SW Design and Implementation

Architectural Design

2. Software design and implementation

FASE di PROGETTAZIONE: come fare? (non cosa fare?)

- The process of converting the system specification into an executable system
- Software design → Progettazione del sw
  - **INPUT**: Le specifiche
  - **Process**: Design a software structure that realises the specification
  - **OUTPUT**: un Documento con il Progetto (risultato dell'attività di progettazione)
- Implementation (attività di programmazione)
  - Translate this structure into an executable program
- The activities of design and implementation are closely related and may be inter-leaved

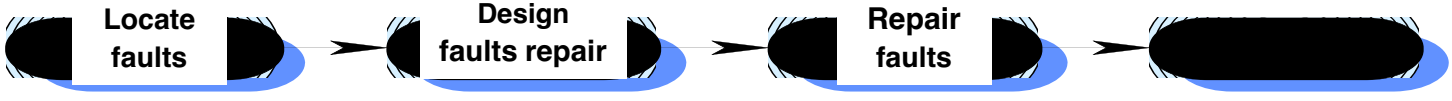
Implementation =
1.Programming + 2.Debugging

- 2 attività distinte, ancorchè interleaved -

Scrivo un pò di codice e poi vedo se quel pezzo di codice che ho scritto funziona

- 1. Translating a design into a program and
  2. removing faults/BUGS from that program
- Programming is a personal activity - there is no generic/standardized programming process
- Programmers carry out some program testing to discover faults/bugs in the program and remove these faults in the debugging process

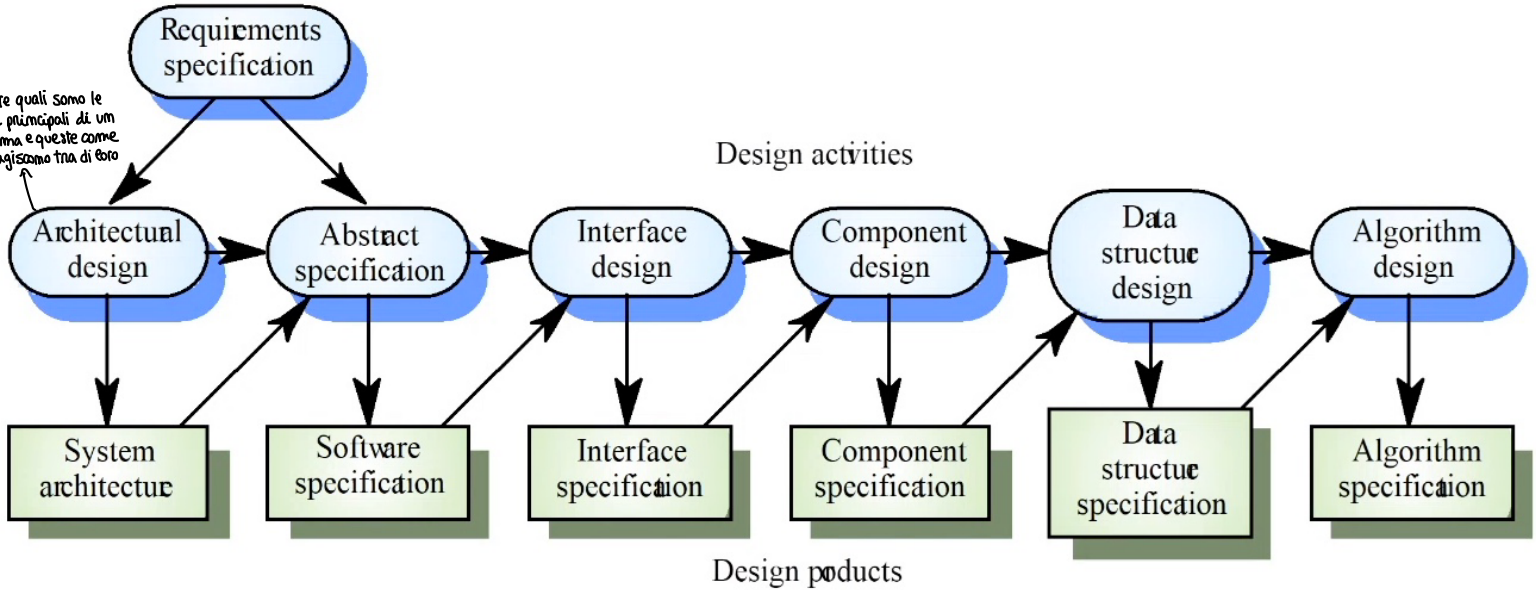
The debugging process



The Design Process

The software design process

capire quali sono le parti principali di un sistema e queste come interagiscono tra di loro

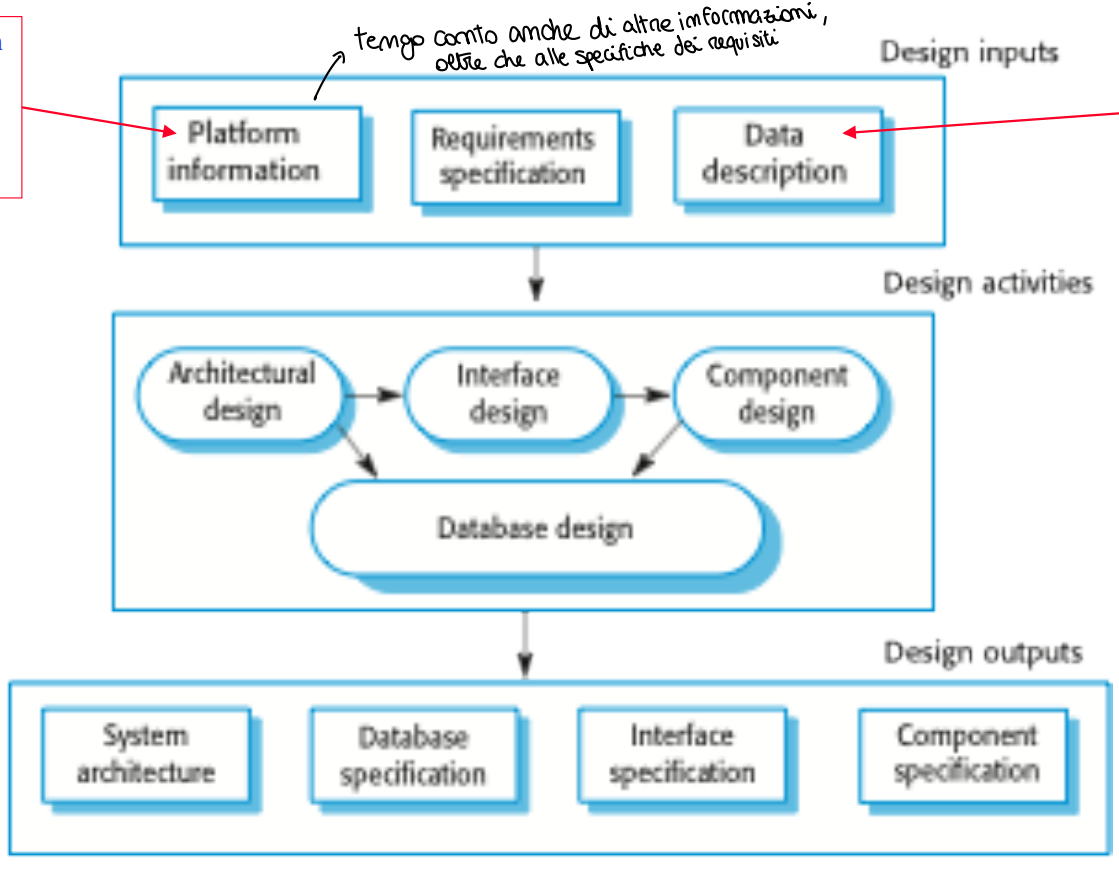


Overall Design Phases

- ***Architectural design:*** Identify sub-systems. → capire quali sono le parti principali di un sistema e queste come interagiscono tra di loro (TOP - DOWN)
- ***Abstract specification:*** Specify sub-systems.
- ***Interface design:*** Describe sub-system interfaces.
- ***Component specification/design&Modular decomposition:*** Decompose sub-systems into components (modules).
- ***Data structure/Data Base design:*** Design data structures to hold problem data (within modules)
- ***Algorithm design:*** Design algorithms for problem functions (within modules)

A more general model of the design process

Operating System
DataBase
MiddleWare,
System SW
Application SW



Data stored in other DataBases

Design activities

- *Architectural design,* where you identify the overall structure of the system, the principal components (sometimes called sub-systems or modules), their relationships and how they are distributed.
- *Interface design,* where you define the interfaces between system components.
- *Component design,* where you take each system component and design how it will operate.
- *Database design,* where you design the system data structures and how these are to be represented in a database.

Structured design methods/ metodologie di sviluppo SW

- Systematic approaches to developing a software design
- The design is usually documented as a set of system **graphical models**
- Possible modelling languages
  - Data-flow model
  - Entity-relation-attribute model
  - Structural model
  - Object models
  - Petri's nets
  - BPMN
  - …
- Current mostly used methods: UML, O-O, …MDD – Model Driven Design (o MBD …Based…)

La Progettazione del SW: un'attività del tutto creativa o sistematica

La progettazione ha certamente degli elementi di creatività, ma nell'affrontare un problema specifico il progettista ha a disposizione anche tutta una serie di schemi di riferimento, sviluppati nel corso degli anni e risultato dell'esperienza e delle best practice, che offrono delle possibili soluzioni efficaci e riusabili per vari aspetti dell'attività di progettazione.

Tali schemi si situano a vario livello di astrazione/genericità/specificità tecnica/completezza e strutturazione della descrizione/, e sono noti con diverse terminologie: **architetture generiche**, **modelli di progettazione**, **design pattern**, **stili architetturali**, **modelli architetturali**, …

Modelli di Progettazione

- STRUTTURAZIONE DELL'ARCHITETTURA
  - 1.1 REPOSITORY
  - 1.2 CLIENT-SERVER
  - 1.3 MACCHINE ASTRATTE
- CONTROLLO
  - 2.1 CENTRALIZZATO
    - 2.1.1 CALL/RETURN
    - 2.1.2 MANAGER MODEL
  - 2.2 EVENT DRIVEN
    - 2.2.1 BROADCAST
    - 2.2.2 INTERRUPT-DRIVEN
- SCOMPOSIZIONE MODULARE
  - 3.1 OBJEC-ORIENTED
  - 3.2 FUNZIONALE (DATA FLOW, PIPELINE, STRUCTURAL)
- ARCHITETTURE SPECIFICHE DEL DOMINIO
  - 4.1 GENERICHE [MVC]
  - 4.2 REFERENCE [ISO/OSI]
- ARCHITETTURE APPLICATIVE GENERICHE
  - 1. TRANSACTION PROCESSING [information systems, e-commerce]
  - 2. LANGUAGE PROCESSING [compiler, interpreter, …]

Architectural Design

→ processo che permette di arrivare a definire qual è l'architettura di un sistema SW

- The design process for identifying the sub-systems making up a system **and** the framework for sub-system control and communication is *architectural design*
- The output of this design process is a description of the *software architecture*

Architectural design

- An early stage of the system design process
- Represents the link between specification and design processes
- Often carried out in parallel with some specification activities
- It involves identifying major system components and their communications

Advantages of explicit architecture

- Stakeholder communication
  - Architecture may be used as a focus of discussion by system stakeholders
- System analysis
  - It requires a first level of analysis that allows preliminary understanding whether the system can meet its functional and non-functional requirements (performance, reliability, ..)
- Large-scale reuse
  - The architecture may be reusable across a range of systems
- Important aspects of the system are considered early in the design
- It allows planning of subsequent phases

Architectural design process

3 aspetti da tenere conto:

1. System structuring → come suddividere il sistema
   - The system is decomposed into several principal sub-systems and communications between these sub-systems are identified
2. Control modelling → problema del controllo
   - A model of the control relationships between the different parts of the system is established
3. Modular decomposition → come si possono costruire le sottoparti
   - The identified sub-systems are decomposed into modules

Le scelte architetturali influenzano parametri importanti, in modo 'conflittuale' ? trade-off

→ se favorisco uno, sfavorisco un altro

- **Performance** → devo ridurre il numero di sottofunzioni in modo da ridurre il numero di passaggi di parametri da una funzione all'altra
  - Localise operations to minimise sub-system communication (large grain components)
- **Security (protezione)** → più stratifico e più sono sicuro che le cose a basso livello siamo inaccessibili, anche se più stratifico più ci sono passaggi di dati (↑ sicurezza ↓ performance)
  - Use a layered architecture with critical assets in inner layers
- **Safety (sicurezza)**
  - Isolate safety-critical components
- **Availability (disponibilità a fornire servizio)**
  - Include redundant components in the architecture
- **Maintainability** → se non divido in tanti moduli io so che se devo cambiare qualcosa devo andare a toccare solo pochi moduli → devo modificare poco
  - Use fine-grain, self-contained components

Architectural conflicts/trade-off

- 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.

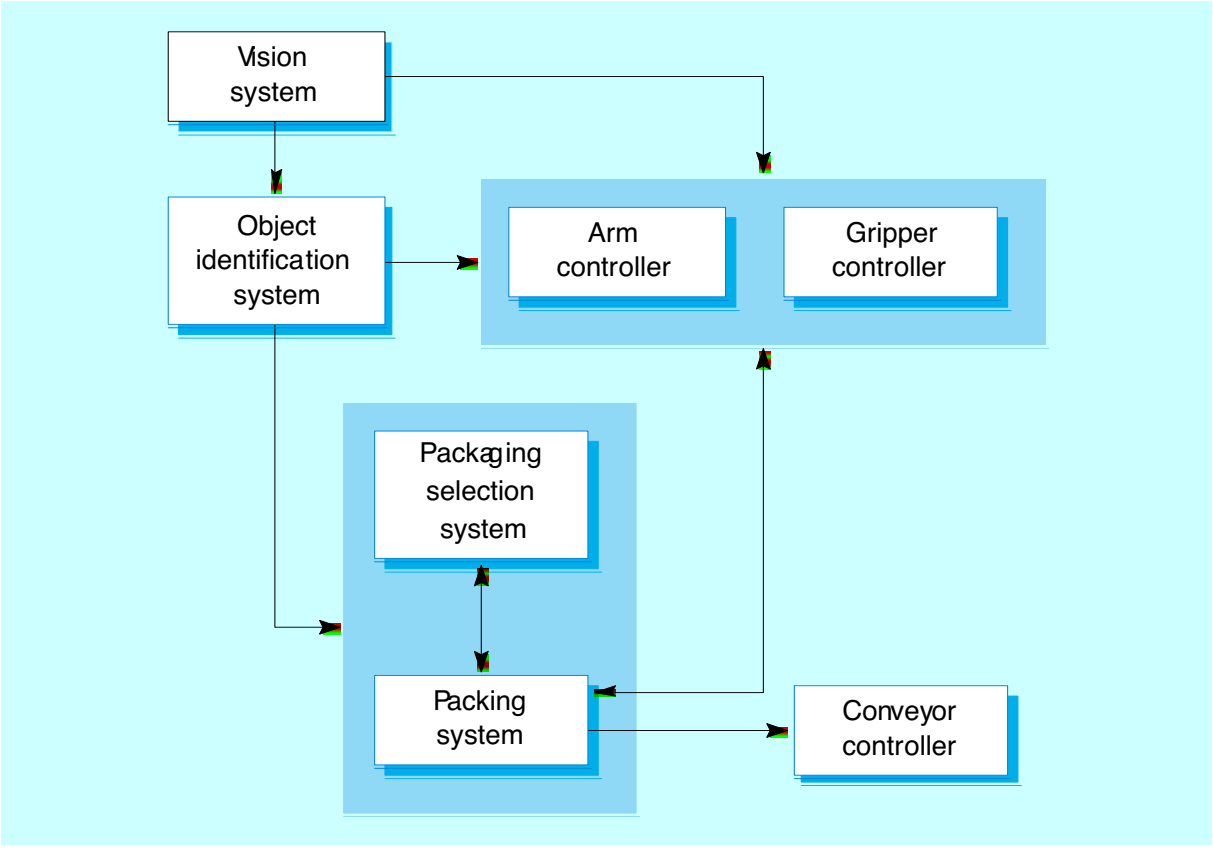
How to represent system structuring

- 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

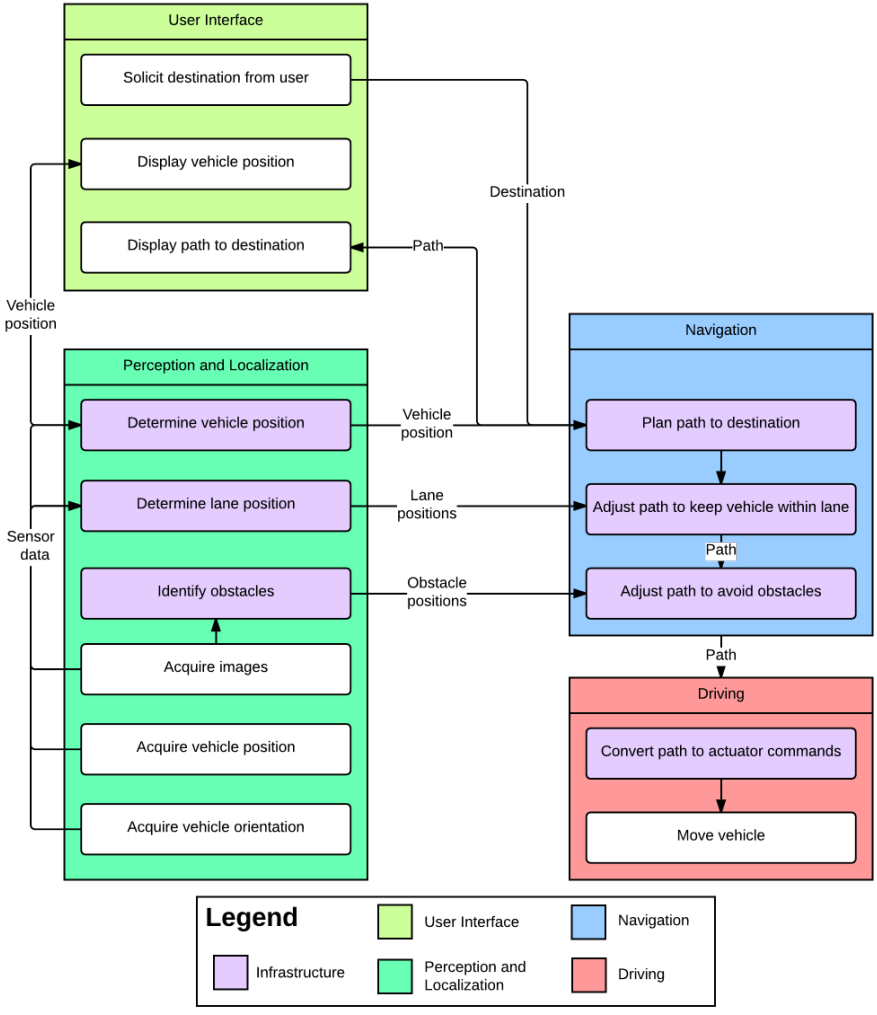
Box and line diagrams

- 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.
- Tipicamente I vari box (rettangoli) sono connessi da frecce che rappresentano I dati che escono/entrano da/in un sottosistema/modulo
  - NATURA funzionale, DFD

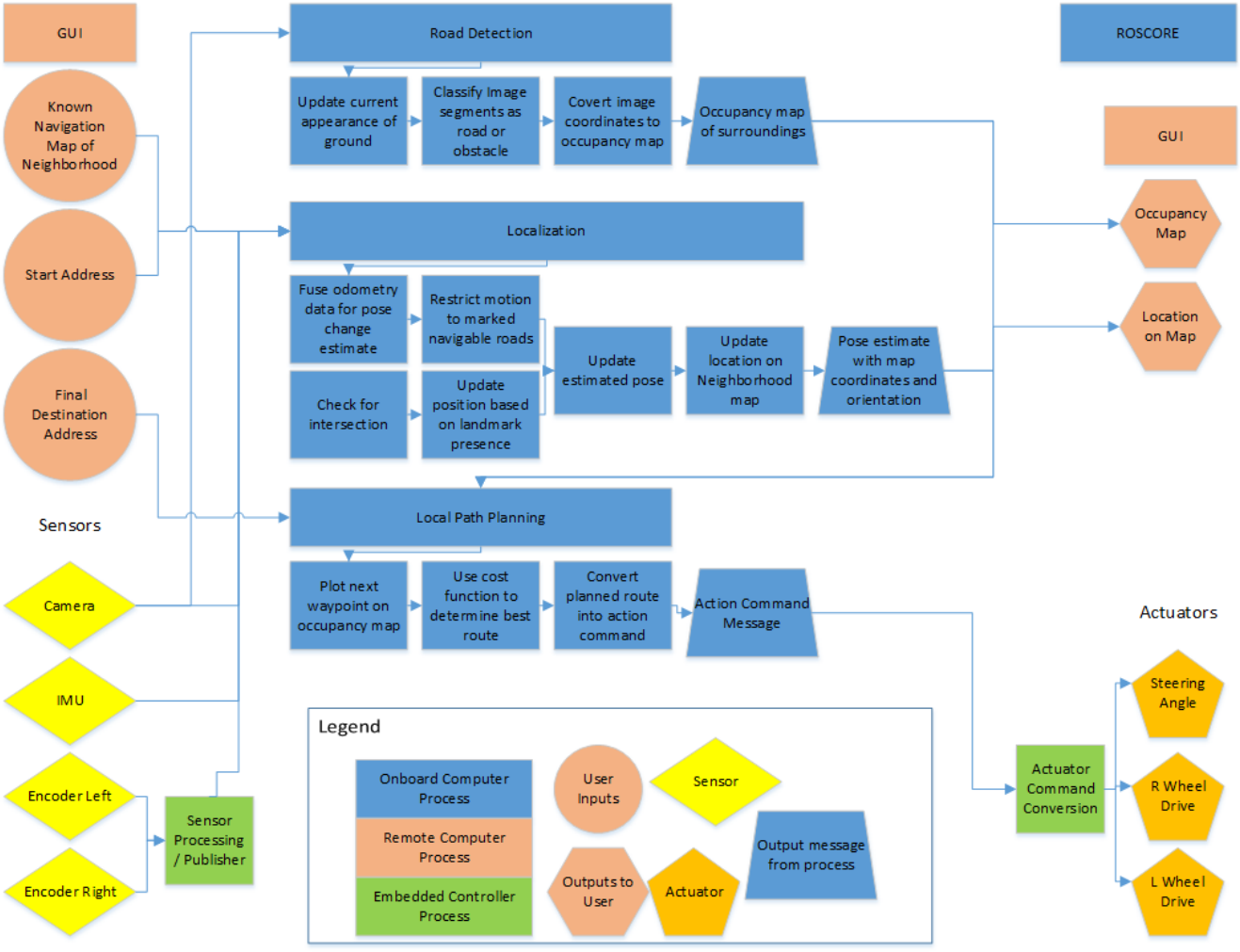
Packing robot control system



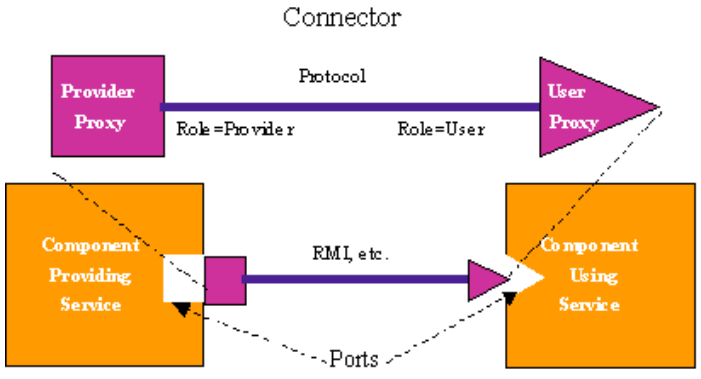
Autonomous Vehicle Control System - 1

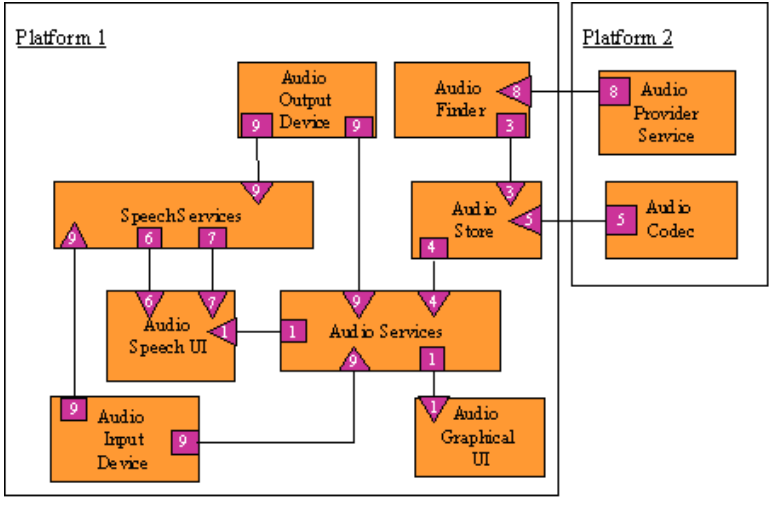


Autonomous Vehicle Control System - 2



ADLs - Architectural Description langauges





Architecture reuse

- Systems in the same domain often have similar architectures that reflect domain concepts.
- Application product lines are built around a core architecture with variants that satisfy particular customer requirements.
- The architecture of a system may be designed around one of more architectural patterns or 'styles'* ** o model** .
  - These capture the essence of an architecture and can be instantiated in different ways.

(*) Analogie e differenze con l'architettura degli edifici: qui è accettata l'eterogeneità!!

(**) Termini generali/generic, nn specificati in dettaglio tecnicamente

Architectural patterns

- Patterns are a means of representing, sharing and reusing knowledge.
- An architectural pattern is a stylized description of good design practice, which has been tried and tested in different environments.
- Patterns should include information about when they are and when the are not useful.
- Patterns may be represented using tabular and graphical descriptions.
- Design Patterns are used also in OO Programming
- Termine più tecnico e specifico

- **STRUTTURAZIONE DELL'ARCHITETTURA**
  - **1.1 REPOSITORY**
  - **1.2 CLIENT-SERVER**
  - **1.3 MACCHINE ASTRATTE**
- CONTROLLO
  - 2.1 CENTRALIZZATO
    - 2.1.1 CALL/RETURN
    - 2.1.2 MANAGER MODEL
  - 2.2 EVENT DRIVEN
    - 2.2.1 BROADCAST
    - 2.2.2 INTERRUPT-DRIVEN
- SCOMPOSIZIONE MODULARE
  - 3.1 OBJEC-ORIENTED
  - 3.2 FUNZIONALE (DATA FLOW, PIPELINE, STRUCTURAL)
- ARCHITETTURE SPECIFICHE DEL DOMINIO
  - 4.1 GENERICHE [MVC]
  - 4.2 REFERENCE [ISO/OSI]
- ARCHITETTURE APPLICATIVE GENERICHE
  - 1. TRANSACTION PROCESSING [information systems, e-commerce]
  - 2. LANGUAGE PROCESSING [compiler, interpreter, …]

Modelli di Progettazione

1. System structuring (organisation)

- Reflects the basic strategy that is used to structure a system.
- Three organisational styles are widely used:
  1. A shared data repository style;
  2. A shared services and servers style;
  3. An abstract machine or layered style.

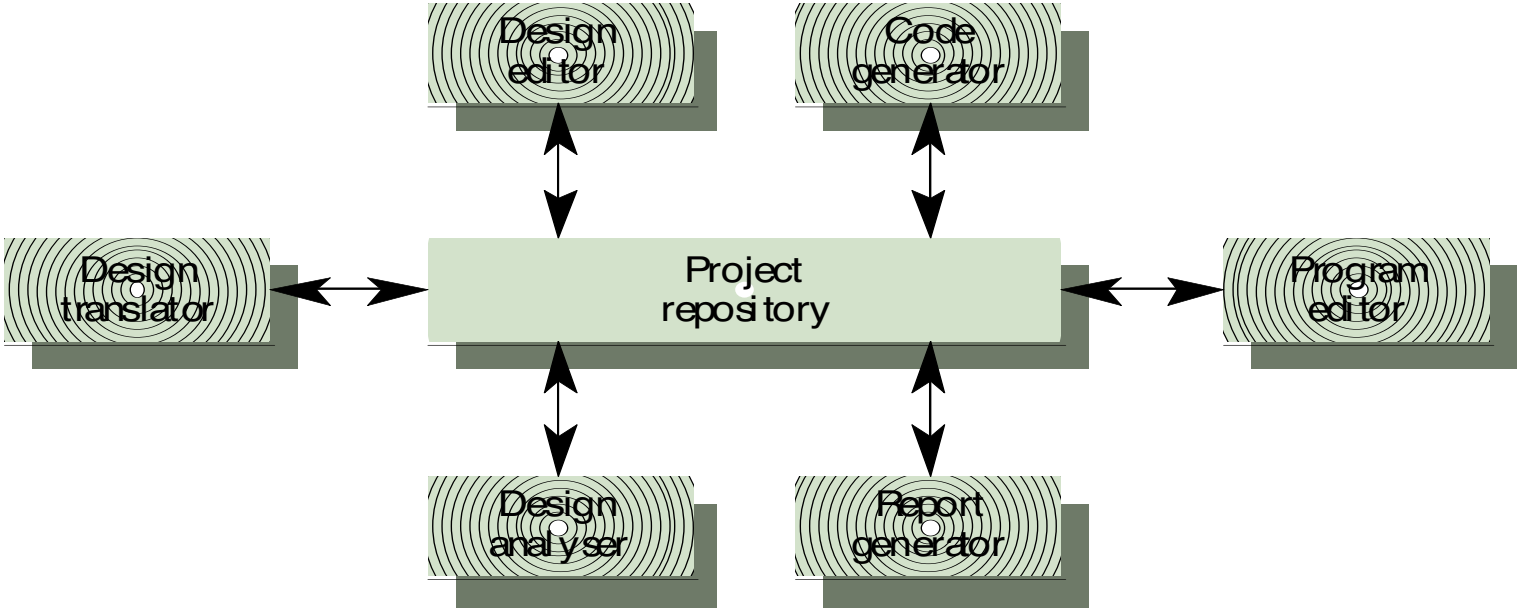
1.1 The repository model

repository centralizzato

devono avvenire scambi di dati tra sotto-sistemi, dove vengono tenuti i dati? centralizzato (tutti vanno a prendere dal DB) / distribuito

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

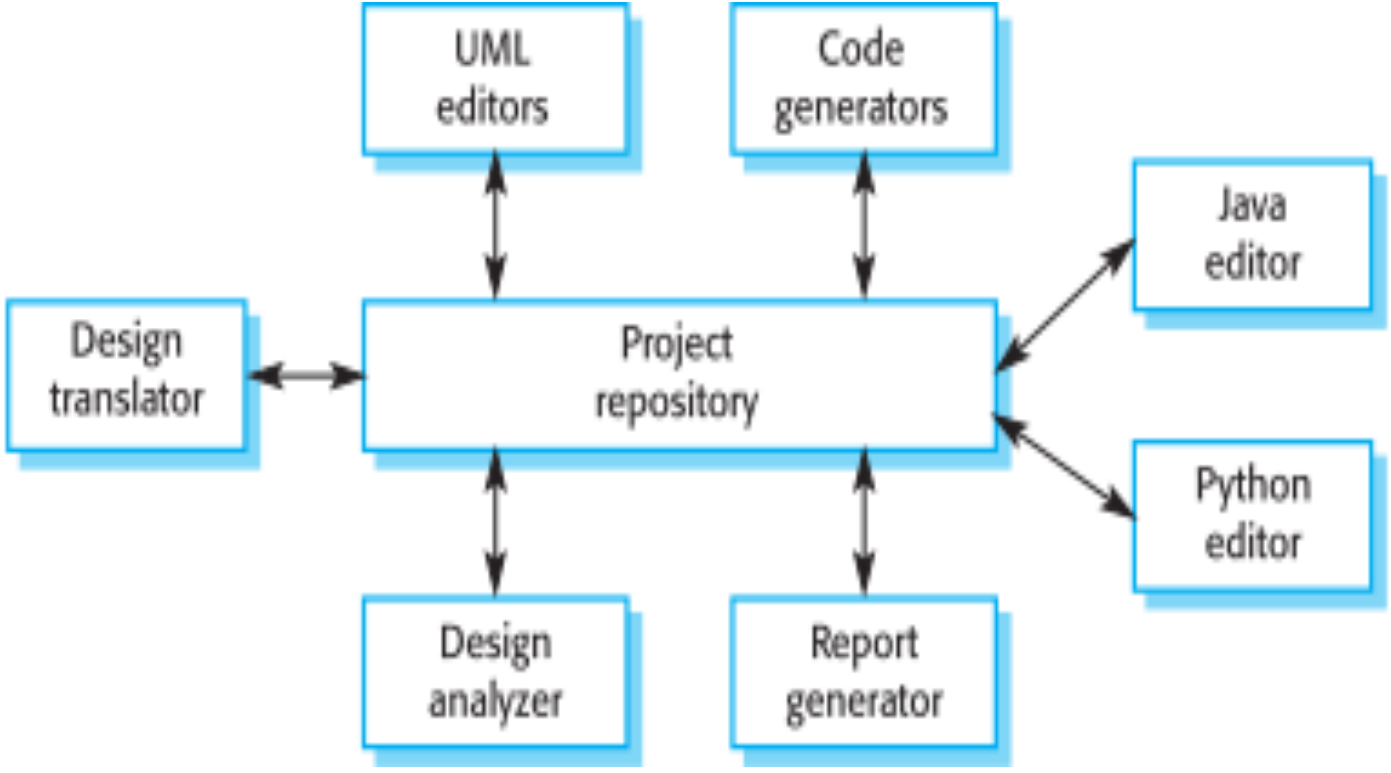
Example: CASE toolset architecture



The Repository pattern

Name	Repository
Description	All data in a system is managed in a central repository that is accessible to all system components. Components do not interact directly, only through the repository.
Example	Figure 6.9 is an example of an IDE where the components use a repository of system design information. Each software tool generates information which is then available for use by other tools.
When used	You should use this pattern when you have a system in which large volumes of information are generated that has to be stored for a long time. You may also use it in data-driven systems where the inclusion of data in the repository triggers an action or tool.
Advantages	Components can be independent—they do not need to know of the existence of other components. Changes made by one component can be propagated to all components. All data can be managed consistently (e.g., backups done at the same time) as it is all in one place.
Disadvantages	The repository is a single point of failure so problems in the repository affect the whole system. May be inefficiencies in organizing all communication through the repository. Distributing the repository across several computers may be difficult.

Example: A repository architecture for an IDE (Integrated Development Environments)



Repository model characteristics

- Advantages → risparmio un sacco di memoria perché è centralizzata
  - 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

Ai tempi nostri vengono più utilizzati i modelli distribuiti, grazie all'aumento della capacità delle memorie

1.2 Client-server architecture

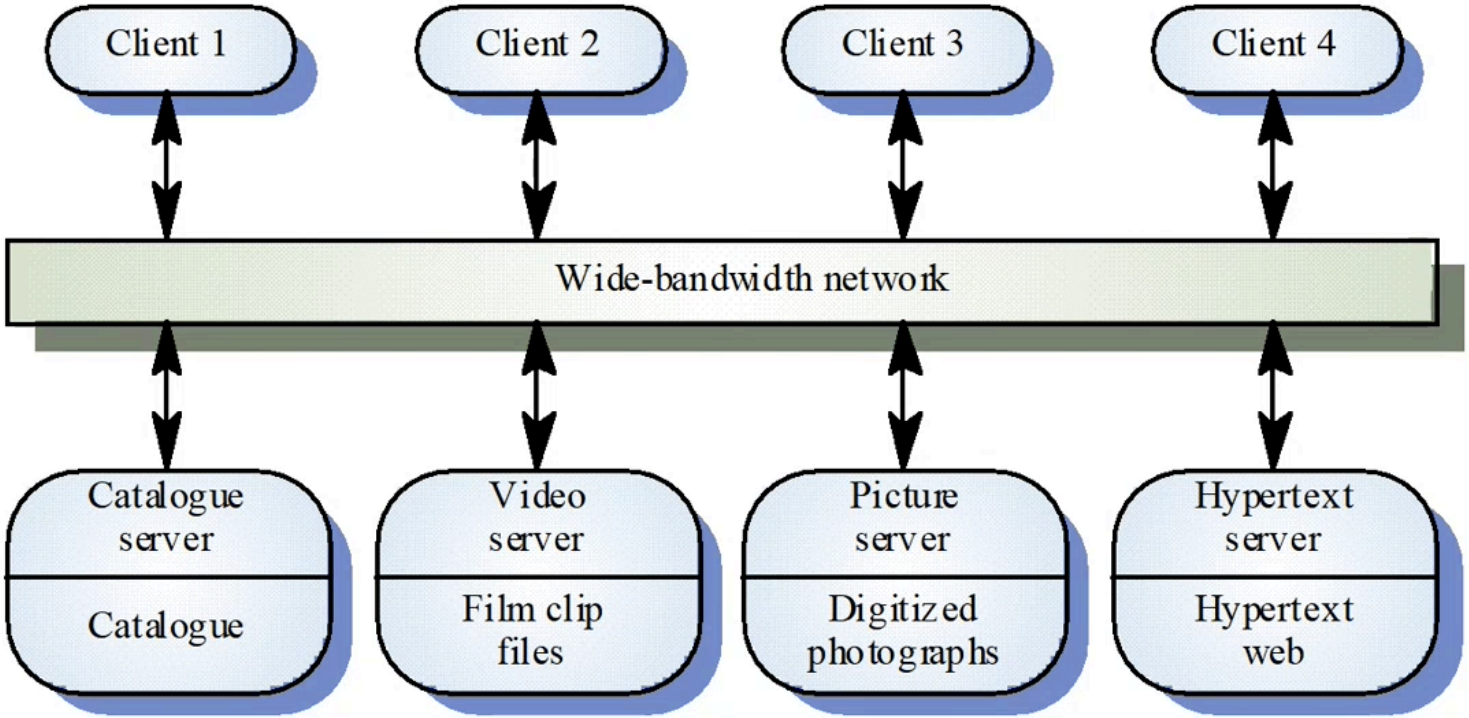
- Distributed system model which shows how data and processing is distributed across a range of components

1. Set of stand-alone **servers** which provide specific services such as printing, data management, etc.
2. Set of **clients** which call on these services
3. A **Network** which allows clients to access servers

↳ client chiede un servizio e un server gli offre il servizio

↳ se la rete non è veloce è un collo di bottiglia e l'efficienza verrebbe a cadere

Film and picture library



The Client–server architectural pattern

Name	Client-server
Description	In a client–server architecture, the functionality of the system is organized into services, with each service delivered from a separate server. Clients are users of these services and access servers to make use of them.
Example	Figure 6.11 is an example of a film and video/DVD library organized as a client–server system.
When used	Used when data in a shared database has to be accessed from a range of locations. Because servers can be replicated, may also be used when the load on a system is variable.
Advantages	The principal advantage of this model is that servers can be distributed across a network. General functionality (e.g., a printing service) can be available to all clients and does not need to be implemented by all services.
Disadvantages	Each service is a single point of failure so susceptible to denial of service attacks or server failure. Performance may be unpredictable because it depends on the network as well as the system. May be management problems if servers are owned by different organizations.

Client-server characteristics

- Advantages ogni server ha la propria gestione dei dati
  - 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
  - If no central register of names and services - it may be hard to find out what servers and services are available

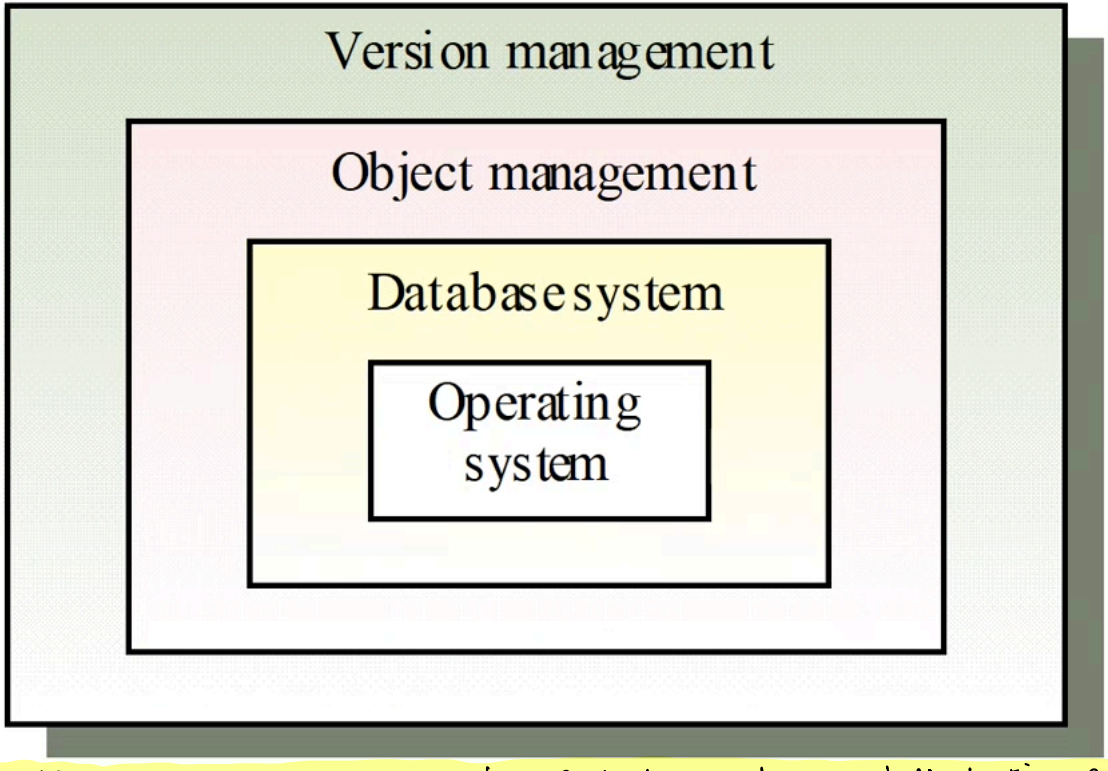
1.3 Abstract machine model/ Layered Architecture

- Used to model the interfacing of sub-systems
- Organises the system into a set of layers (or *abstract machines*) each of which provides 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

- Ciascun LAYER riceve richieste di servizio SOLO dal layer sovrastante e può richiedere servizi SOLO al layer sottostante.
- In linea di principio NON sono ammessi scambi tra layer non adiacenti!!!

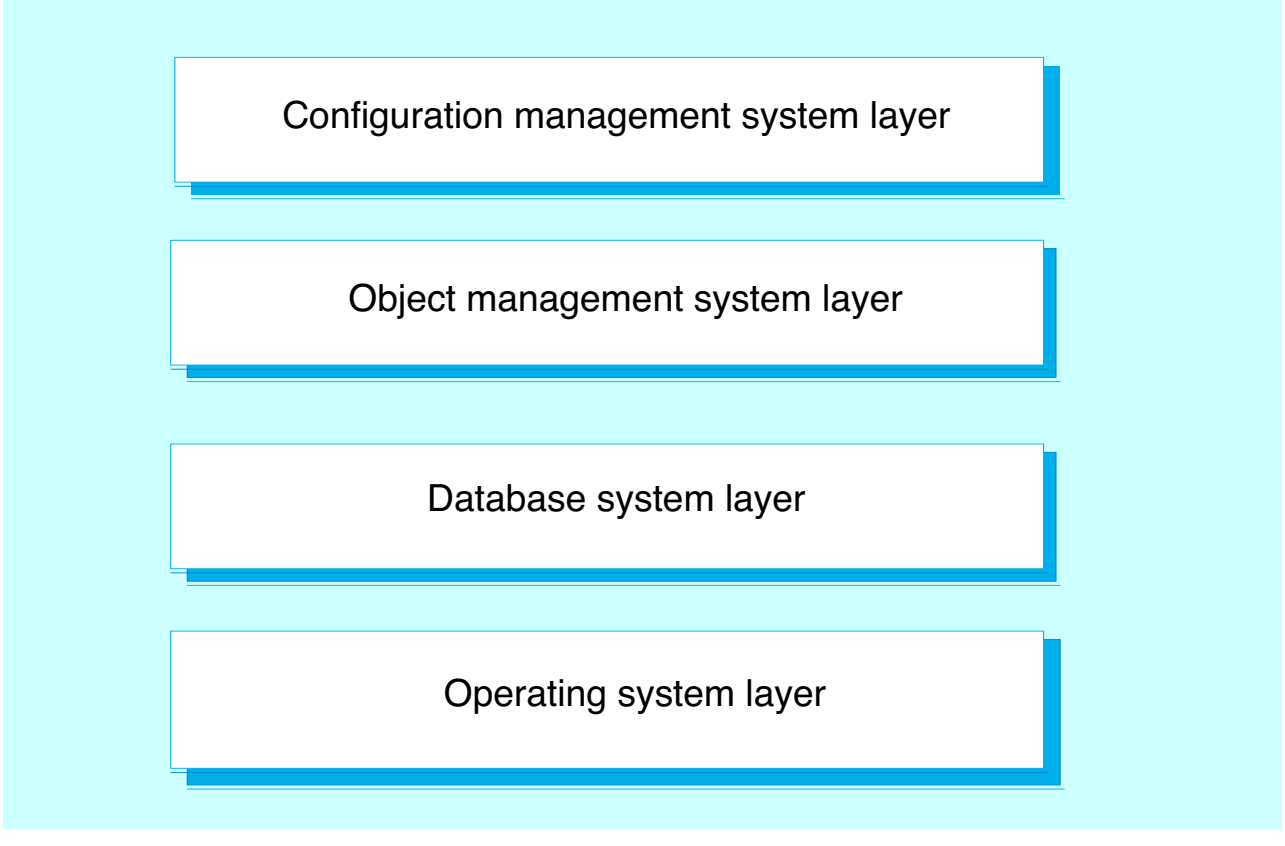
organizzare partendo dai dettagli dello strato più esterno → visione più vicina a quella utente
i livelli più esterni "sfruttano" gli strati più interni per svolgere i loro compiti

Version management system

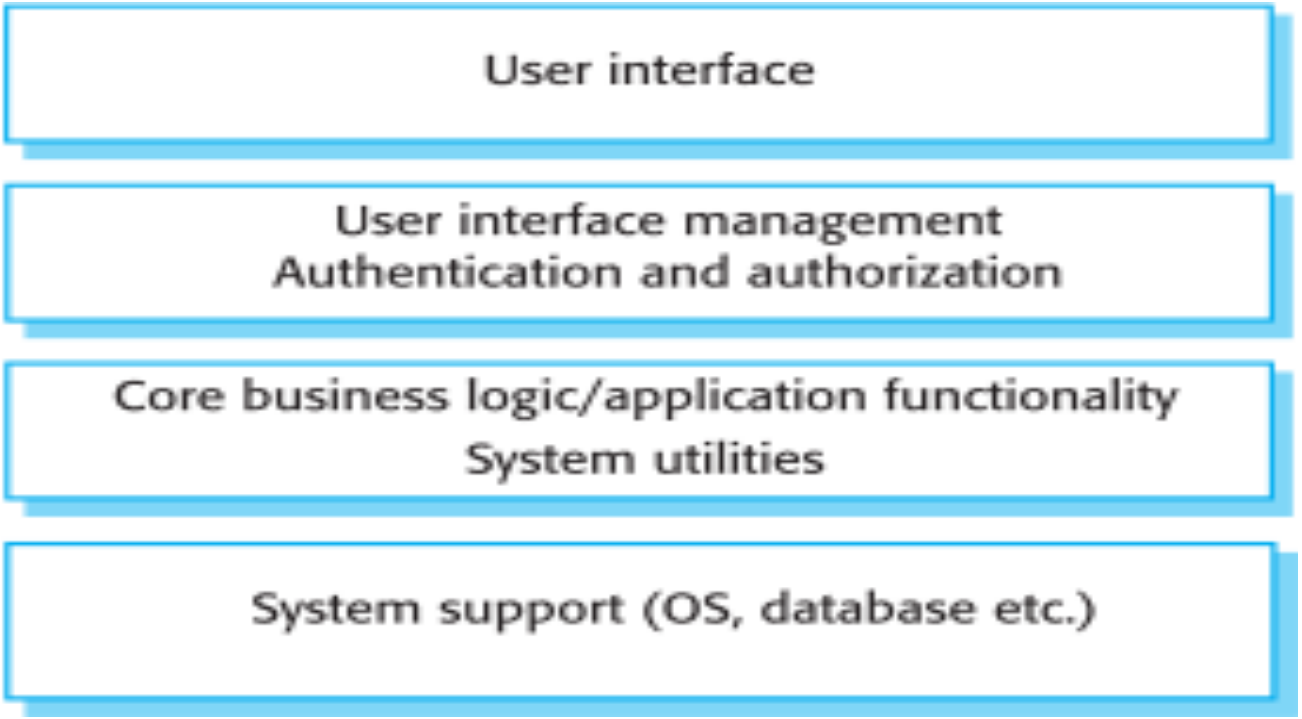


→ posso sviluppare un singolo strato con tecnologie diverse da quelle degli strati sopra e sotto.
l'importante è che riescano a comunicare → MAGGIORE FLESSIBILITÀ => MANUTENZIONE PIÙ DIFFICILE

Version management system



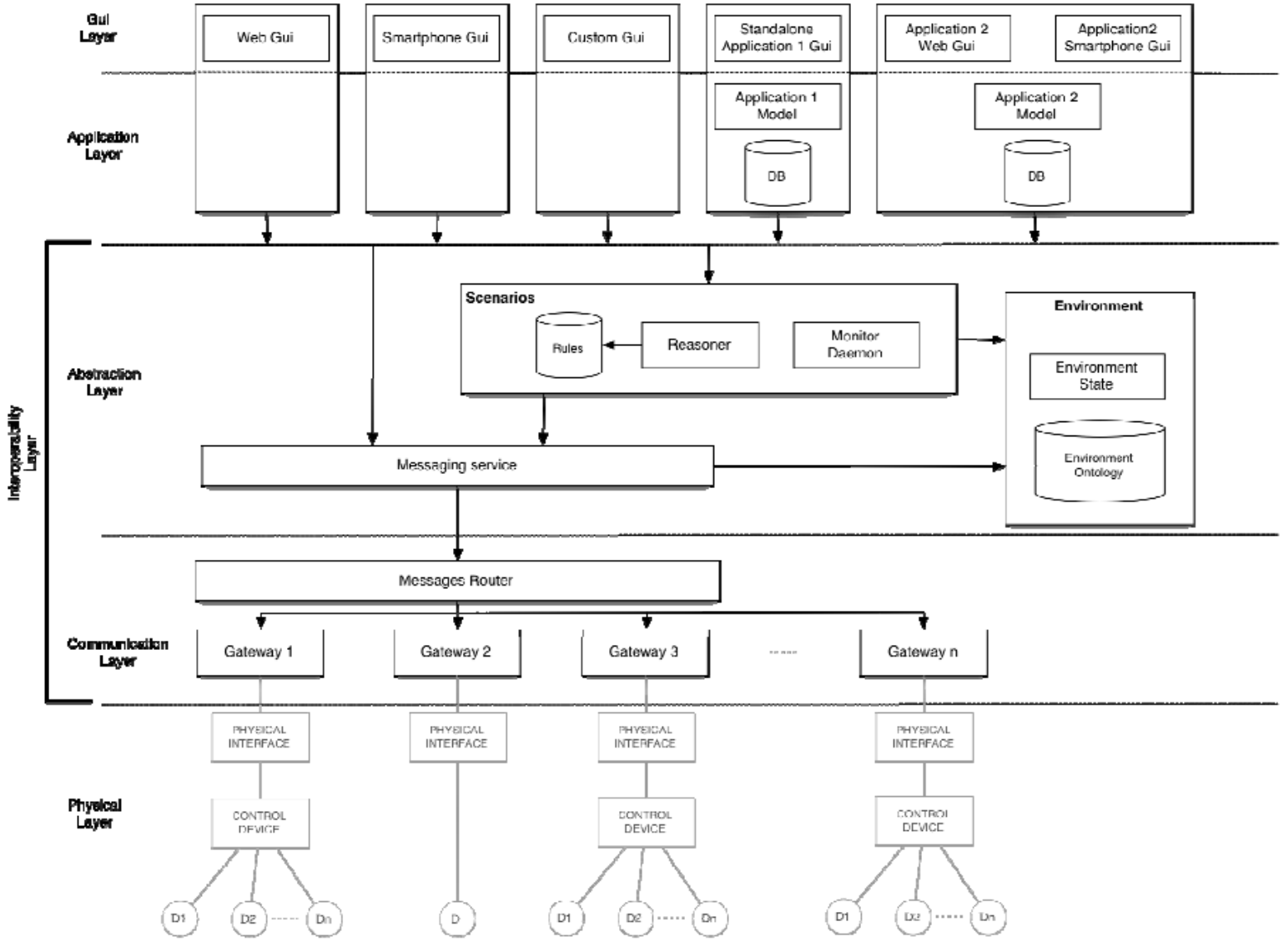
A generic layered architecture



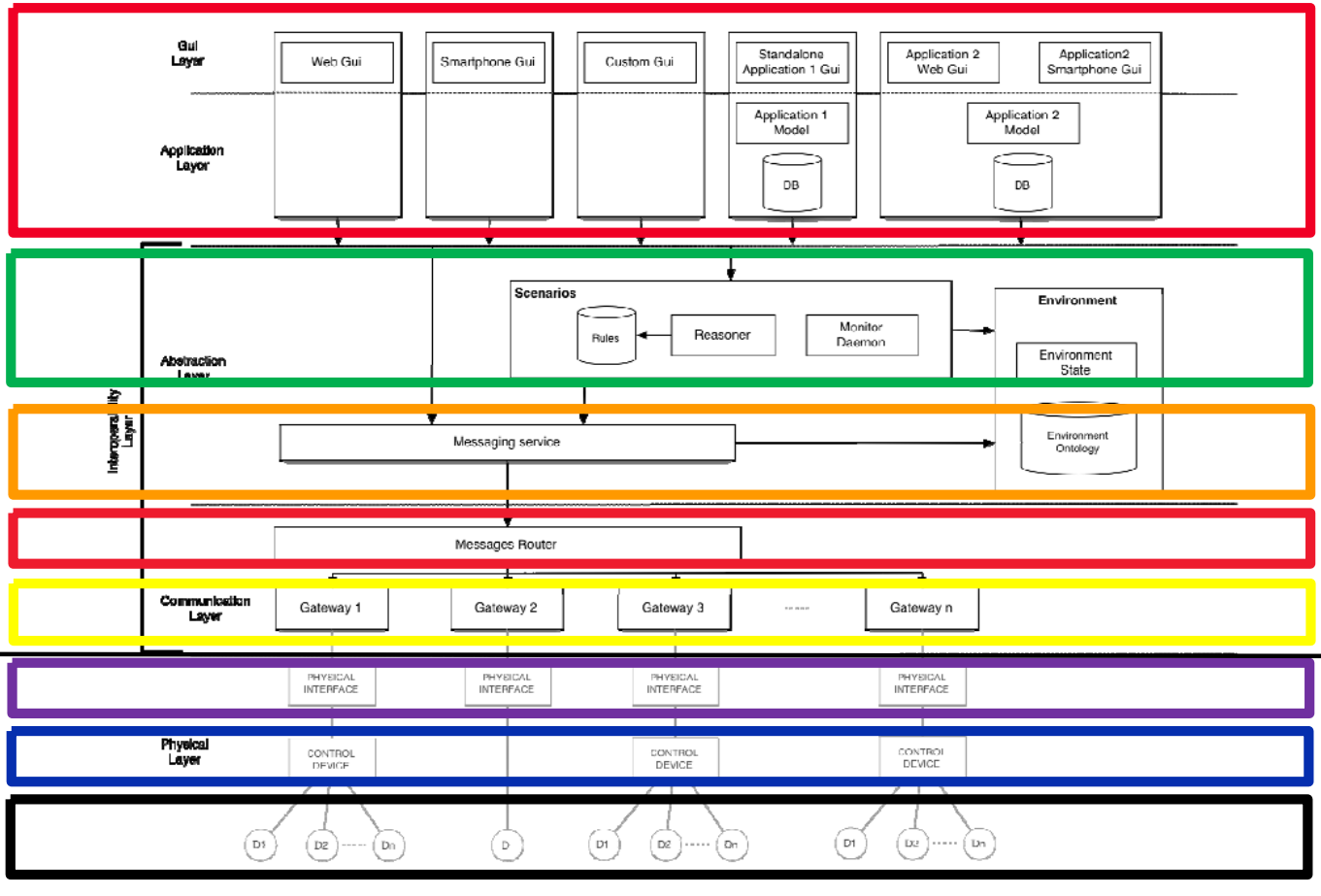
The Layered architecture pattern

Name	Layered architecture
Description	Organizes the system into layers with related functionality associated with each layer. A layer provides services to the layer above it so the lowest-level layers represent core services that are likely to be used throughout the system. See Figure 6.6.
Example	A layered model of a system for sharing copyright documents held in different libraries, as shown in Figure 6.7.
When used	Used when building new facilities on top of existing systems; when the development is spread across several teams with each team responsibility for a layer of functionality; when there is a requirement for multi-level security.
Advantages	Allows replacement of entire layers so long as the interface is maintained. Redundant facilities (e.g., authentication) can be provided in each layer to increase the dependability of the system.
Disadvantages	In practice, providing a clean separation between layers is often difficult and a high-level layer may have to interact directly with lower-level layers rather than through the layer immediately below it. Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer.

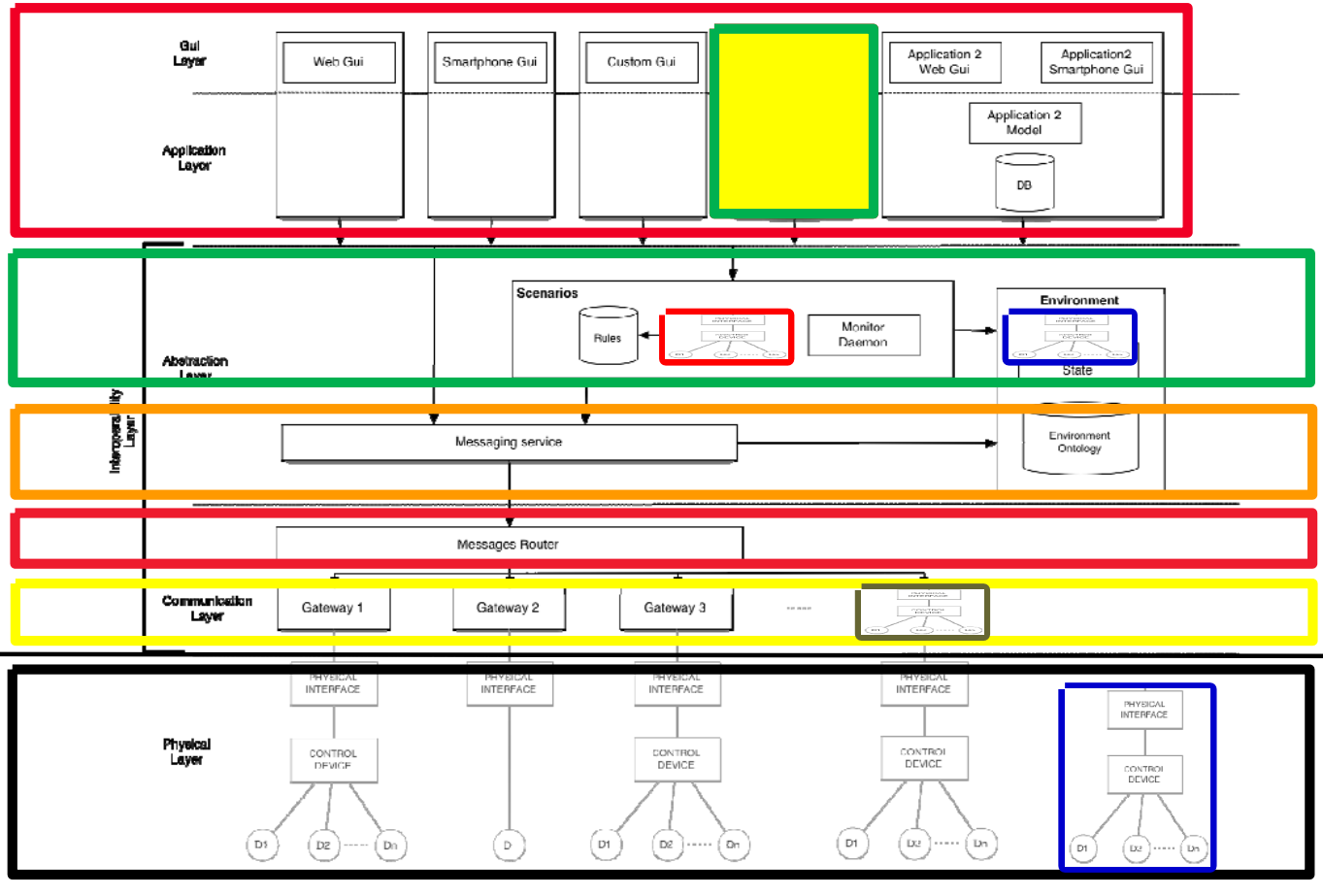
Esempio dal Progetto Domotica FVG


Gui Layer
Web Gui
Smartphone Gui
Custom Gui
Standalone Application 1 Gui
Application 2 Web Gui
Application2 Smartphone Gui
Application Layer
Application 1 Model
DB
Application 2 Model
DB
Interoperability Layer
Abstraction Layer
Scenarios
Rules
Reasoner
Monitor Daemon
Environment
Environment State
Environment Ontology
Messaging service
Messages Router
Communication Layer
Gateway 1
Gateway 2
Gateway 3
Gateway n
PHYSICAL INTERFACE
PHYSICAL INTERFACE
PHYSICAL INTERFACE
PHYSICAL INTERFACE
Physical Layer
CONTROL DEVICE
CONTROL DEVICE
CONTROL DEVICE
D1
D2
Dn
D
D1
D2
Dn
D1
D2
Dn

Un'Architettura a livelli disaccoppiati



Un'Architettura a livelli disaccoppiati



I Benefici

INTEGRABILITÀ DI COMPONENTI E MODULI ETEROGENEI

FLESSIBILITÀ NELLA PROGETTAZIONE

MODIFICABILITÀ ED ESTENDIBILITÀ DEL SISTEMA, ANCHE CON MODULI INTELLIGENTI

SINERGIA E COOPERAZIONE TRA OPERATORI

COMUNITA' APERTA DI SVILUPPO

GENERAZIONE DI CONOSCENZA E KNOW-HOW

- STRUTTURAZIONE DELL'ARCHITETTURA
  - 1.1 REPOSITORY
  - 1.2 CLIENT-SERVER
  - 1.3 MACCHINE ASTRATTE
- **CONTROLLO** (qual è il prossimo modulo/sottoprogramma che va in esecuzione?)
  - **2.1 CENTRALIZZATO**
    - **2.1.1 CALL/RETURN**
    - **2.1.2 MANAGER MODEL**
  - **2.2 EVENT DRIVEN**
    - **2.2.1 BROADCAST**
    - **2.2.2 INTERRUPT-DRIVEN**
- SCOMPOSIZIONE MODULARE
  - 3.1 OBJEC-ORIENTED
  - 3.2 FUNZIONALE (DATA FLOW, PIPELINE, STRUCTURAL)
- ARCHITETTURE SPECIFICHE DEL DOMINIO
  - 4.1 GENERICHE [Compiler, MVC]
  - 4.2 REFERENCE [ISO/OSI]

Modelli di Progettazione

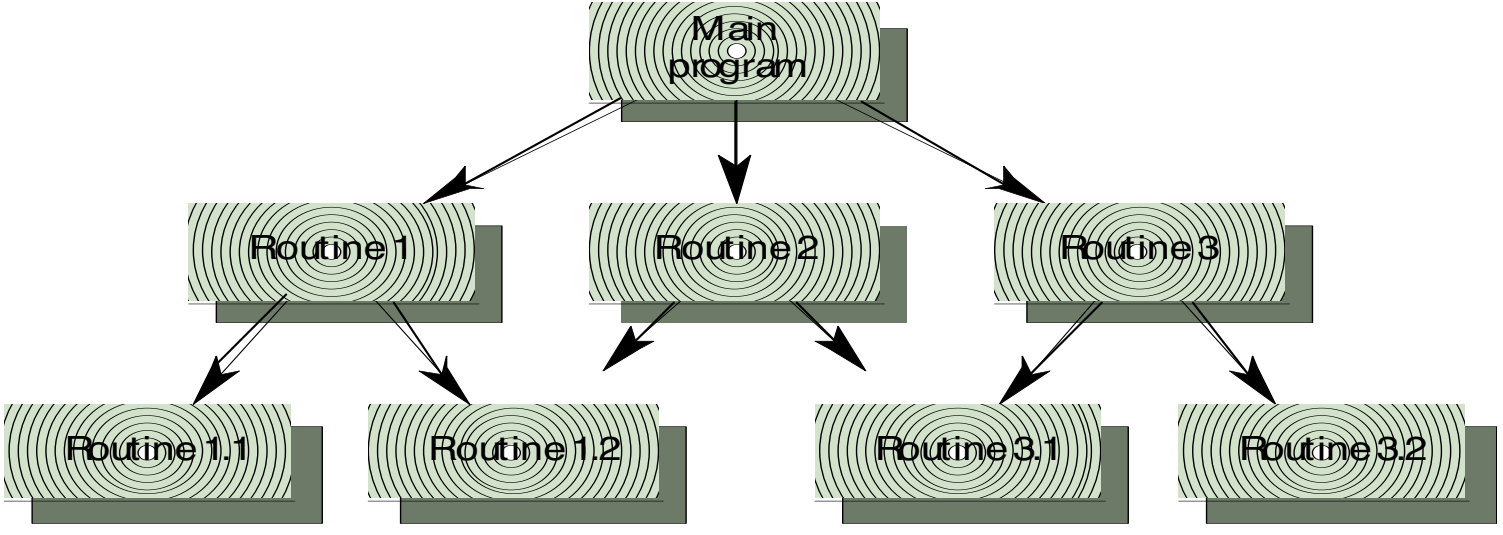
2. Control models (styles)

- Are concerned with the control flow between sub-systems. Distinct from the system decomposition model
- 2.1 Centralised control
  - CALL RETURN
  - MANAGER
  - One sub-system has overall responsibility for control and starts and stops other sub-systems
- 2.2 Event-based control
  - Each sub-system can respond to externally generated events from other sub-systems or the system's environment

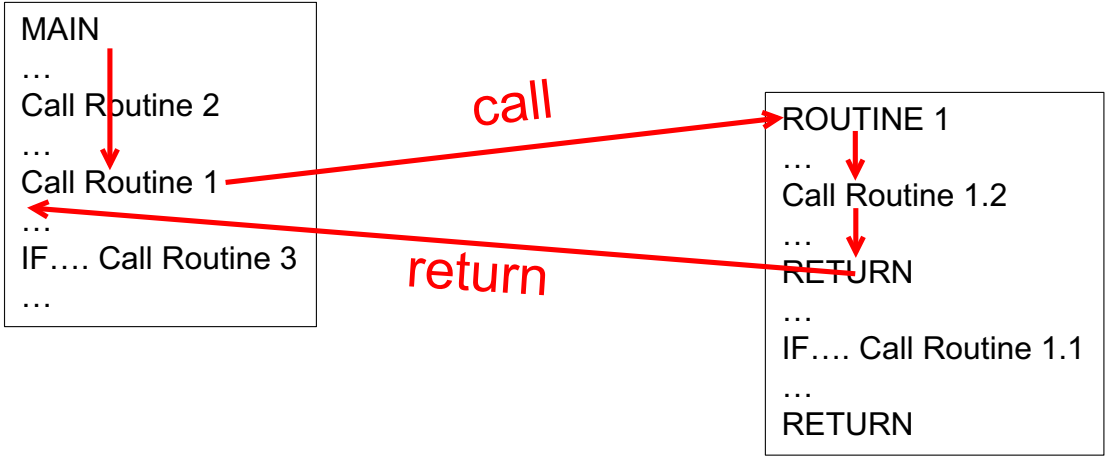
Centralised control

- A control sub-system takes responsibility for managing the execution of other sub-systems
- 2.1.1 Call-return model → programmaz. sequenziale dove quando richiamo un sottosistema, l'esecuz. principale termina finché il sottosistema non termina
  - Top-down subroutine model where control starts at the top of a subroutine hierarchy and moves downwards. Applicable to sequential systems
- 2.1.2 Manager model → programma centrale che attiva lui i sotto sistemi
  - Applicable to concurrent systems. One system component controls the stopping, starting and coordination of other system processes. Can be implemented in sequential systems as a case statement

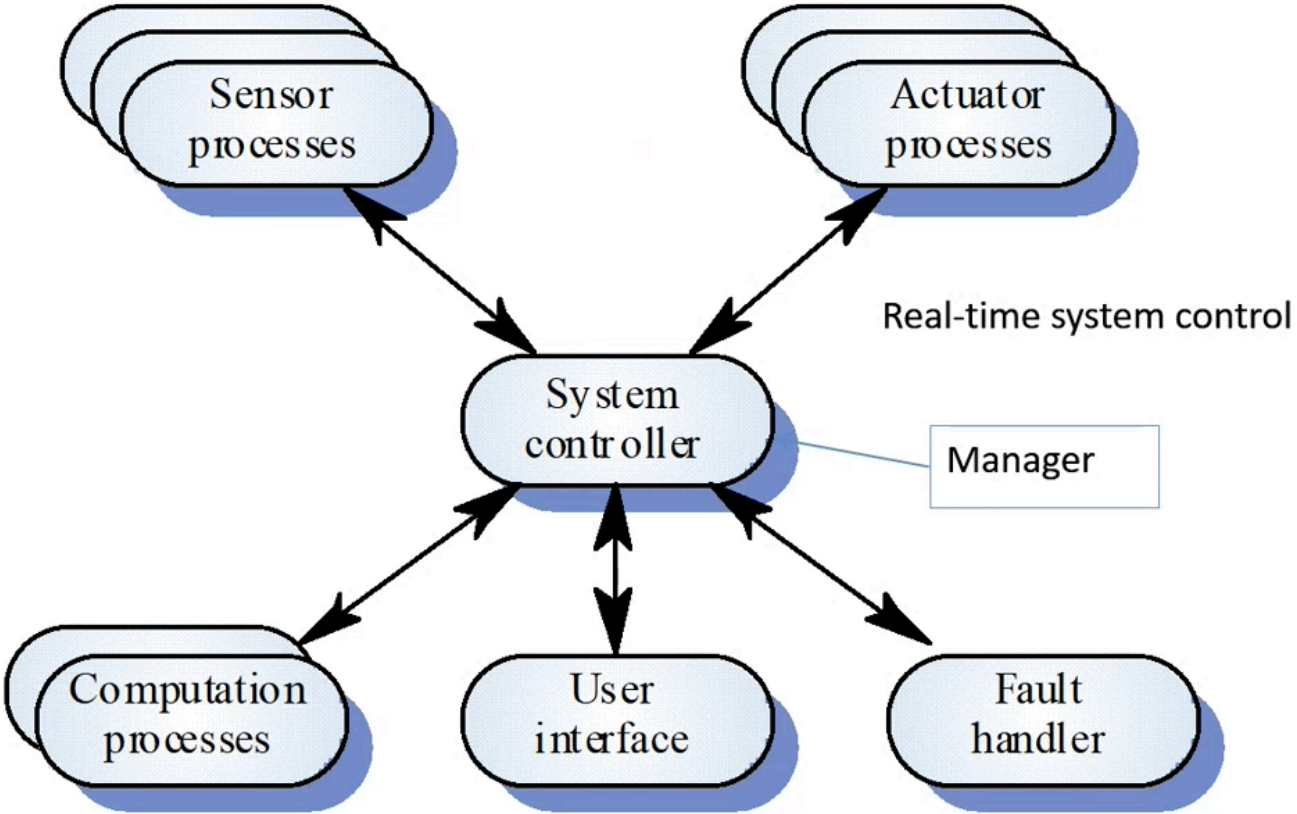
Call-return model (2.1.1)



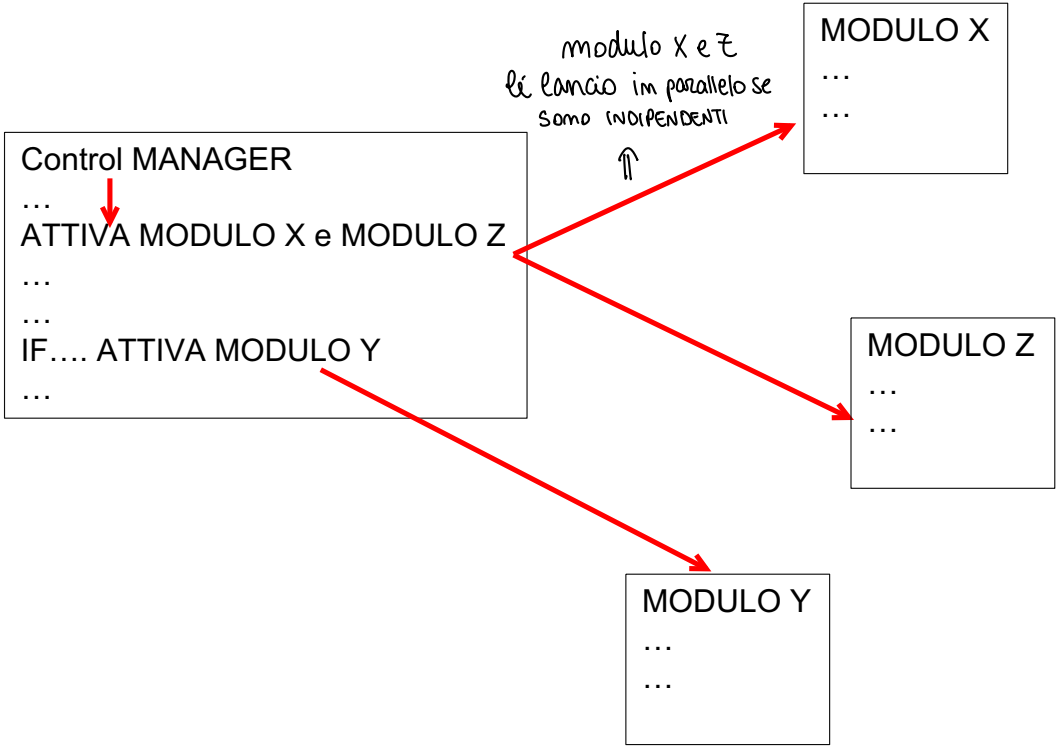
Call-return: sorgente e control flow



Manager Model(2.1.2)



Centralizzato: sorgente e control flow



2.2 Event-driven systems

→ modelli meno rigidi dei precedenti, i moduli/sottosistemi sono in attesa di una richiesta, non hanno un ordine sequenziale di come e quando agire. Si attiva on-demand

- Driven by externally generated events where the timing of the event is outwith the control of the sub-systems which process the event
- Two principal event-driven models
  - 2.2.1 Broadcast models. An event is broadcast to all sub-systems. Any sub-system which can handle the event may do so

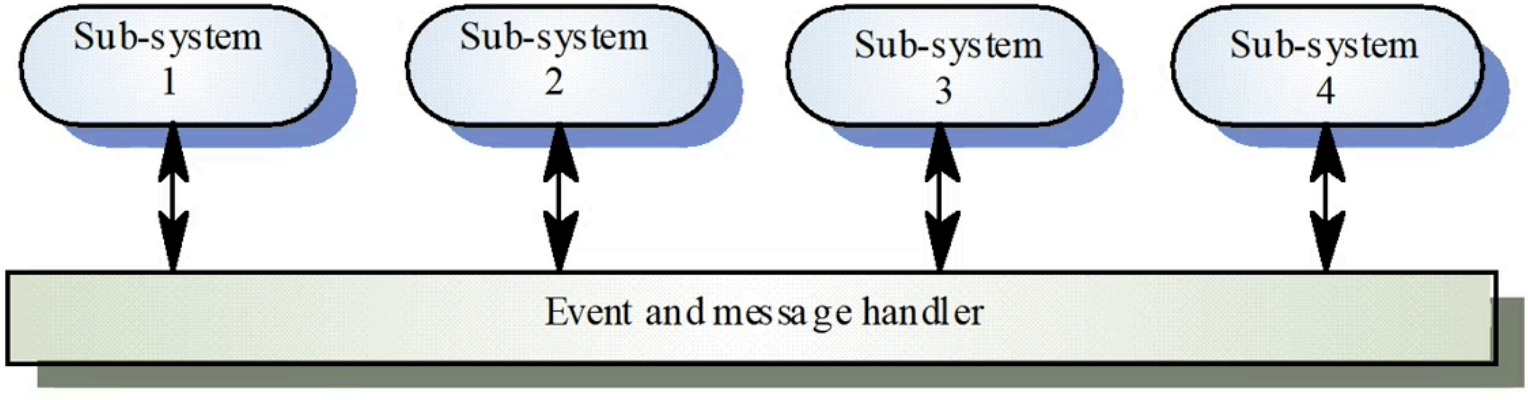
    → si scaterna un evento e si manda in broadcast una richiesta di controllo e il sottosistema che sa gestirlo prende il controllo, gli altri lo ignorano
  - 2.2.2 Interrupt-driven models. Used in real-time systems where interrupts are detected by an interrupt handler and passed to some other component for processing

    → si scaterna un evento e si va a vedere nel vettore delle interruz. a che sottosistema affidare il controllo → come un s.o.
- Other event driven models include spreadsheets, production systems, blackboard systems

Broadcast model

- Effective in integrating sub-systems on different computers in a network
- Sub-systems register an interest in specific events. When these occur, control is transferred to the sub-system which can handle the event
- Control policy is not embedded in the event and message handler. Sub-systems decide on events of interest to them
- However, sub-systems don't know if or when an event will be handled
- UTILIZZATO nei sistemi client-server

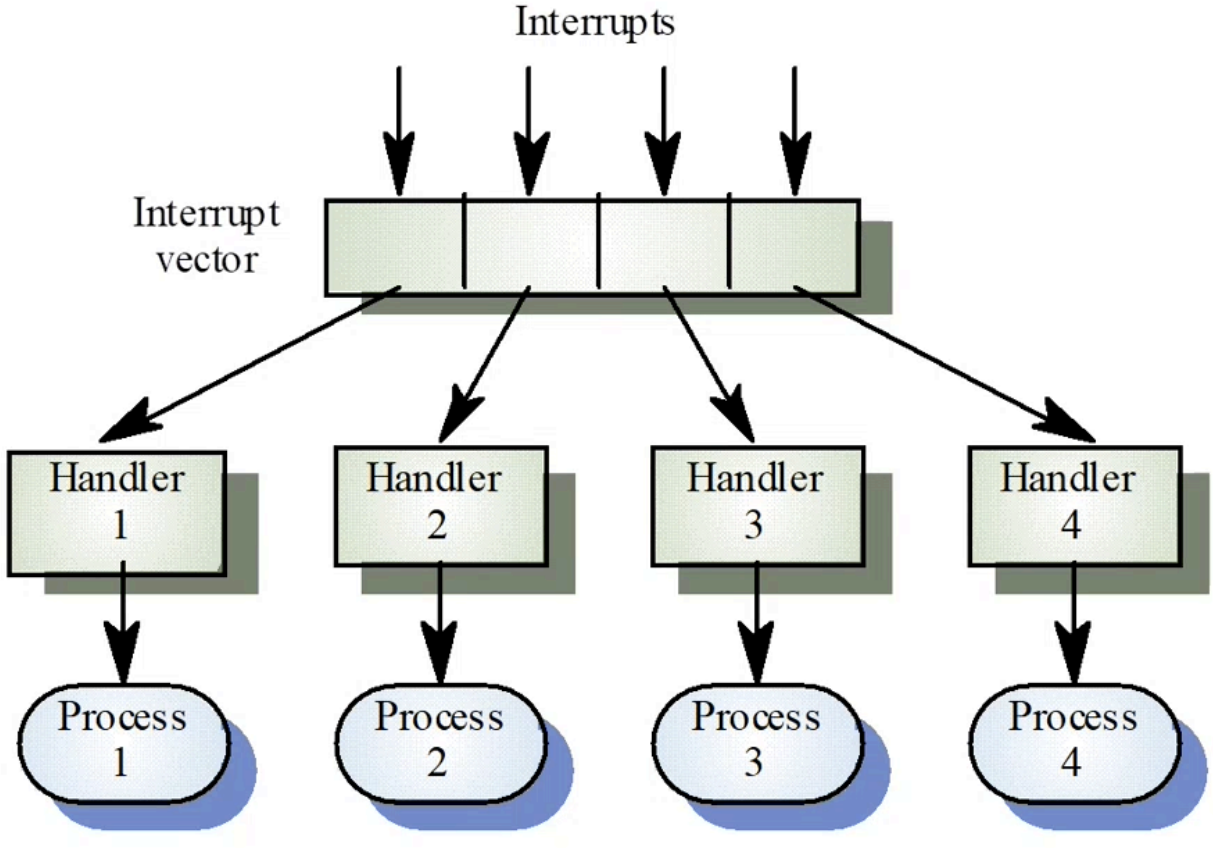
Selective broadcasting (2.2.1)



2.2.2 Interrupt-driven systems

- Used in real-time systems where fast response to an event is essential
- There are known interrupt types with a handler defined for each type
- Each type is associated (HARDWARE) with a memory location and a hardware switch causes transfer to its handler
- Allows fast response but complex to program and difficult to validate

Interrupt-driven control



- STRUTTURAZIONE DELL'ARCHITETTURA
  - 1.1 REPOSITORY
  - 1.2 CLIENT-SERVER
  - 1.3 MACCHINE ASTRATTE
- CONTROLLO
  - 2.1 CENTRALIZZATO
    - 2.1.1 CALL/RETURN
    - 2.1.2 MANAGER MODEL
  - 2.2 EVENT DRIVEN
    - 2.2.1 BROADCAST
    - 2.2.2 INTERRUPT-DRIVEN
- SCOMPOSIZIONE MODULARE
  - 3.1 OBJEC-ORIENTED
  - 3.2 FUNZIONALE (DATA FLOW, PIPELINE, STRUCTURAL)
- ARCHITETTURE SPECIFICHE DEL DOMINIO
  - 4.1 GENERICHE [MVC]
  - 4.2 REFERENCE [ISO/OSI]
- ARCHITETTURE APPLICATIVE GENERICHE
  - 1. TRANSACTION PROCESSING [information systems, e-commerce]
  - 2. LANGUAGE PROCESSING [compiler, interpreter, …]

Modelli di Progettazione

3. Modular decomposition styles

- Styles of decomposing sub-systems into modules.
- No rigid distinction between system organisation and modular decomposition.

Sub-systems and modules

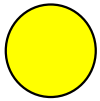
- A sub-system is a system in its own right whose operation is independent of the services provided by other sub-systems.
- A module is a system component that provides services to other components but would not normally be considered as a separate system
- Terminologia altrettanto usata al posto di modulo: ‘componenete’

Modular decomposition

- Another structural level where sub-systems are decomposed into modules
- Two modular decomposition models covered
  - (3.1) An object model where the system is decomposed into interacting objects
  - (3.2) 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

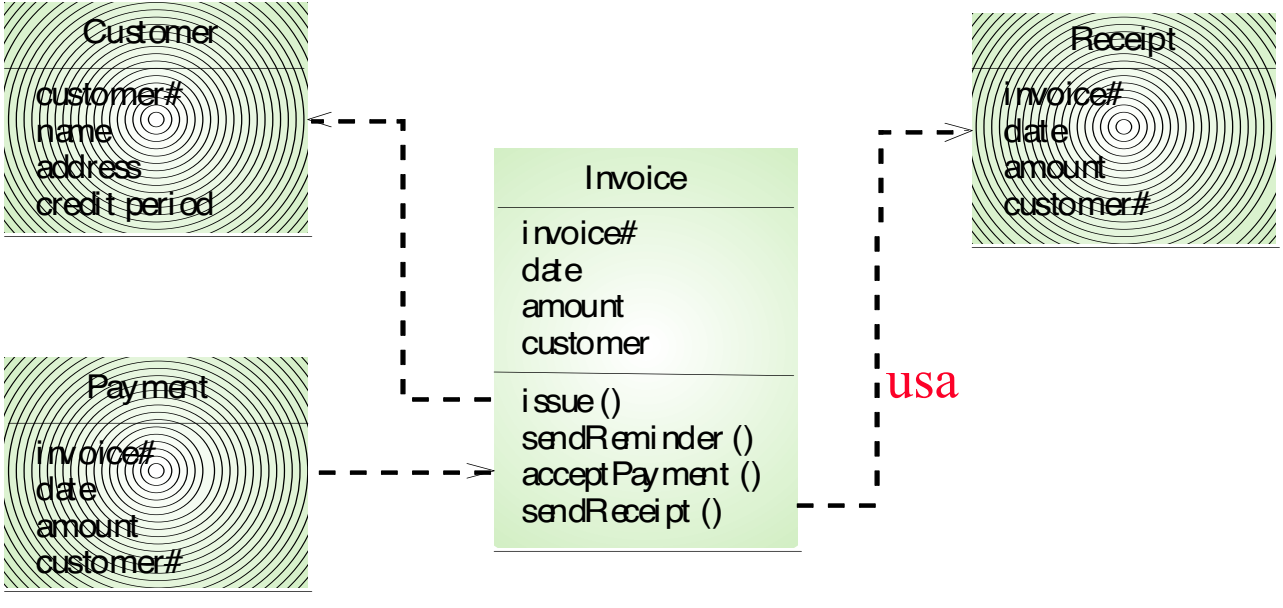
3.1 Object models

- 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



Vedi Lezioni Dr. Baruzzo

Invoice processing system

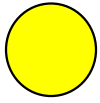


Vedi Lezioni Dr. Baruzzo

Object model advantages

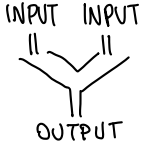
(ex 7th ed)

- Objects are loosely coupled so their implementation can be modified without affecting other objects.
- The objects may reflect real-world entities.
- OO implementation languages are widely used.
- However, object interface changes may cause problems and complex entities may be hard to represent as objects.



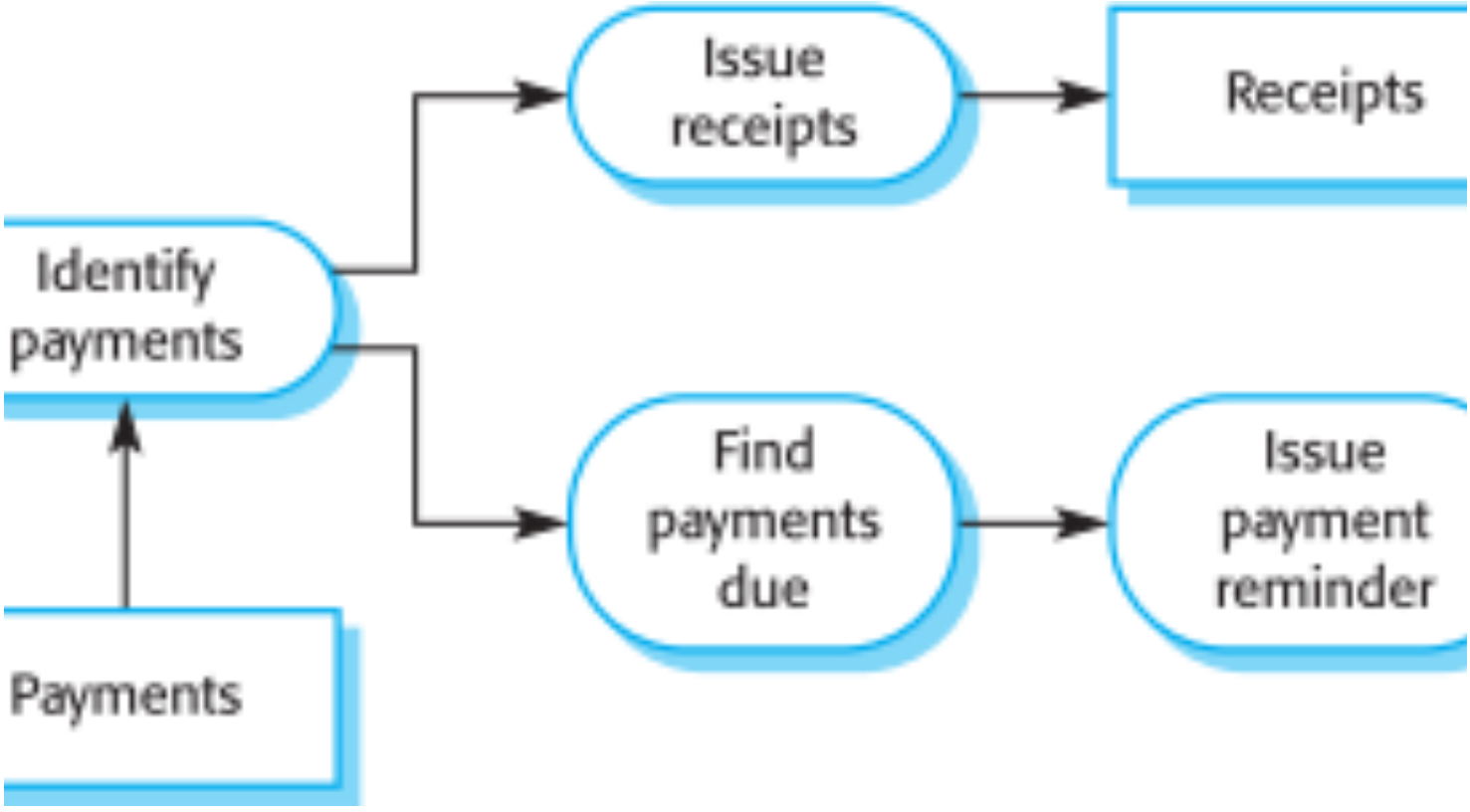
Vedi Lezioni Dr. Baruzzo

3.2 Data-flow models (Function-oriented pipelining/Pipe and filter architecture)



- Functional transformations process their inputs to produce outputs
- May be referred to as a **pipe** and **filter** model (as in LINUX/UNIX shell)
- Variants of this approach are very common. When transformations are sequential, this is a batch sequential model which is extensively used in data processing systems
- Not really suitable for interactive systems

An example of the pipe and filter architecture used in a payments system



Pipeline model advantages

(ex 7th ed)

- Supports transformation reuse (riuso delle processi/bolle di trasformazione/elaborazione).
- Intuitive organisation for stakeholder communication.
- Easy to add new transformations.
- Relatively simple to implement as either a concurrent or sequential system.
- However, requires a common format for data transfer along the pipeline and difficult to support event-based interaction.

The pipe and filter pattern

Name	Pipe and filter
Description	The processing of the data in a system is organized so that each processing component (filter) is discrete and carries out one type of data transformation. The data flows (as in a pipe) from one component to another for processing.
Example	Previous slide shows an example of a pipe and filter system used for processing invoices.
When used	Commonly used in data processing applications (both batch- and transaction-based – APPLICAZIONI GESTIONALI) where inputs are processed in separate stages to generate related outputs.
Advantages	Easy to understand and supports transformation reuse. Workflow style matches the structure of many business processes. Evolution by adding transformations is straightforward. Can be implemented as either a sequential or concurrent system.
Disadvantages	The format for data transfer has to be agreed upon between communicating transformations. Each transformation must parse its input and unparse its output to the agreed form. This increases system overhead and may mean that it is impossible to reuse functional transformations that use incompatible data structures.

- STRUTTURAZIONE DELL'ARCHITETTURA
  - 1.1 REPOSITORY
  - 1.2 CLIENT-SERVER
  - 1.3 MACCHINE ASTRATTE
- CONTROLLO
  - 2.1 CENTRALIZZATO
    - 2.1.1 CALL/RETURN
    - 2.1.2 MANAGER MODEL
  - 2.2 EVENT DRIVEN
    - 2.2.1 BROADCAST
    - 2.2.2 INTERRUPT-DRIVEN
- SCOMPOSIZIONE MODULARE
  - 3.1 OBJEC-ORIENTED
  - 3.2 FUNZIONALE (DATA FLOW, PIPELINE, STRUCTURAL)
- ARCHITETTURE SPECIFICHE DEL DOMINIO
  - 4.1 GENERICHE [MVC]
  - 4.2 REFERENCE [ISO/OSI]
- ARCHITETTURE APPLICATIVE GENERICHE
  - 1. TRANSACTION PROCESSING [information systems, e-commerce]
  - 2. LANGUAGE PROCESSING [compiler, interpreter, …]

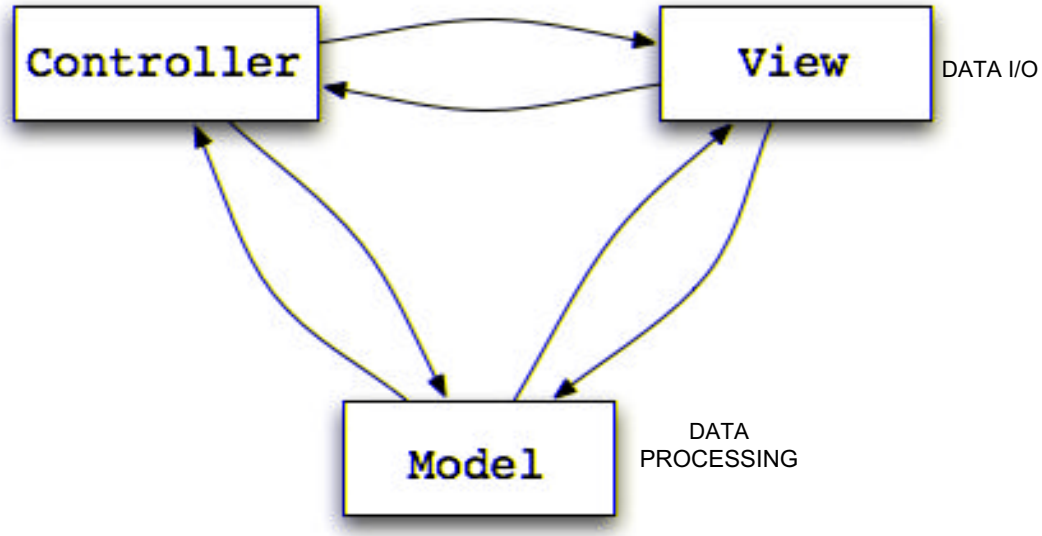
Modelli di Progettazione

4. Domain-specific architectures

- Architectural models which are specific to some application domain
- Two types of domain-specific models:
  - Generic models which are abstractions from a number of real systems and which encapsulate the principal characteristics of these systems
  - Reference models which are more abstract, idealised model. Provide a means of information about that class of system and of comparing different architectures
- Generic models are usually bottom-up models; Reference models are top-down models

Esempio di Generic Model: Architettura Model-View-Controller (MVC) for WebApplications

↳ per sistemi interattivi (interaz. con utente)



- Architectural Pattern from Smalltalk (1979)
- Decouples data/data-processing and presentation
- Eases the development

The Model-View-Controller (MVC) architectural pattern

Name	MVC (Model-View-Controller)
Description	Separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other. The Model component manages the system data and associated operations on that data. The View component defines and manages how the data is presented to the user. The Controller component manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to the View and the Model. See Figure 6.3.
Example	Figure 6.4 shows the architecture of a web-based application system organized using the MVC pattern.
When used	Used when there are multiple ways to view and interact with data. Also used when the future requirements for interaction and presentation of data are unknown. / esigenza di separazione dati da loro visualizzazione
Advantages	Allows the data to change independently of its representation and vice versa. Supports presentation of the same data in different ways with changes made in one representation shown in all of them.
Disadvantages	Can involve additional code and code complexity when the data model and interactions are simple.

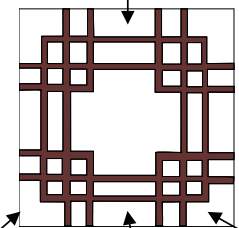
MVC

model: modello del dominio, business logic, data management, ..

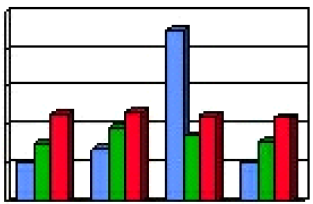
controller: comunica con l'utente, decide e controlla le altre 2 componenti

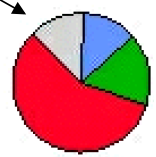
view: visualizza il modello e riceve input dall'utente

float sales[3][4];

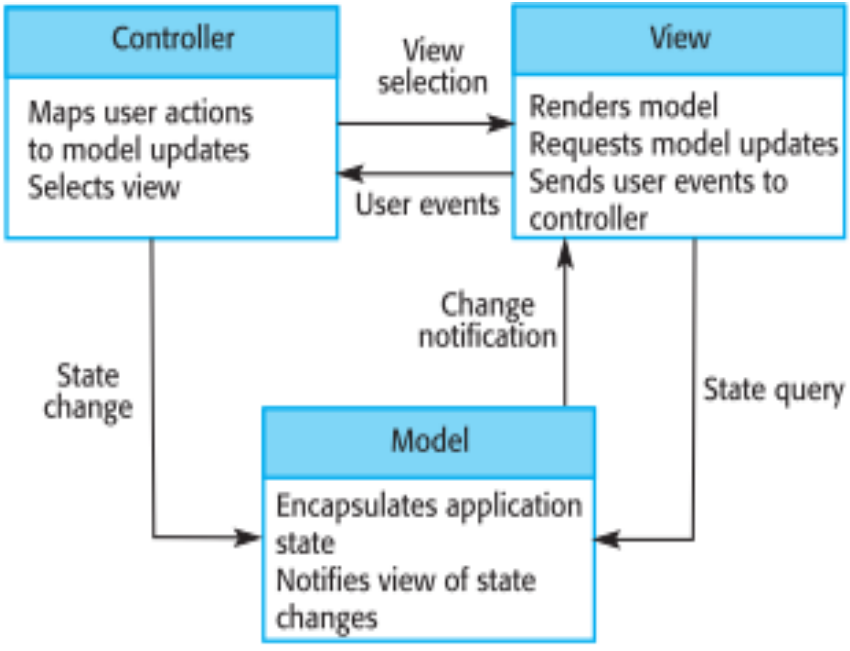


	1st Qtr	2nd Qtr	3rd Qtr	4th Qtr
East	20.4	27.4	90	20.4
West	30.6	38.6	34.6	31.6
North	45.9	46.9	45	43.9

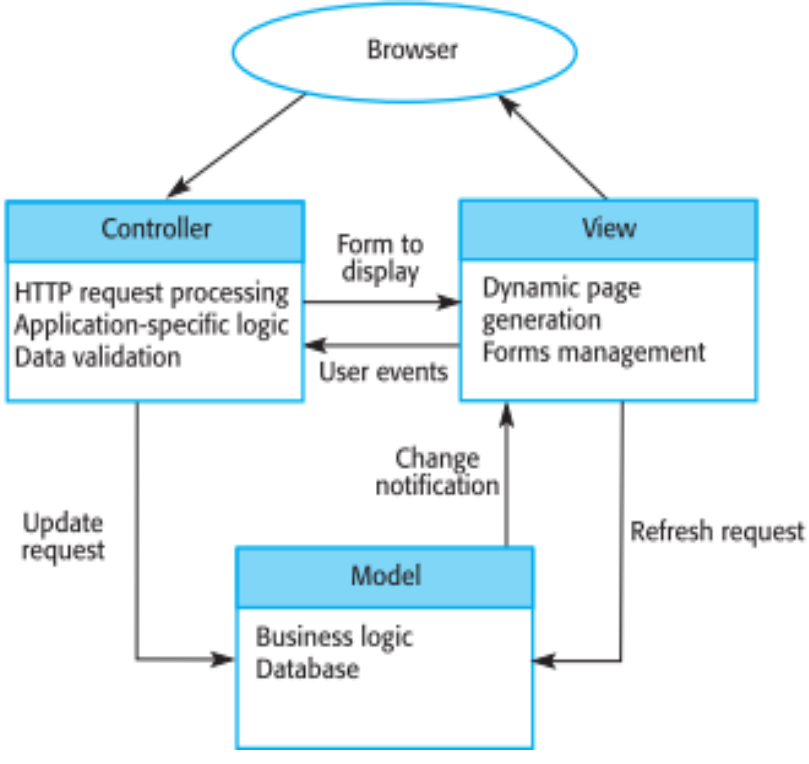




The organization of the Model-View-Controller



Web application architecture using the MVC pattern



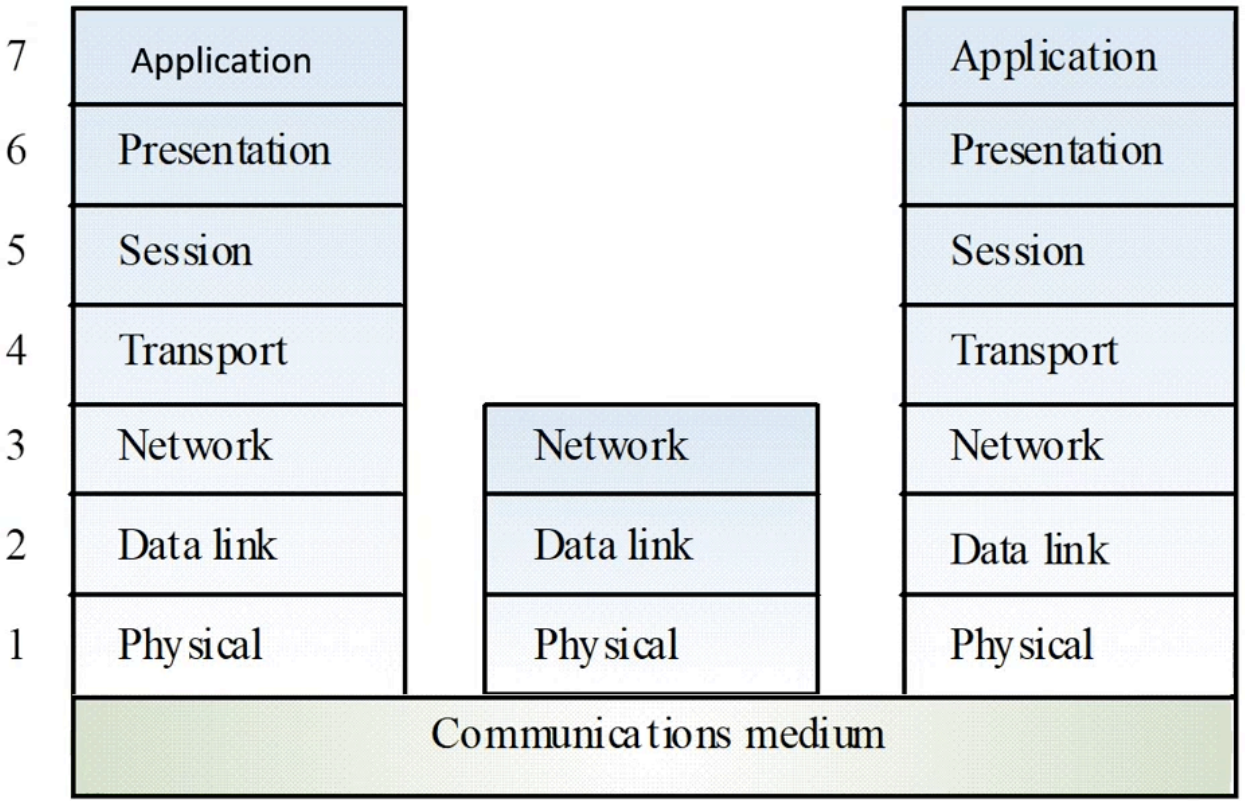
Example Control Flow in MVC

1. User interacts with the VIEW UI
2. CONTROLLER handles the user input
3. CONTROLLER updates/calls the MODEL
4. VIEW uses MODEL to generate new UI
5. UI waits for user interaction
6. …

Reference architectures

- Reference models are derived from a study of the application domain rather than from existing systems
- May be used as a basis for system implementation or to compare different systems. It acts as a standard against which systems can be evaluated
- Esempio: OSI model is a layered model for communication systems

OSI reference model



- STRUTTURAZIONE DELL'ARCHITETTURA
  - 1.1 REPOSITORY
  - 1.2 CLIENT-SERVER
  - 1.3 MACCHINE ASTRATTE
- CONTROLLO
  - 2.1 CENTRALIZZATO
    - 2.1.1 CALL/RETURN
    - 2.1.2 MANAGER MODEL
  - 2.2 EVENT DRIVEN
    - 3.1 BROADCAST
    - 3.2 INTERRUPT-DRIVEN
- SCOMPOSIZIONE MODULARE
  - 3.1 OBJEC-ORIENTED
  - 3.2 FUNZIONALE (DATA FLOW, PIPELINE)
- ARCHITETTURE SPECIFICHE DEL DOMINIO
  - 4.1 GENERICHE [Compiler, MVC]
  - 4.2 REFERENCE [ISO/OSI]
- **ARCHITETTURE APPLICATIVE GENERICHE**
  - **1. TRANSACTION PROCESSING [information systems, e-commerce]**
  - **2. LANGUAGE PROCESSING [compiler, interpreter, …]**

Modelli di Progettazione

Application architectures

Obiettivo

- Illustrare altri modelli architetturali che sono stati progettati per le tipologie più comuni di applicazioni.

Application architectures

- Application systems are designed to meet an organisational need.
- As businesses have much in common, their application systems also tend to have a common architecture that reflects the application requirements.
- A generic application architecture is an architecture for a type of software system that may be configured and adapted to create a system that meets specific requirements.

Many uses of application architectures

- Similmente a quanto si può dire per le ‘reference architecture’ -

- As a starting point for architectural design.
- As a design checklist.
- As a way of organising the work of the development team.
- As a means of assessing components for reuse.
- As a vocabulary for talking about application types.

Examples of application types

Sistema offre un servizio e l'utente accede con i propri dati. Il servizio si conclude con l'aggiornamento del db (TRANSAZIONE)

1. Transaction processing applications
   - Data-centred applications that process user requests and update information in a system database.
2. Language processing systems
   - Applications where the users' intentions are specified in a formal language that is processed and interpreted by the system.

Application type examples

- Two very widely used generic application architectures are transaction processing systems and language processing systems.

1. Transaction processing systems

   E-commerce systems;

   Reservation systems.

2. Language processing systems

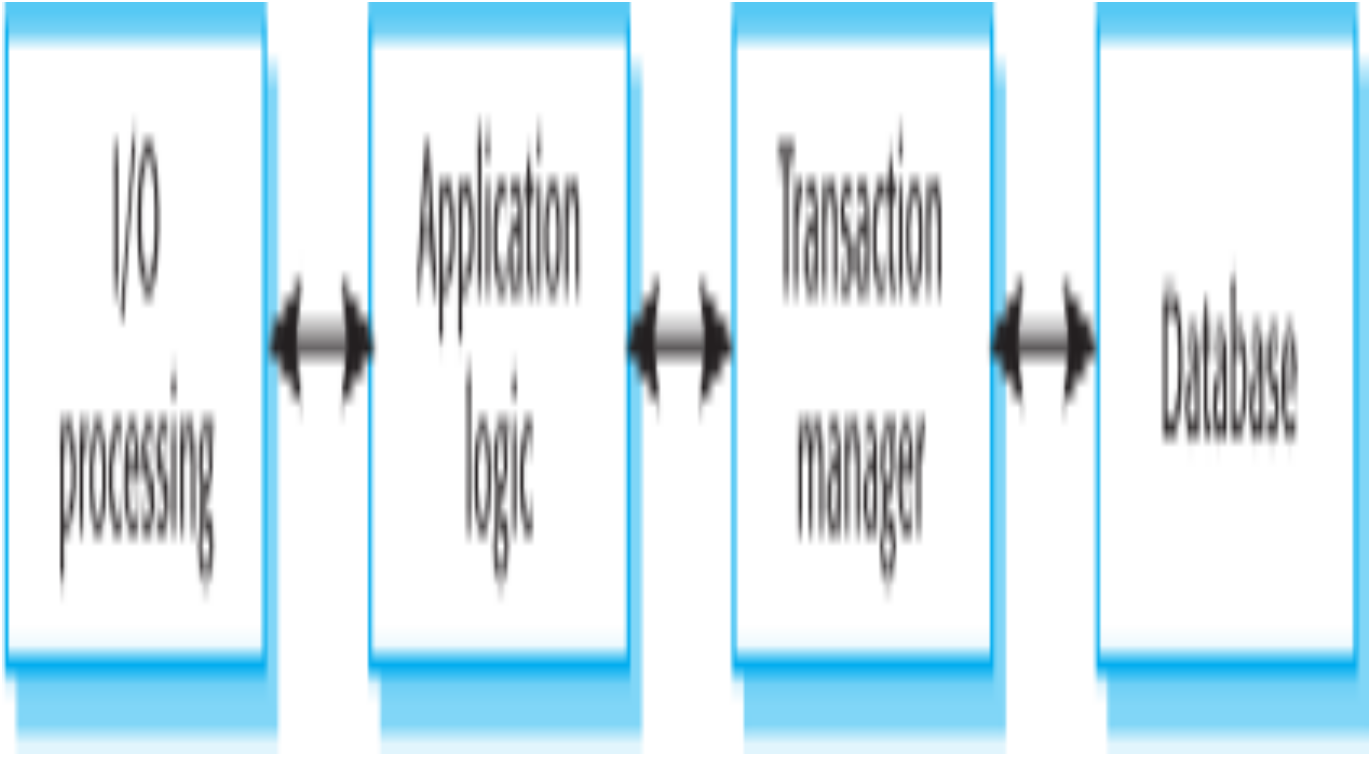
   Compilers;

   Command interpreters.

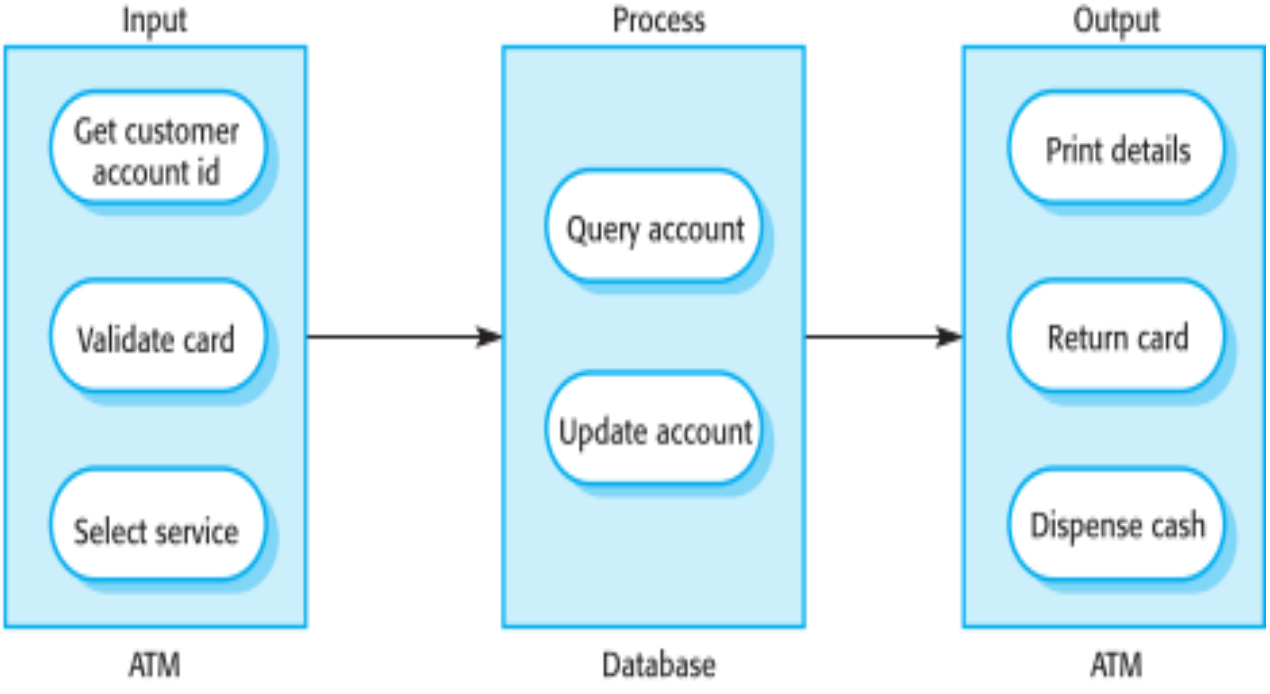
1. Transaction processing systems

- Process user requests for information from a database or requests to update the database.
- From a user perspective a transaction is:
  - Any coherent sequence of operations that satisfies a goal;
  - For example - find the times of flights from London to Paris.
- Users make asynchronous requests for service which are then processed by a transaction manager
- E' conforme ad un'organizzazione generale a pipe line

The structure of transaction processing applications



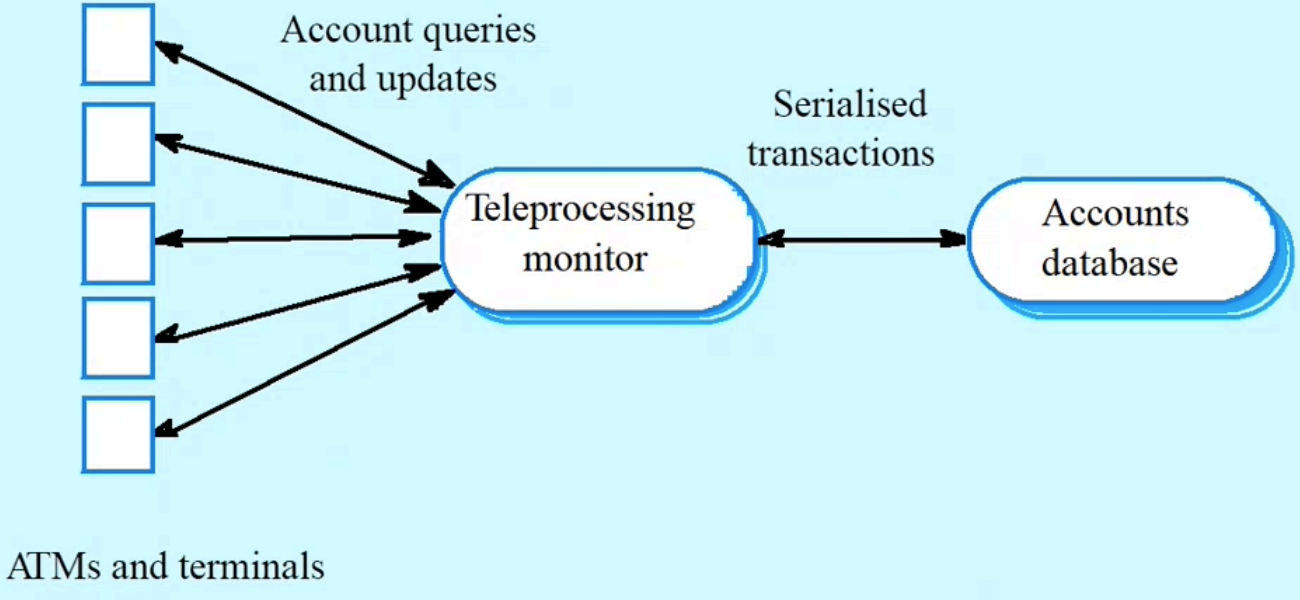
The software architecture of an ATM system



Transaction processing middleware

- Dato che al Sistema possono arrivare contemporaneamente più richieste di effettuare una transazione, è necessario coordinare l'accesso ai DB
- Transaction management **middleware** or teleprocessing monitors (**TP monitor**, es. CICS di IBM) handle communications with different terminal types (e.g. ATMs and counter terminals), **seria**lises data and sends it for processing.
- Query processing takes place in the system database and results are sent back through the transaction manager to the user's terminal.

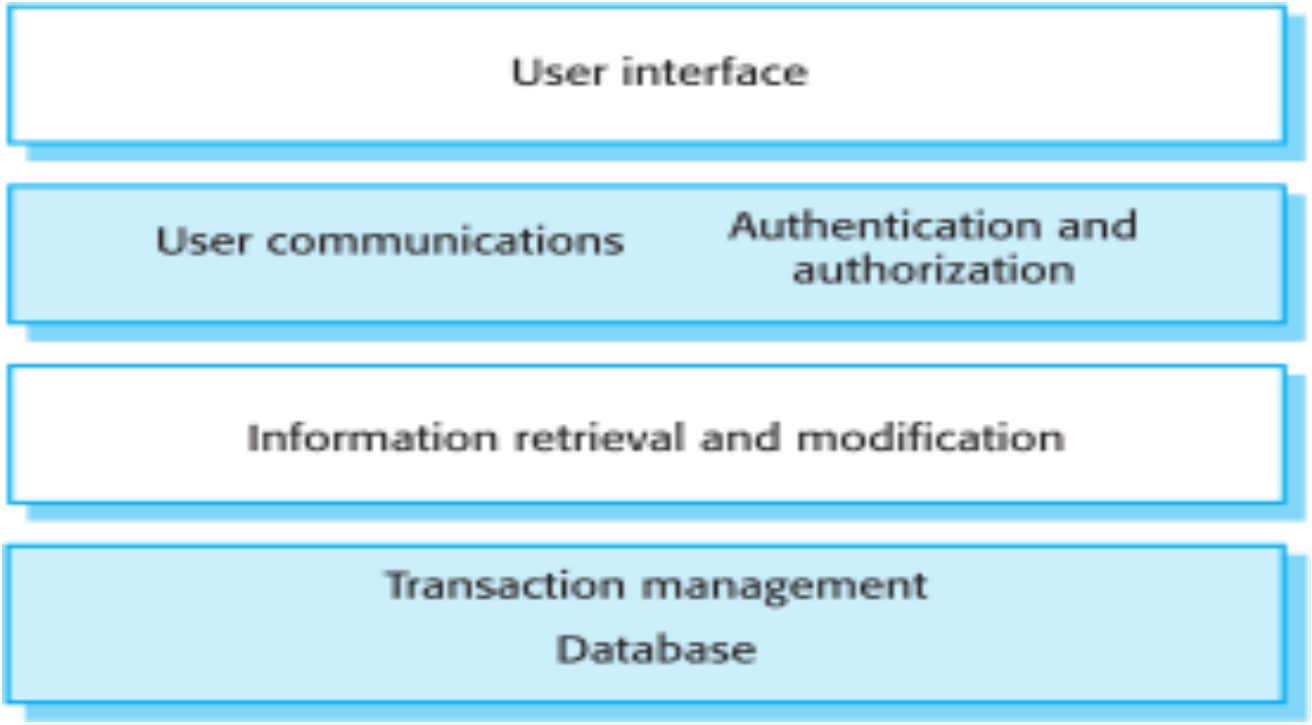
Transaction management



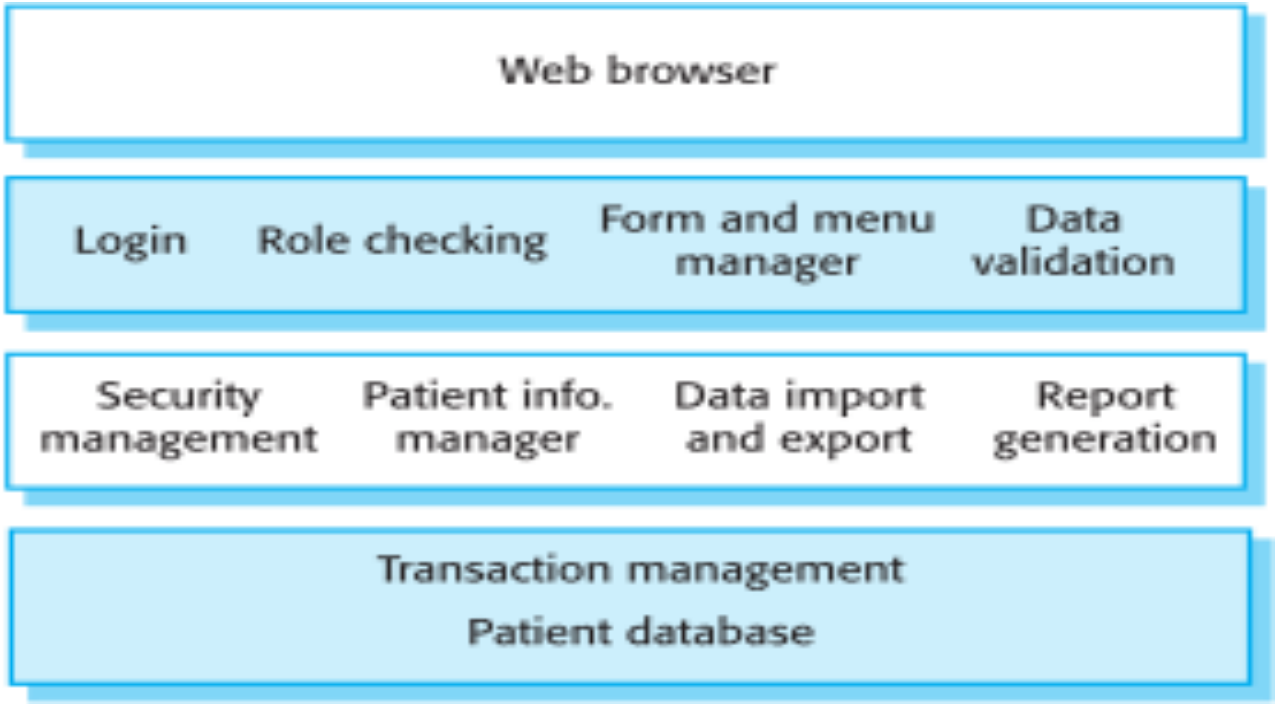
Management Information systems architecture (Sistemi Informativi)

- Information systems have a generic architecture that can be organised as a layered architecture.
- These are transaction-based systems as interaction with these systems generally involves database transactions.
- Layers include:
  - The user interface
  - User communications
  - Information retrieval
  - System database

Layered information system architecture



The architecture of the Mentcare system

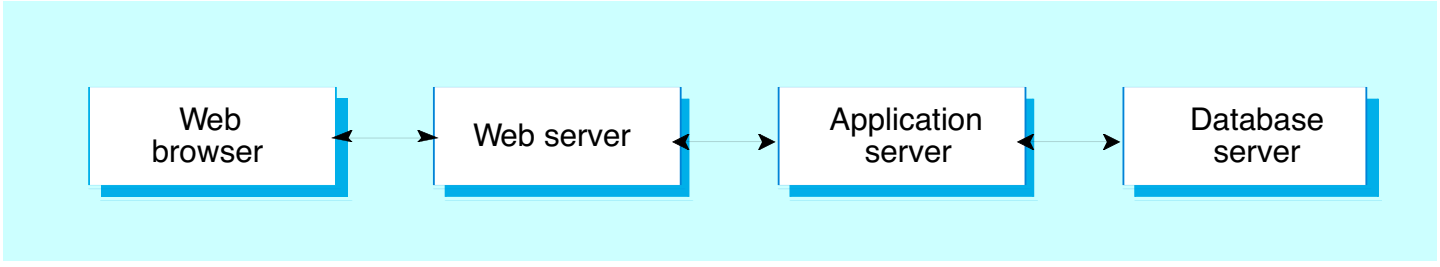


Web-based information systems

- Information and resource management systems are now usually web-based systems where the user interfaces are implemented using a web browser.
- For example, e-commerce systems are Internet-based resource management systems that accept electronic orders for goods or services and then arrange delivery of these goods or services to the customer.
- In an e-commerce system, the application-specific layer includes additional functionality supporting a ‘shopping cart’ in which users can place a number of items in separate transactions, then pay for them all together in a single transaction.

E-commerce system architecture

- E-commerce systems are Internet-based resource management systems that accept electronic orders for goods or services.
- They are usually organised using a multi-tier architecture with application layers associated with each tier.

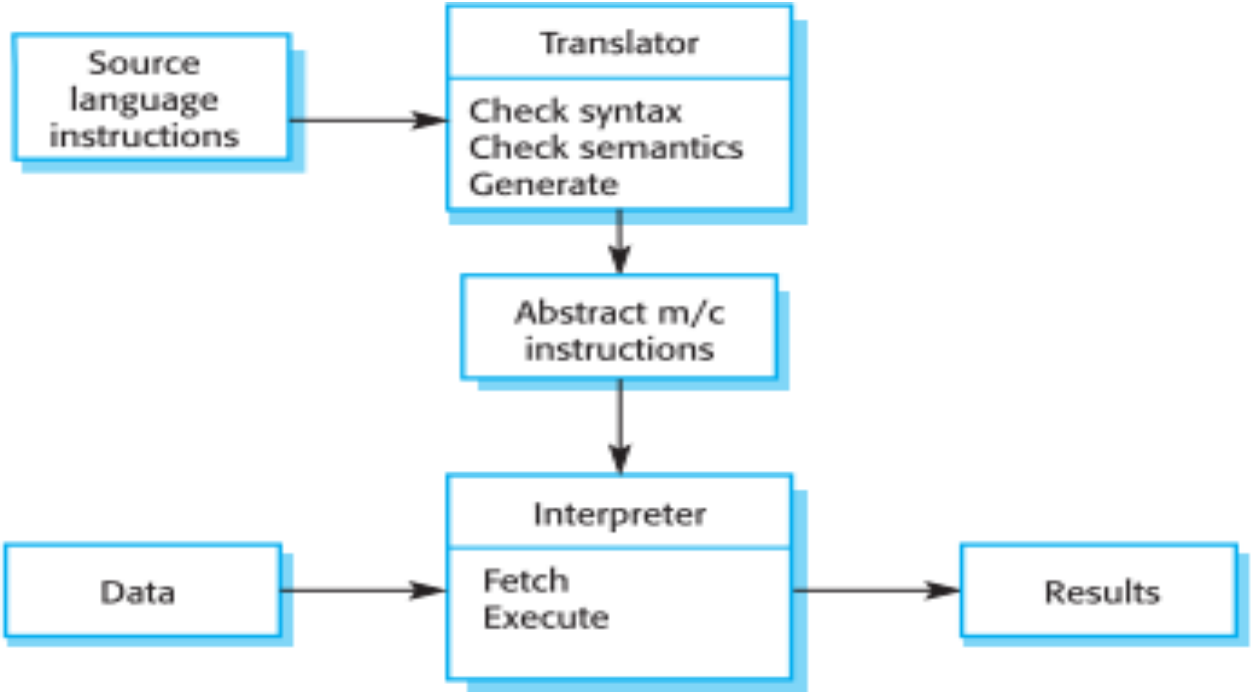


2. Language processing systems

- Accept a natural or artificial language as input and generate some other representation of that language. (es. compilatore)
- May include an interpreter to act on the instructions in the language that is being processed.

- Qualsiasi software che prevede delle funzionalità da attivare con specifici comandi può essere considerate di questo tipo, costituendo i vai comandi un vero e proprio linguaggio da interpretare.

The architecture of a language processing system



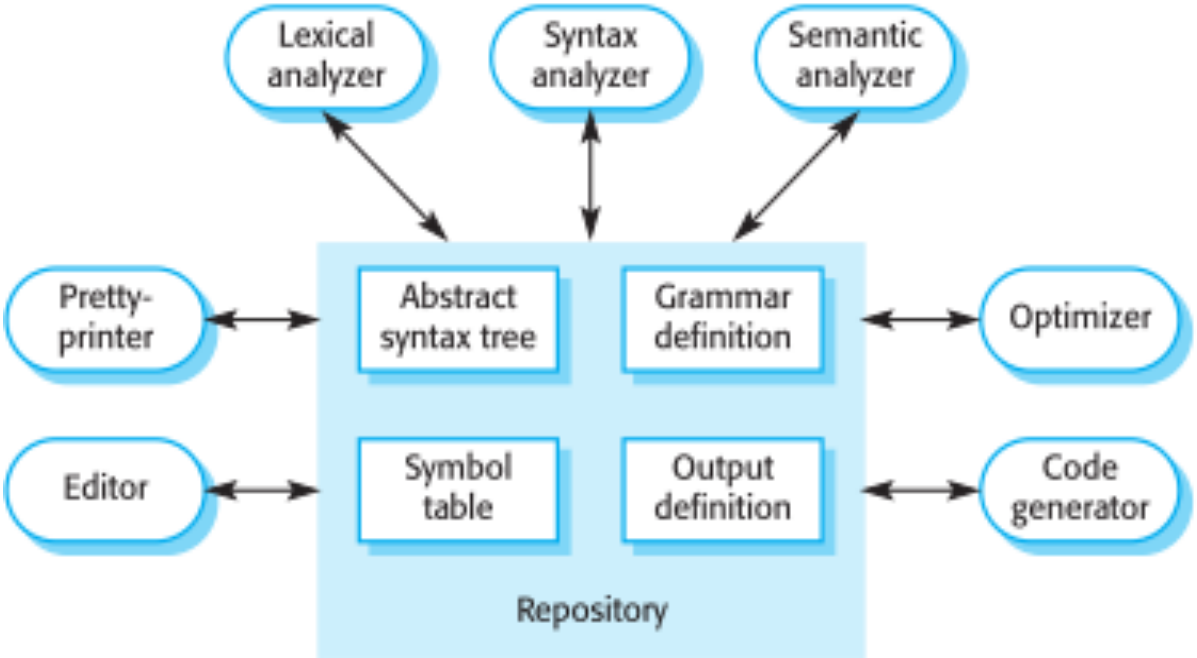
Compiler components

- A lexical analyzer, which takes input language tokens and converts them to an internal form.
- A symbol table, which holds information about the names of entities (variables, class names, object names, etc.) used in the text that is being translated.
- A syntax analyzer, which checks the syntax of the language being translated.
- A syntax tree, which is an internal structure representing the program being compiled.

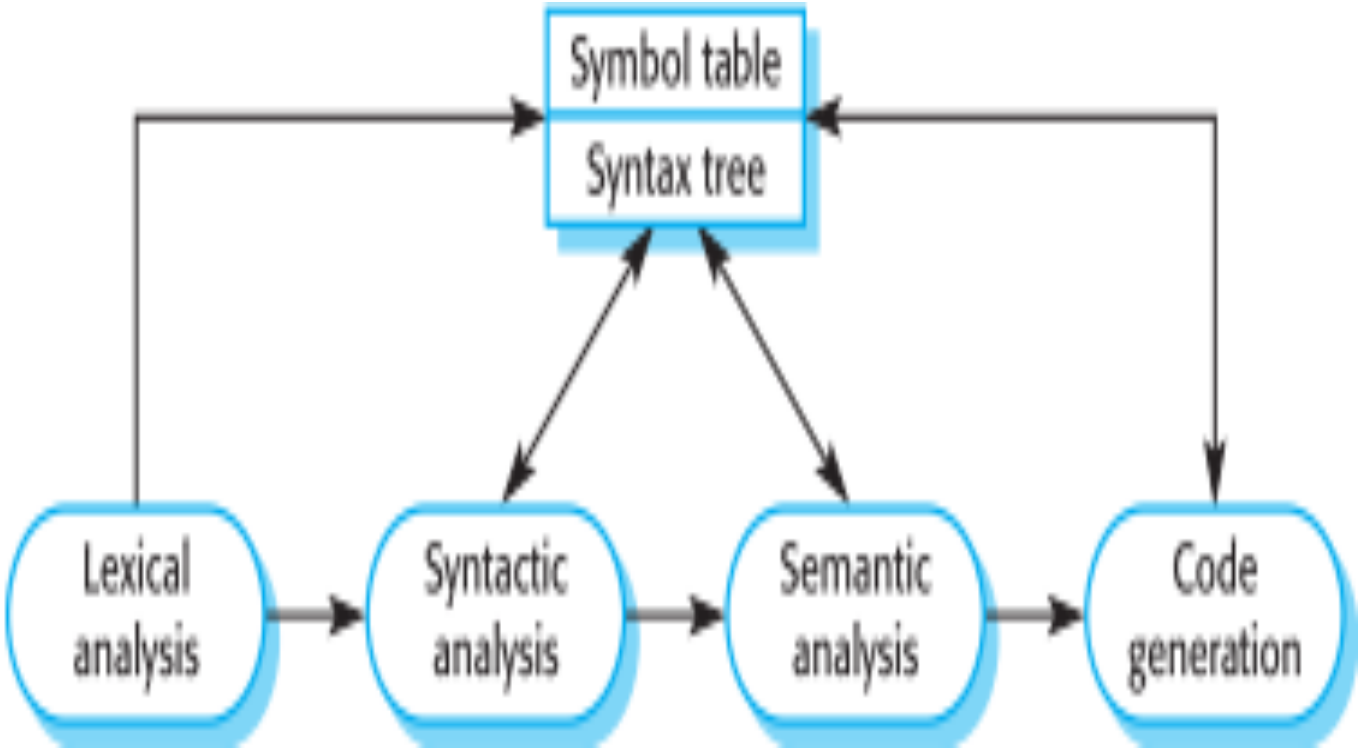
Compiler components

- A semantic analyzer that uses information from the syntax tree and the symbol table to check the semantic correctness of the input language text.
- A code generator that ‘walks’ the syntax tree and generates abstract machine code.

A repository architecture for a language processing system



A pipe and filter compiler architecture



- STRUTTURAZIONE DELL'ARCHITETTURA
  - 1.1 REPOSITORY
  - 1.2 CLIENT-SERVER
  - 1.3 MACCHINE ASTRATTE
- CONTROLLO
  - 2.1 CENTRALIZZATO
    - 2.1.1 CALL/RETURN
    - 2.1.2 MANAGER MODEL
  - 2.2 EVENT DRIVEN
    - 3.1 BROADCAST
    - 3.2 INTERRUPT-DRIVEN
- DECOMPOSIZIONE MODULARE
  - 3.1 OBJEC-ORIENTED
  - 3.2 FUNZIONALE (DATA FLOW, PIPELINE)
- ARCHITETTURE SPECIFICHE DEL DOMINIO
  - 4.1 GENERICHE (MVC]
  - 4.2 REFERENCE [ISO/OSI]
- **ARCHITETTURE APPLICATIVE GENERICHE**
  - **ARCHITETTURE APPLICATIVE GENERICHE**
  - **1. TRANSACTION PROCESSING [information systems, e-commerce]**
  - **2. LANGUAGE PROCESSING [compiler, interpreter, …]**

Architectural e design patterns

Esistono molti pattern architetturali, tipicamente relativi a tipiche soluzioni architetturali a problem/situazioni che si presentano spesso.

A tali pattern architetturali corrispondono spesso dei **DESIGN Pattern**, che dettagliano la rappresentazione del pattern architetturale fino al livello del codice, tipicamente si aspetti più localizzati/ristretti.

Examplee of very Common Architectural Patterns

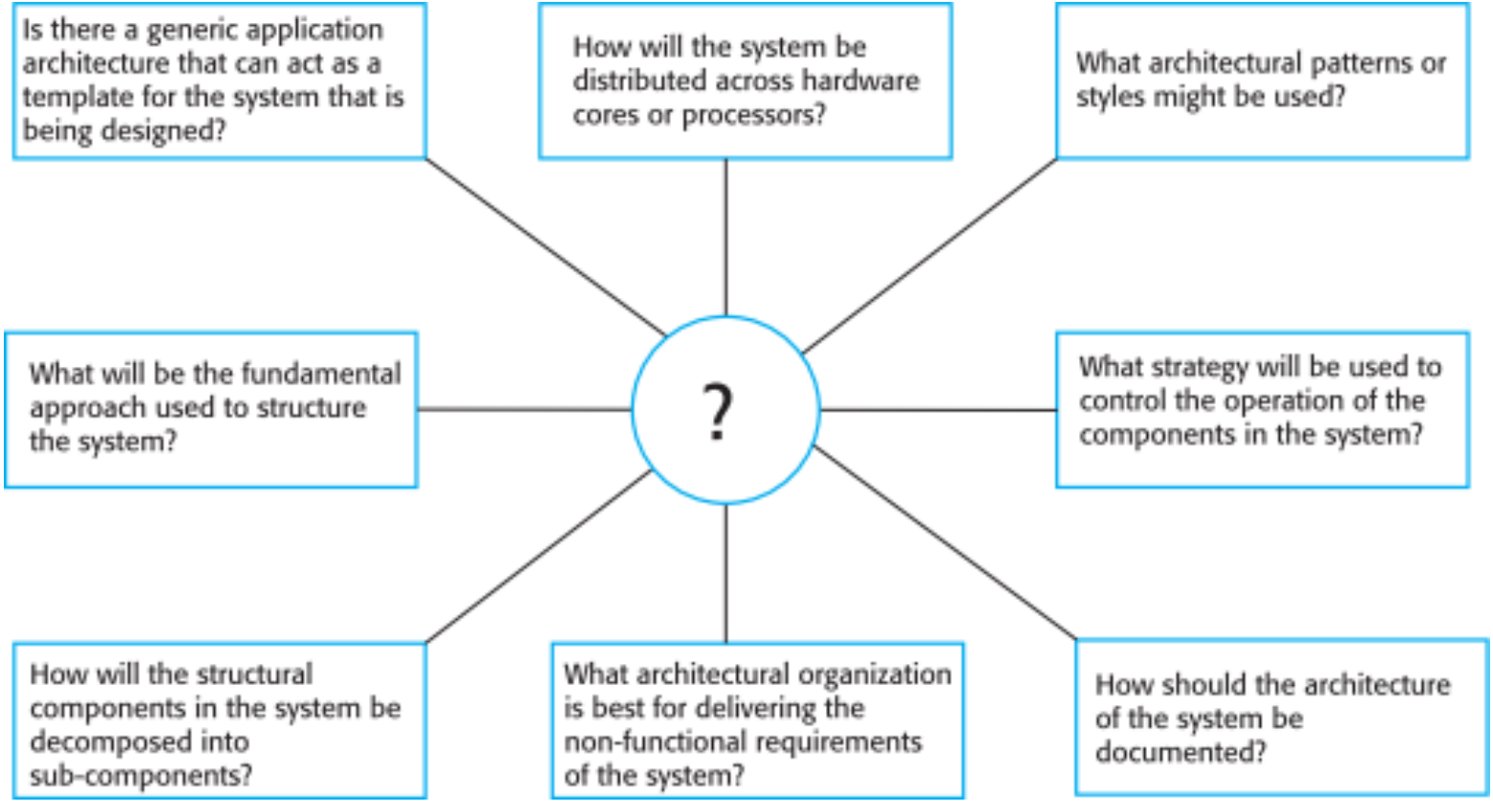
- Layered pattern
- Client-server pattern
- Master-slave pattern
- Pipe-filter pattern
- Broker pattern
- Peer-to-peer pattern
- Event-bus pattern
- Model-view-controller pattern
- Blackboard pattern
- Interpreter pattern
- Three-tier
- Microkernel
- …

Fragment of Client-Server Design pattern code

→ DESIGN PATTERN per un CLIENT-SERVER (pezzi di codice da riutilizzare)

```
public class Server {
private int port;
private ServerSocket server;

public Server(int port) throws Exception{
    this.port = port;
    try{
          server = new ServerSocket(port);
     }catch(Exception e){
          System.out.println("Server kann nicht gestartet werden: "+e.getMessage());
          throw e;
     }
}

public Server(int port){
    this.port = port;
}

public void start(){
    Socket client;
    Thread thread;

    while (true) {
        try {
            client = server.accept();
            // Verbindung eingegangen, Objekt erzeugen und in Thread laufen lassen
            System.out.println("Verbindung von "+client.getInetAddress());
            CommandHandler handler = new CommandHandlerImpl(client,new
```

Recap: Architectural design decisions



Set 9 - Cosa ricordare: concetti, motivazioni, conseguenze, relazioni fra concetti, ecc.

- Scenario generale del processo di progettazione e attività tipiche della progettazione. Definizione di progettazione architetturale (P.A.) e architettura sw, architetti sw. Importanza del procedere al P.A. Processi principali: strutturazione, controllo, decomposizione modulare. Rappresentazione della strutturazione del sistema. Impatto delle scelte architetturali, alcuni parametri importanti e concetti tipici: re-uso, stili di P.A., modelli architetturali, pattern architetturali.
- Vari modelli di P.A. per vari aspetti del processo di design: vedi elenco della seconda slide con tutti i Modelli di Progettazione considerati nel corso. Per ogni modello: caratteristiche, applicabilità, vantaggi, svantaggi.
- Architetture specifiche di un dominio. Architetture applicative generiche.