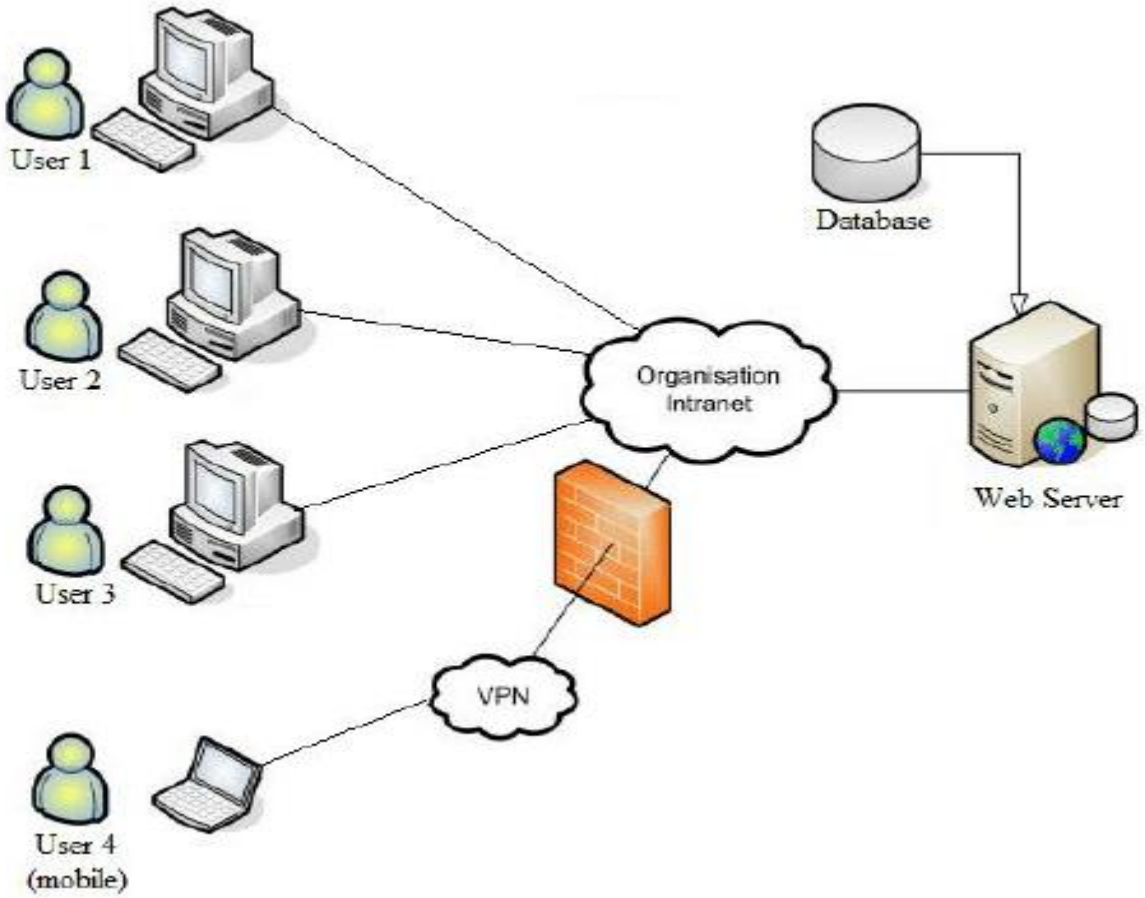


Repository Model Characteristics

- **Advantages**
  - Can efficiently share large amounts of data
  - Sub-systems need not be concerned with how data is produced by other sub-systems
  - Centralized backup, access control, and error recovery
  - New tools compatible with the repository schema (data model) are easily integrated
- **Disadvantages**
  - Sub-systems must agree on a repository data model, compromising on needs of each tool, affecting performance and integration with incompatible tools
  - Translating data into different data model is difficult, expensive, or impossible;
  - Same policy forced on all sub-systems
  - Difficult to distribute repository over many machines efficiently, leading to problems with data redundancy and inconsistency

Client-server architecture

- **Distributed system model** which shows how data and processing is distributed across a range of components
- Set of stand-alone **servers which provide specific services** such as printing, data management, etc.
- Set of **clients which call** on these services
- **Network** which allows clients to access servers


User 1
User 2
User 3
User 4
(mobile)
Database
Organisation
Intranet
Web Server
VPN

Client-server characteristics

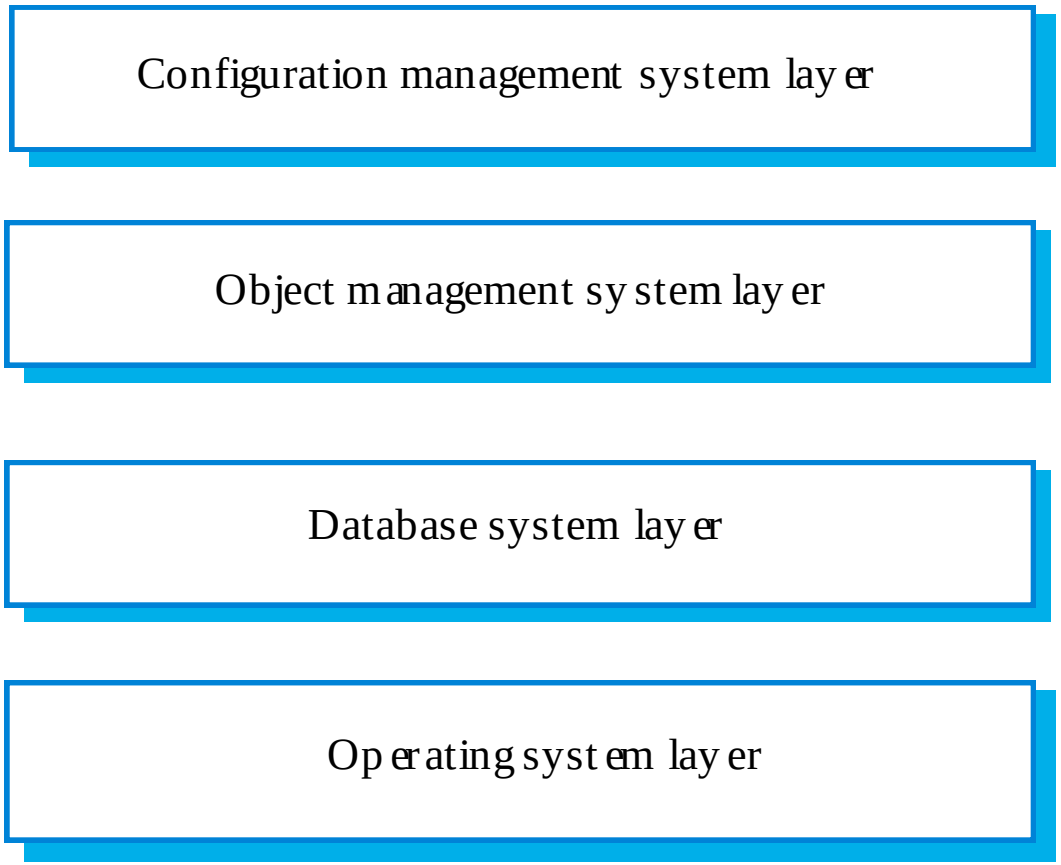
- Advantages
  - Distribution of data is straightforward
  - Makes effective use of networked systems. May require cheaper hardware
  - Easy to add new servers or upgrade existing servers
- Disadvantages
  - No shared data model so sub-systems use different data organisation. data interchange may be inefficient
  - Redundant management in each server
  - No central register of names and services - it may be hard to find out what servers and services are available

Abstract machine model **(Layered Model)**

- Used to model the interfacing of sub-systems
- Organises the system into a set of layers (or abstract machines) each of which provide a set of services
- Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected
- However, often difficult to structure systems in this way

- Difficult to structure system in layers:
  - Inner layers may provide services required in several layers, making outer layers depend on more than its adjacent layer
  - Performance may suffer when service requests must be interpreted across many layers before processing

Example: Abstract machine model for Version management system



Topics covered

- Introduction
- Architectural design decisions
- System organisation
- **Decomposition models**
- Control models

Modular decomposition

- Another structural level where **sub-systems are decomposed into modules**
- Two modular decomposition models covered
  - **An object model** where the system is decomposed into interacting objects
  - **A data-flow model** where the system is decomposed into functional modules which transform inputs to outputs. Also known as the pipeline model
- If possible, decisions about concurrency should be delayed until modules are implemented

Object models decomposition

- Structure the system into a set of **loosely coupled objects** with well-defined interfaces
- Object-oriented decomposition is concerned with identifying object classes, their attributes and operations
- When implemented, objects are created from these classes and some control model used to coordinate object operations

Architecture Concepts

Some concepts related to architecture:

1) subsystems
    a) classes
    b) services

2) design principles for defining subsystems:
    1) coupling
    2) cohesion

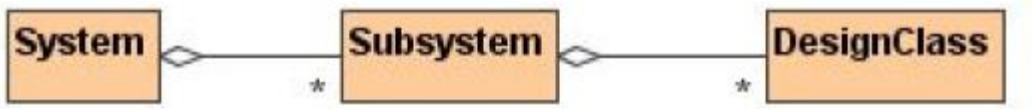
3) layering strategy for defining subsystems:
    1) responsibility driven
    2) reuse driven

Subsystems: Classes

A solution domain may be decomposed into smaller parts called subsystems.

Subsystems may be recursively decomposed into simpler subsystems.

Subsystems are composed of solution domain classes (design classes).



Coupling

Definition

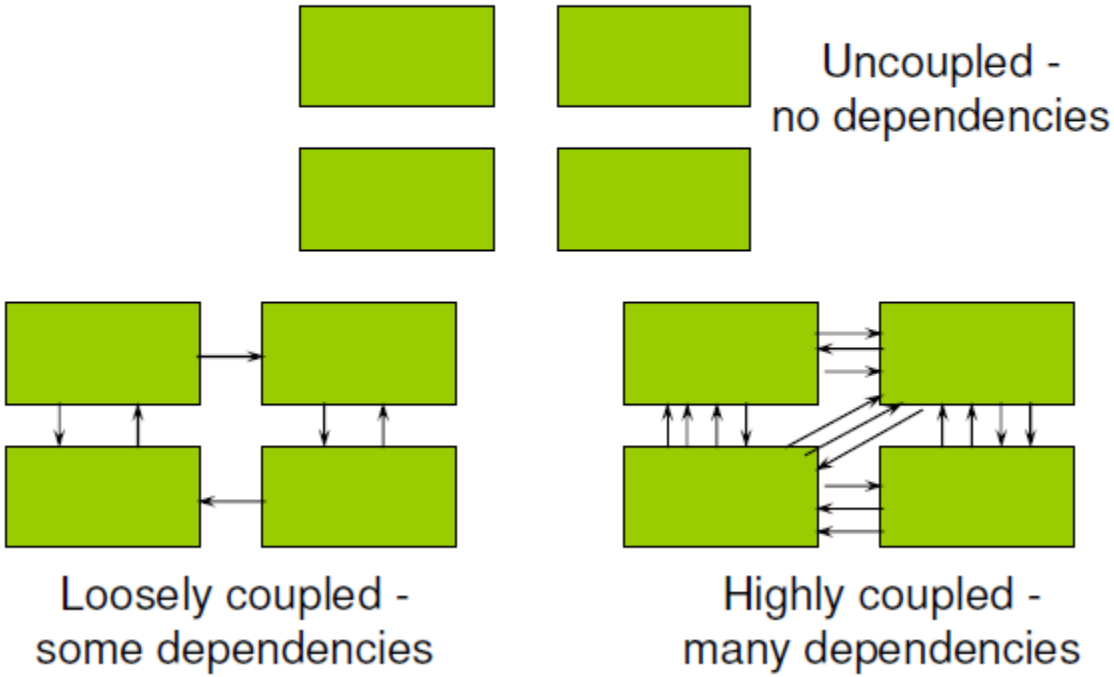
Coupling is the strength of dependencies between two sub-systems.

Loose coupling results in:

1) sub-system independence
2) better understanding of sub-systems
3) easier modification and maintenance

High coupling is generally undesirable.

Coupling Example



Cohesion

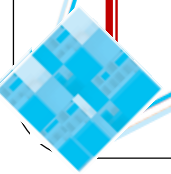
Definition

Cohesion or Coherence is the strength of dependencies within a subsystem.

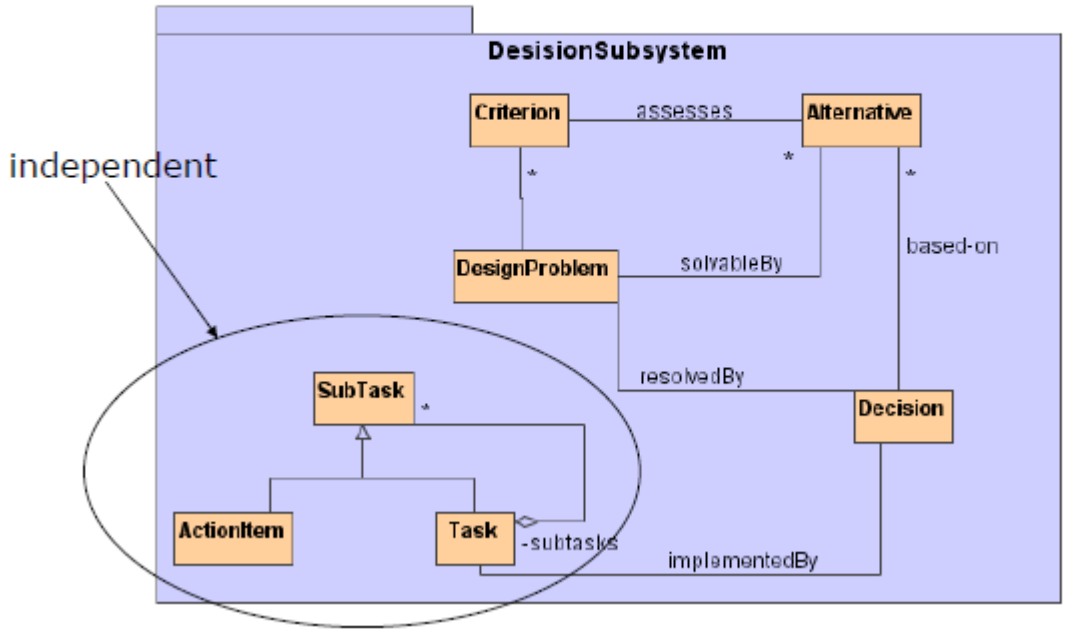
In a highly cohesive subsystem:

- subsystem contains related objects
- all elements are directed toward and essential for performing the same task.

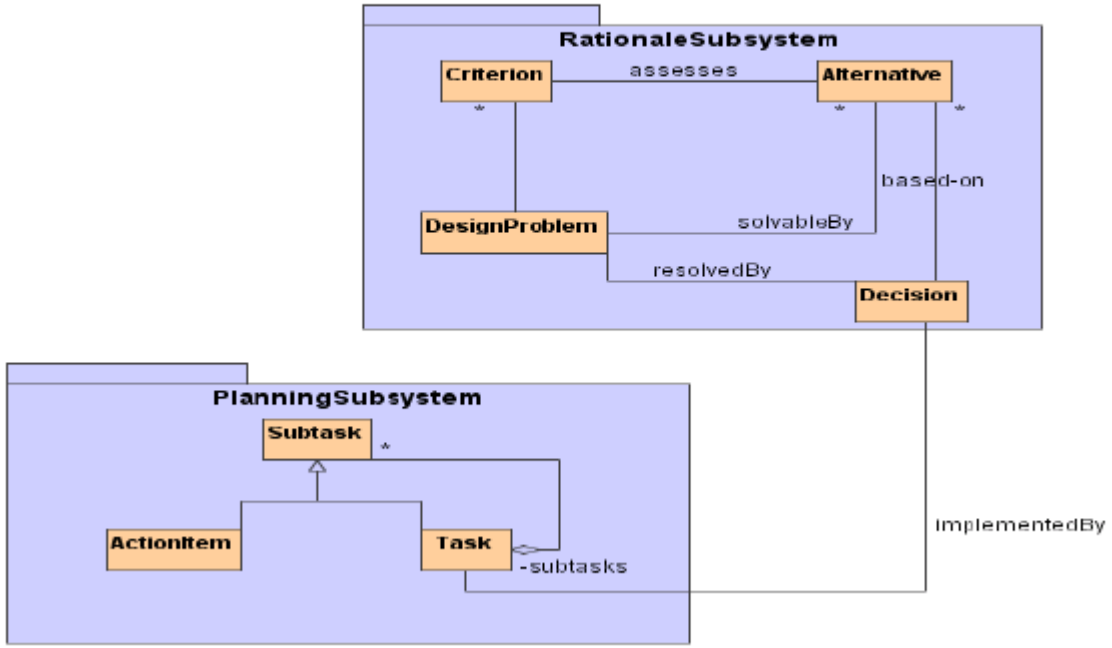
Low cohesion is generally undesirable



Low Cohesion Example



High Cohesion Example



Object models decomposition

- **Advantages:**
  - Loose coupling ensures that changes in one object class does not affect other objects
  - Since objects tend to reflect real-world entities, object models are easy to understand

- **Disadvantages:**
  - Changes to the interface of an object have an impact on other users of the object
  - Complex entities may be difficult to represent as objects