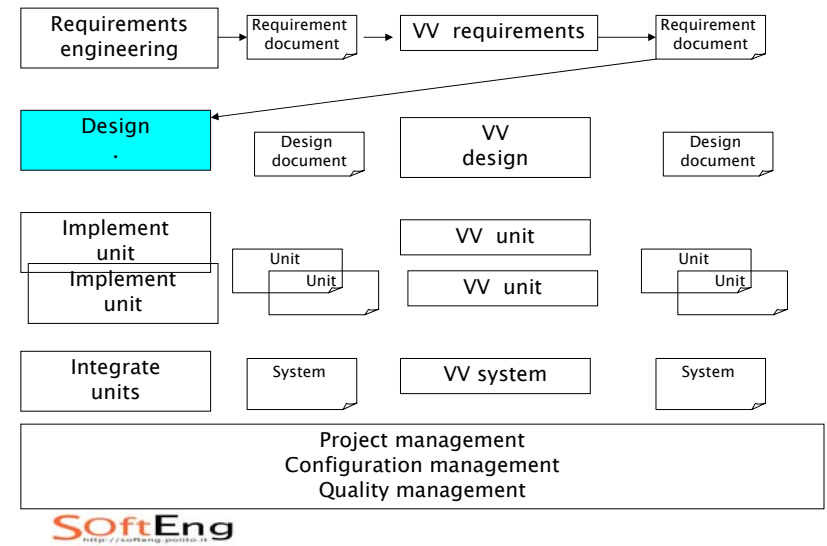


Architecture and Design



The whole picture



Architecture

- Requirements: **what** the system should do
- Architecture, design: **how** the system should be built
 - ♦ Architecture, design: same flavour but
 - Architecture: high level, decide major components and their control and communication framework
 - Design: lower level, decide internals of each component

Architecture, activities

- Analysis
- Formalization
 - ♦ Text, diagrams (UML)
- Verification

Requirements to design

- Given one set of requirements
- In general many different designs are possible (*design choices*)
 - ♦ Cfr. Requirement: mid sized car in price range 10 to 20k
 - ♦ Designs: hundreds of models on the market,
 - High level design choices
 - diesel or gas engine
 - front or rear or all wheel drive
 - Low level design choices
 - Color
 - With ABS, ESP, or not
- But not all designs are equal

Requirements to design

- A creative process
- Driven by skill and experience
- Experience formalized in semi formal guidelines
 - ♦ Architectural models (layered, etc, see later)
 - ♦ Design patterns (not in this course)

Architecture, design, why

- Most defects come from requirements and design
- Essential to define, analyze and evaluate design choices *early*
- If no design is defined, but code is developed immediately, design choices are made implicitly and evaluated *late*
- Doing design allows to make design choices *explicit*, document and evaluate them

Properties of a design

- Functional properties
 - ♦ Does the design support the requirements ?
 - As described on the requirements document
- Non functional properties
 - ♦ Reliability (Robustness)
 - What if one component/function fails?
 - ♦ Maintainability (Ease of change)
 - What if one component changes?
 - What if one requirement changes?
 - ♦ Efficiency
 - Response time for a function/requirement
 - Memory occupation

Properties

- ◆ Portability
 - What if MS Windows instead of Linux?
 - What if SQLserver instead of Oracle?
- ◆ Usability
- ◆ Safety
- ◆ Security
 - Privacy of data, access control, ..
- ◆ And also
 - Cost
 - Schedule
 - Skills available from staff

Properties – what to do

- ◆ Performance
 - Localise critical operations and minimise communications. Use large rather than fine-grain components.
- ◆ Security
 - Use a layered architecture with critical assets in the inner layers.
- ◆ Safety
 - Localise safety-critical features in a small number of sub-systems.
- ◆ Availability
 - Include redundant components and mechanisms for fault tolerance.
- ◆ Maintainability
 - Use fine-grain, replaceable components.

Properties

- ◆ Using large-grain components improves performance but reduces maintainability.
- ◆ Introducing redundant data improves availability but makes security more difficult.
- ◆ Localising safety-related features usually means more communication so degraded performance

Properties, trade offs

- Not all properties can be satisfied
- Design is also about deciding tradeoffs
 - ◆ Ex security (add layers) vs. speed (avoid layers)
 - ◆ Ex. changeability (add abstraction layer to insulate from hardware change) vs. speed (avoid layers)
- Possibly, trade offs are decided at requirement time
 - ◆ Ex: requirement: security prevails on speed

Formalization of architecture

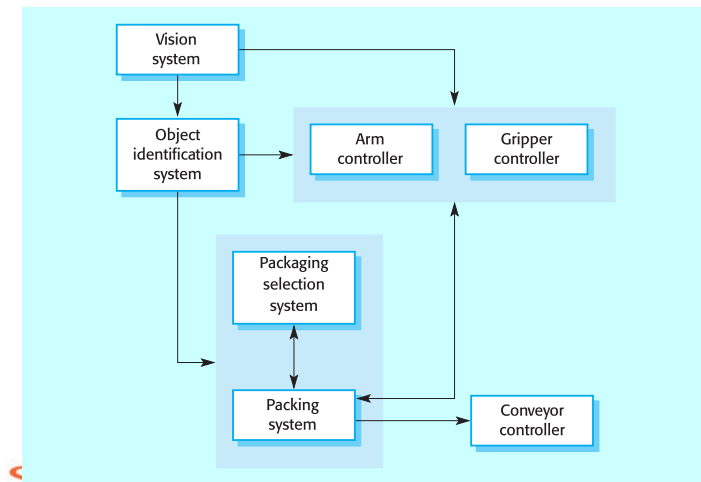
Formalizing the architecture

- Informal
 - ♦ box and lines
- Semiformal
 - ♦ UML diagrams
 - Structural views
 - Component, package diagrams
 - Class diagrams
 - Deployment diagram
 - Dynamic views
 - Sequence diagrams
 - State charts
- Formal ADL (Architecture description languages)
 - ♦ Many, ex C2 (component Connector)

Box and line diagrams

- Very abstract – they do not show the nature of component relationships nor the externally visible properties of the sub-systems.
- However, useful for communication with stakeholders and for project planning.

Packing robot control system



UML diagrams

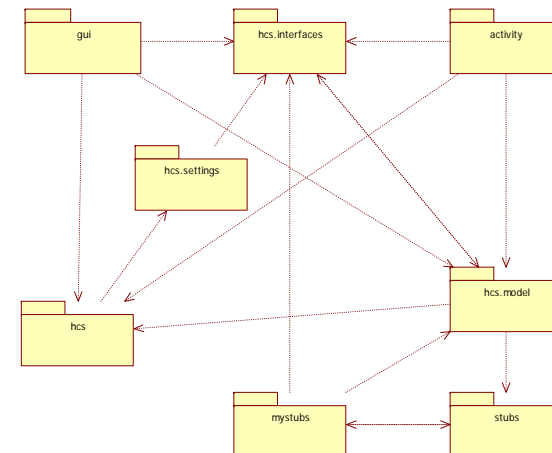
- Structural view
 - ♦ Component or package diagram for high level view
 - ♦ Class diagram (inside each package or component)
 - ♦ Class description (for each class)

Heating control system

- ♦ (see requirement document and design document)
- Choices – high level
 - ♦ One CPU, one process (no distribution, no concurrency)
 - ♦ Communication and control: procedure call
 - ♦ Layered style (at least partially)
- Choices – low level
 - ♦ Observer pattern

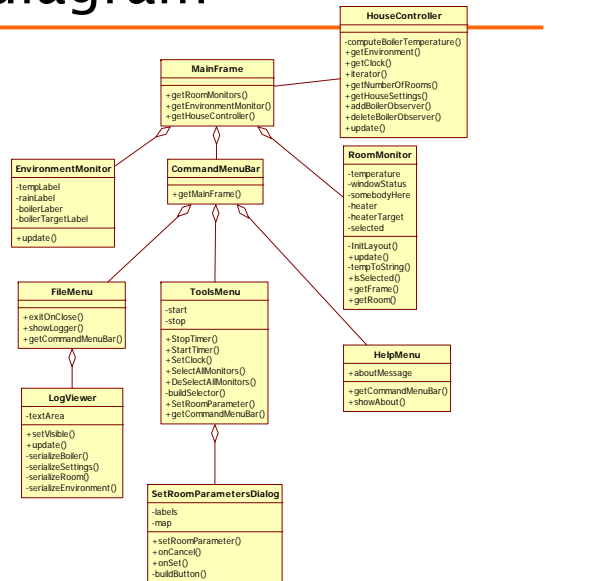
UML – structural

UML – package diagram



UML – class diagram

- Package GUI



Class (HouseController)

- The main class in the heating control system, it integrates the logical model of the various parts of the house and performs the high-level activities.
- computeBoilerTemperature()
 - Computes the desired water temperature in the boiler
- getEnvironment()
 - Navigates to the logical model of the environment
- getClock()
 - Navigates to the Clock
- iterator()
 - Returns an iterator to the contained Rooms
- getNumberofRooms()
 - Returns the number of rooms
- getHouseSettings()
 - Navigates to the current global settings
- update()
 - Computes the next logical state of the system
- addBoilerObserver()
 - Adds an observer to the Boiler
- deleteBoilerObserver()
 - Removes an object from the list of Boiler observers

Structure and hierarchy

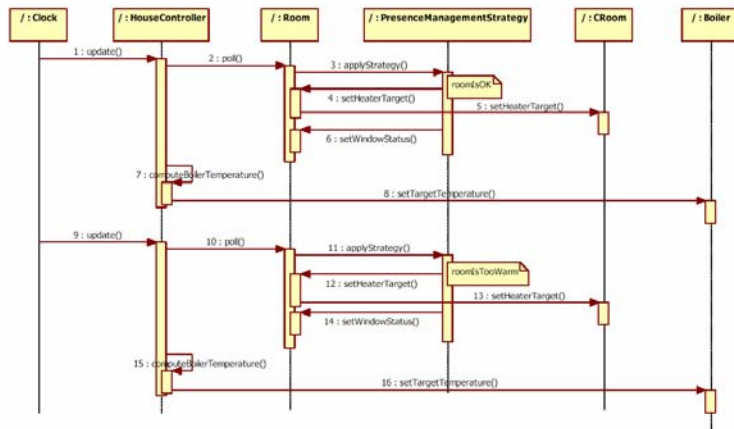
- UML helps in presenting structure in an organized (hierarchical) way
 - Packages in system
 - Classes in package
 - Attributes and methods in class
- Presentation is sequential, but the definition of such a structure requires several iterations

UML – dynamic

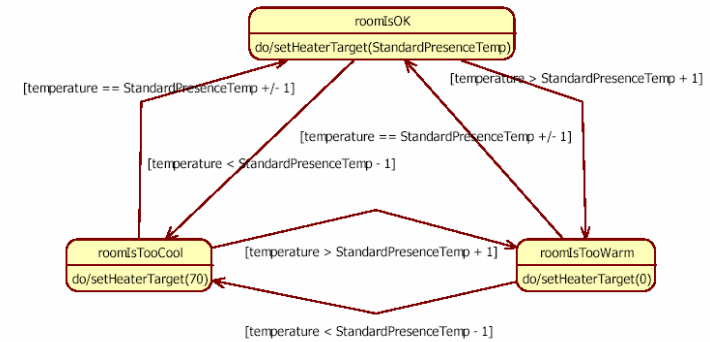
- State charts
- Sequence diagrams

Sequence

Sequence diagram for scenario 11:



State chart



ADL: C2

- Based on Components and Connectors
- Effort to describe more formally the interactions

C2

- A C2 architecture is composed by components, connectors and interconnections
 - ♦ Layered
 - ♦ Elements in an architecture communicate by means of asynchronous messages
 - ♦ Each component may have its own control thread

Components

- A *component* is a unit of computation or a data store (has state)

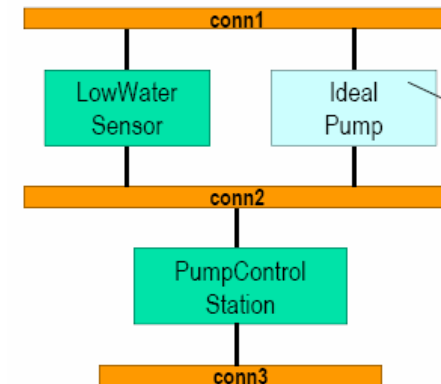
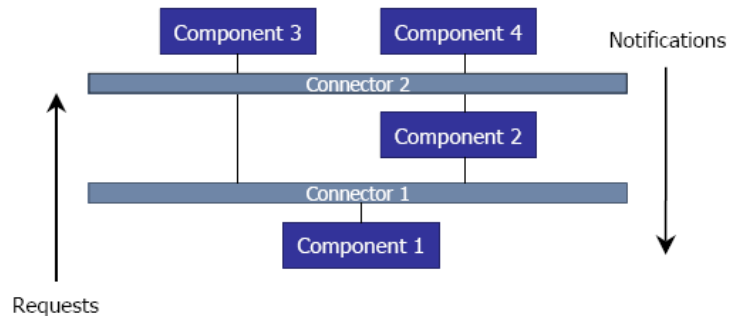
Clients Servers
Databases Filters
Layers ADTs

- A component may be composite
 - ♦ composite components describe a (sub)system

Connectors

- A *connector* is an architectural element that models
 - ♦ interactions among components
 - ♦ rules that govern those interactions
- Simple interactions
 - ♦ procedure calls
 - ♦ shared variable access
- Complex and semantically rich interactions
 - ♦ client-server protocols
 - ♦ database access protocols
 - ♦ asynchronous event multicast
 - ♦ piped data streams

Example



Styles

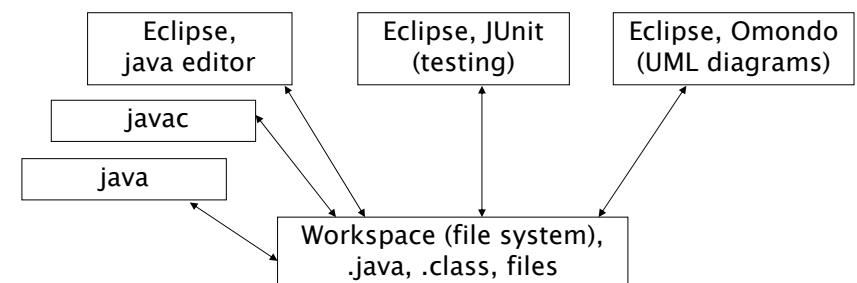
Styles

- A style is a basic strategy used to organize a system
- Styles are recurrent, designers keep reusing them, ex.:
 - ♦ A shared data repository style;
 - ♦ A shared services and servers style (client server, 2tier, 3 tier)
 - ♦ An abstract machine or layered style
- However, a real system is usually influenced by many styles

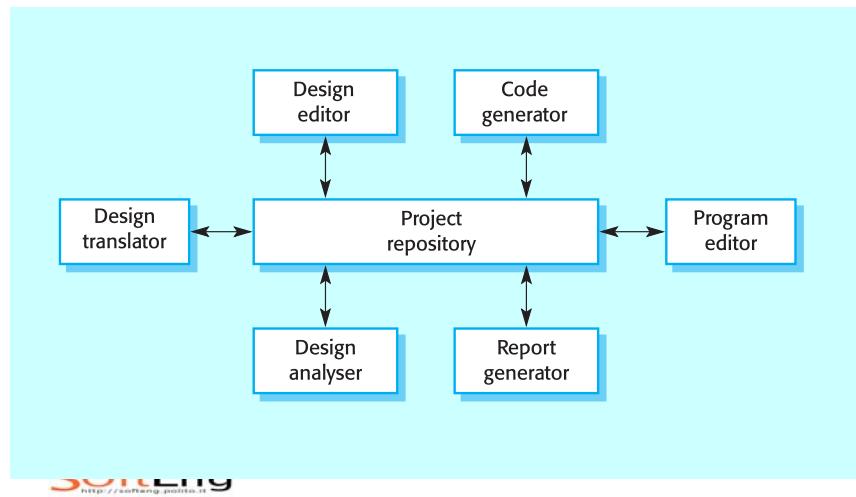
The repository style

- 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.

Eclipse and plugins



CASE toolset architecture



Repository style characteristics

- Advantages
 - ♦ Efficient way to share large amounts of data;
 - ♦ Sub-systems need not be concerned with how data is produced Centralised management e.g. backup, security, etc.
 - ♦ Sharing model is published as the repository schema.
- Disadvantages
 - ♦ Sub-systems must agree on a repository data model. Inevitably a compromise;
 - ♦ Data evolution is difficult and expensive;
 - ♦ No scope for specific management policies;
 - ♦ Difficult to distribute efficiently.

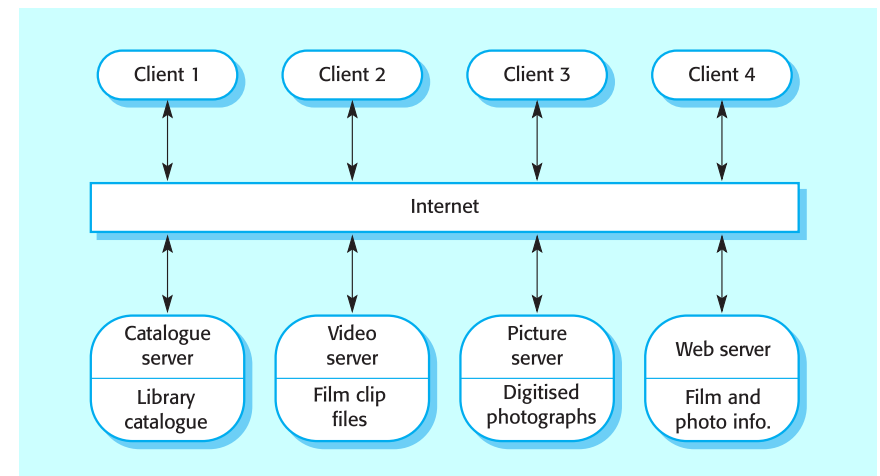
SoftEng
http://softeng.pdinfo.it

Client-server model

- ♦ 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.

SoftEng
http://softeng.pdinfo.it

Film and picture library



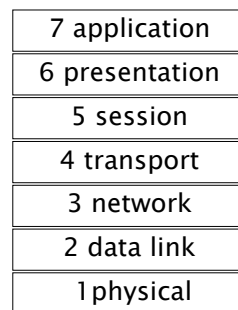
Client-server characteristics

- 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.

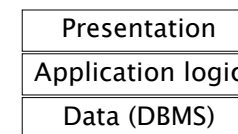
Abstract machine (layered) model

- ♦ 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 artificial to structure systems in this way.

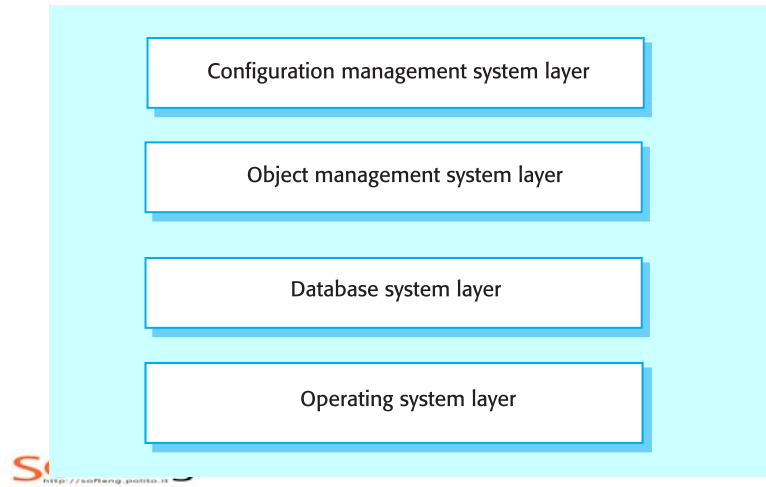
ISO Osi model



3 tier architecture



Version management system



Verification

Verification

- Functional requirements
 - ◆ Traceability matrix
 - ◆ Scenarios executed on architecture
 - ◆ Inspection
- Non functional requirements
 - ◆ Performance
 - Scenarios enriched with time model
 - ◆ (Inspection)

Traceability matrix

	Away ManagementStrategy	Boiler	C Room	DefaultHomesSettings	Env	Environment	HouseController	InvalidTimeException	PhysBoiler	PresenceManagementStrategy	Room	RoomManagementStrategy	RoomSettings	SetRoomParametersActivity	SetRoomParametersDialog	XMLSettings
Temp-UR-F1											X		X	X	X	X
Temp-UR-F2											X		X	X	X	X
Temp-UR-F3											X		X	X	X	X
Temp-UR-F4											X		X	X	X	X
Temp-UR-F5											X		X	X	X	X
Temp-UR-F6		X	X		X	X	X		X	X	X	X				
Temp-UR-F7	X	X	X		X	X	X		X	X	X	X				
Temp-UR-F8		X	X		X	X	X		X	X	X	X				
Temp-UR-F9		X	X		X	X	X		X	X	X	X				
Temp-UR-F10	X	X	X		X	X	X		X		X	X				
Temp-UR-F11								X							X	
Temp-UR-F12				X									X	X		
Temp-UR-F13	X	X	X		X	X	X		X		X	X				
Temp-UR-F14	X		X		X	X	X			X	X	X				
Temp-UR-F15			X		X	X	X				X	X	X			
Temp-UR-F16	X									X						
Temp-UR-F17			X	X			X				X					X
Temp-UR-F18			X				X		X							
UR-Inv 1	X	X	X		X	X	X		X		X	X				
UR-Inv 2		X	X		X	X	X		X	X	X	X				

Key points

- Architecture
 - ♦ defining high level components and their control, communication model
 - ♦ Tools: UML or ADL models, structural and dynamic
 - ♦ Styles: Layered, client server (2 tier, 3 tier), peer to peer, shared repository
- Design
 - ♦ Define internals of components
 - ♦ Tools: UML models
 - ♦ Design patterns

Key points

- Verification
 - ♦ inspections
 - Architecture can satisfy functional properties (as defined in requirements doc)?
 - Traceability matrixes
 - Scenario execution
 - Architecture can satisfy non functional requirements?
 - Enriched scenarios
 - ♦ build prototype