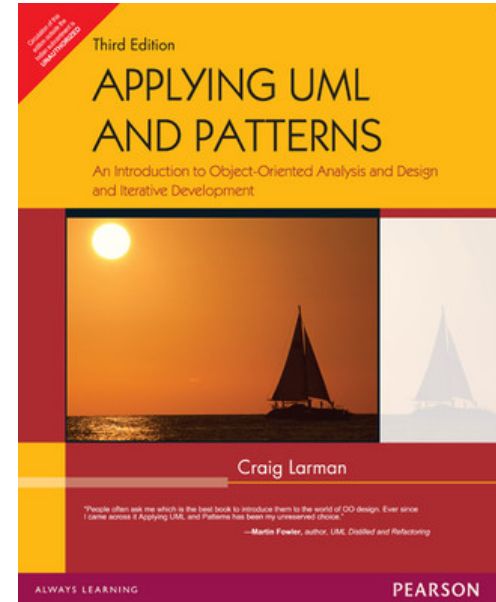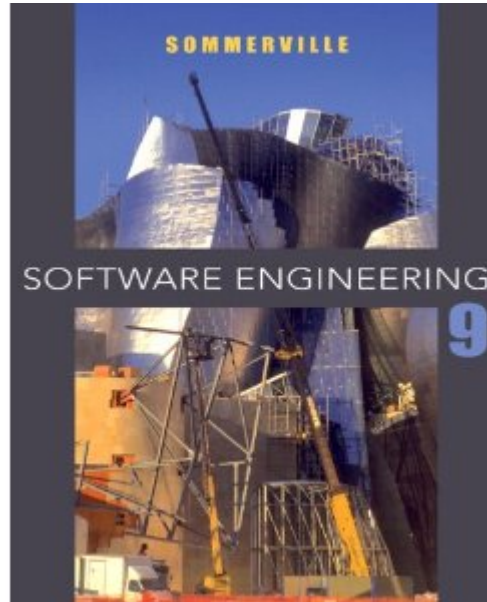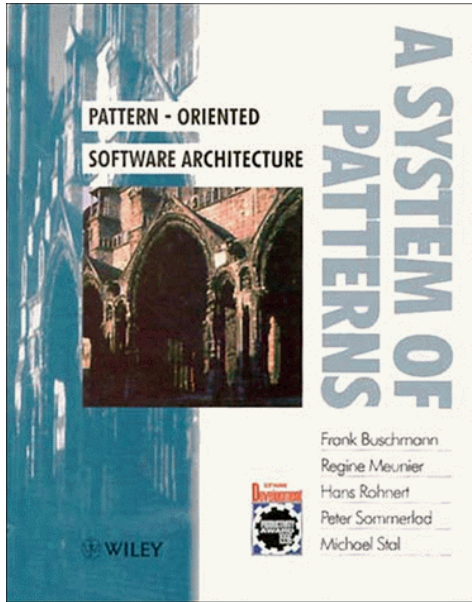# ARCHITECTURAL DESIGN

Sources

Definition

- Architectural design : the process for identifying the sub-systems and the framework for sub-systems control and communication

- The software architecture is the output of this process

Definition

- An architecture is the set of significant decisions about the organization of a software system, the selection of the structural elements and their interfaces by which the system is composed, together with their behaviour as specified in the collaborations among those elements, the composition of these structural and behavioural elements into progressively larger subsystems, and the architectural style that guides this organization---these elements and their interfaces, their collaborations, and their composition. [BRJ99]

Architecture analysis

- Investigation
  - Identify functional and non functional requirements that have an impact on system design
- Design
  - Resolution of these forces and requirements in the design of software, hardware, operations, policies,…

Construction analogy

- Architecture
  - Programs and relations between programs and their components
- Urbanisation (not seen here)
  - Architecture in the large
  - Enterprise systems
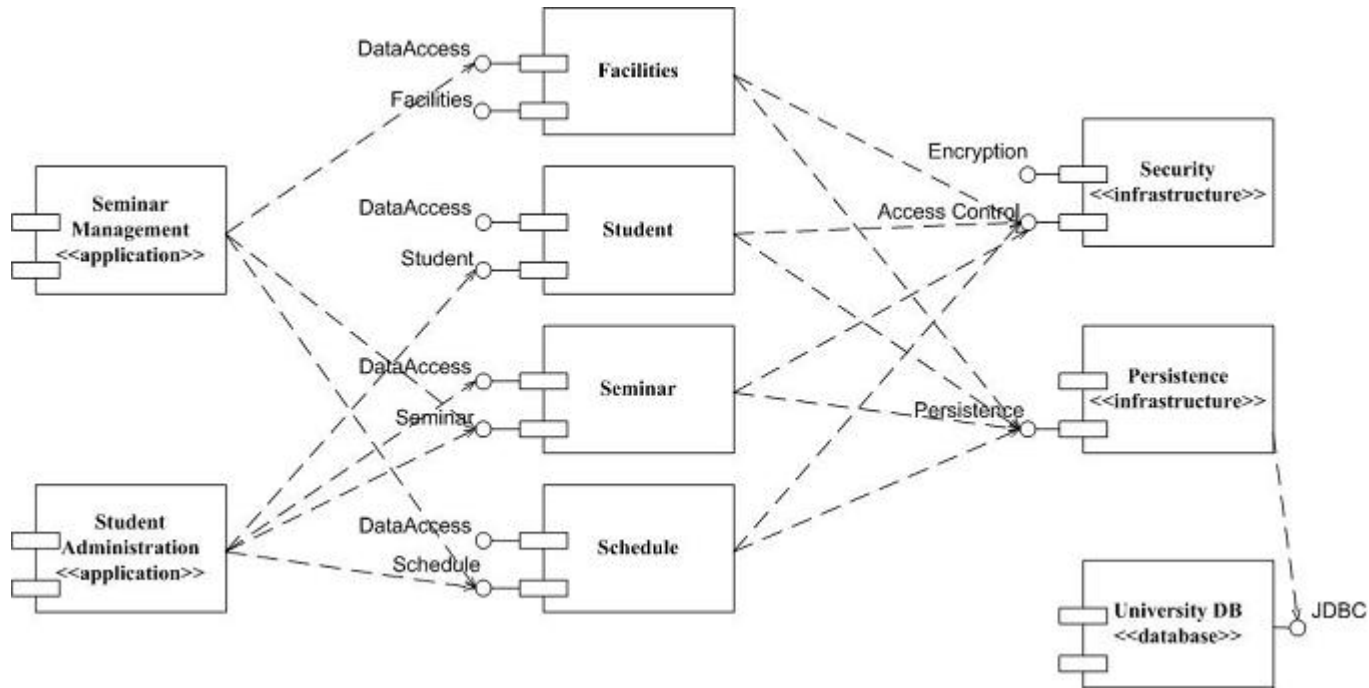
Explicit architecture

- Stakeholder communication
  - Architecture may be used as a focus of discussion by system stakeholders.
- System analysis
  - Means that analysis of whether the system can meet its non-functional requirements is possible.
- Large-scale reuse
  - The architecture may be reusable across a range of systems
  - Product-line architectures may be developed.

SE9

Architectural representation

- Box and lines
- Component diagrams
- Architectural models

Component Diagram

http://www.agilemodeling.com/artifacts/componentDiagram.htm

Architectural design decisions

- Is there a generic application architecture that can be used?
- How will the system be distributed?
- What architectural styles are appropriate?
- What approach will be used to structure the system?
- How will the system be decomposed into modules?
- What control strategy should be used?
- How will the architectural design be evaluated?
- How should the architecture be documented?

SE9

Architecture and system characteristics

- 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.

SE9

Architectural views

- What views or perspectives are useful when designing and documenting a system's architecture?
- What notations should be used for describing architectural models?
- Each architectural model only shows one view or perspective of the system.
  - It might show how a system is decomposed into modules, how the run-time processes interact or the different ways in which system components are distributed across a network. For both design and documentation, you usually need to present multiple views of the software architecture.

Chapter 6 Architectural design

4 + 1 view model of software architecture

- A logical view, which shows the key abstractions in the system as objects or object classes.
- A process view, which shows how, at run-time, the system is composed of interacting processes.
- A development view, which shows how the software is decomposed for development.
- A physical view, which shows the system hardware and how software components are distributed across the processors in the system.
- Related using use cases or scenarios (+1)

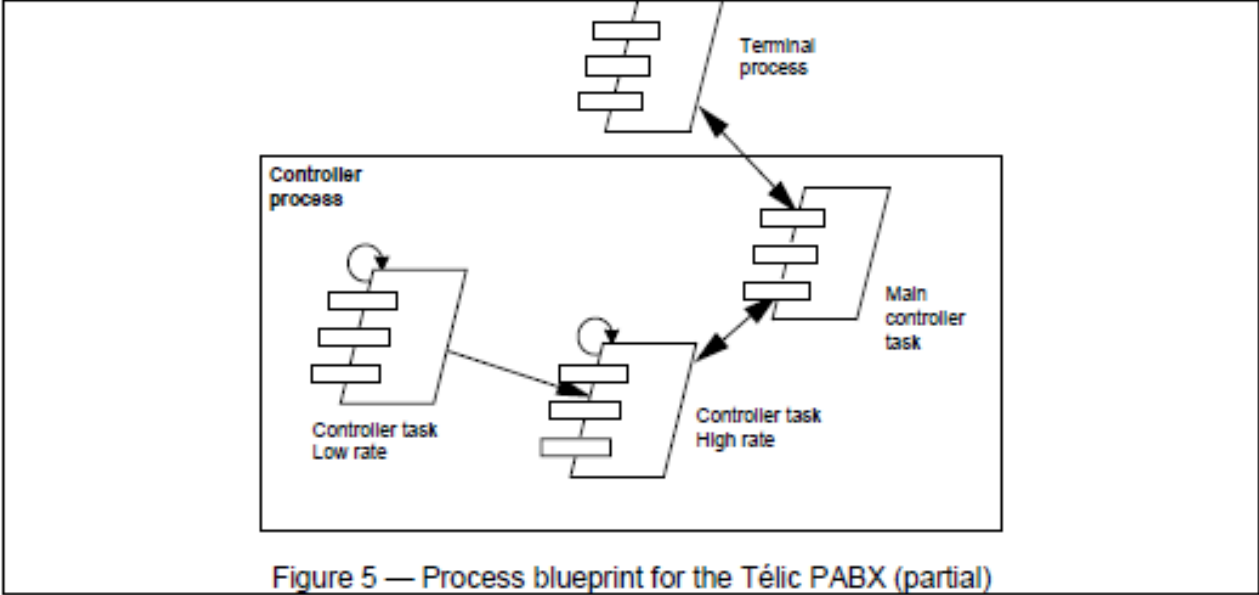Chapter 6 Architectural design
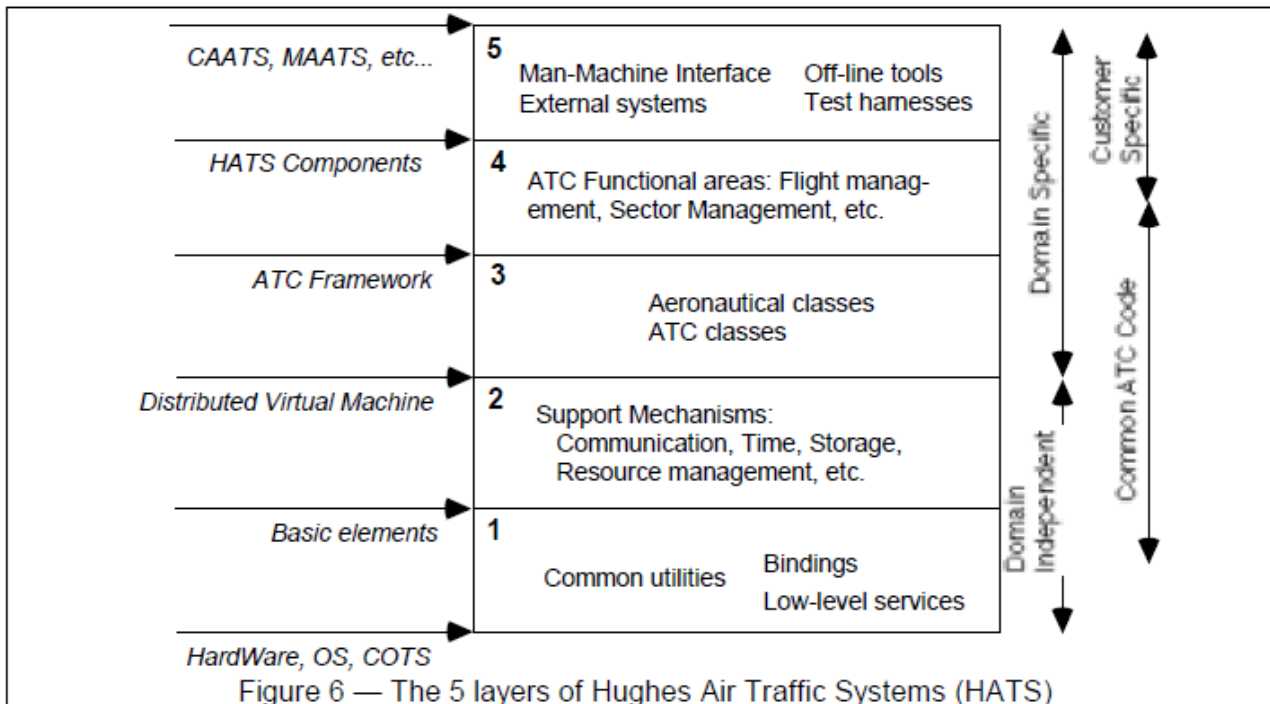
The views

Figure 1 — The "4+1" view model
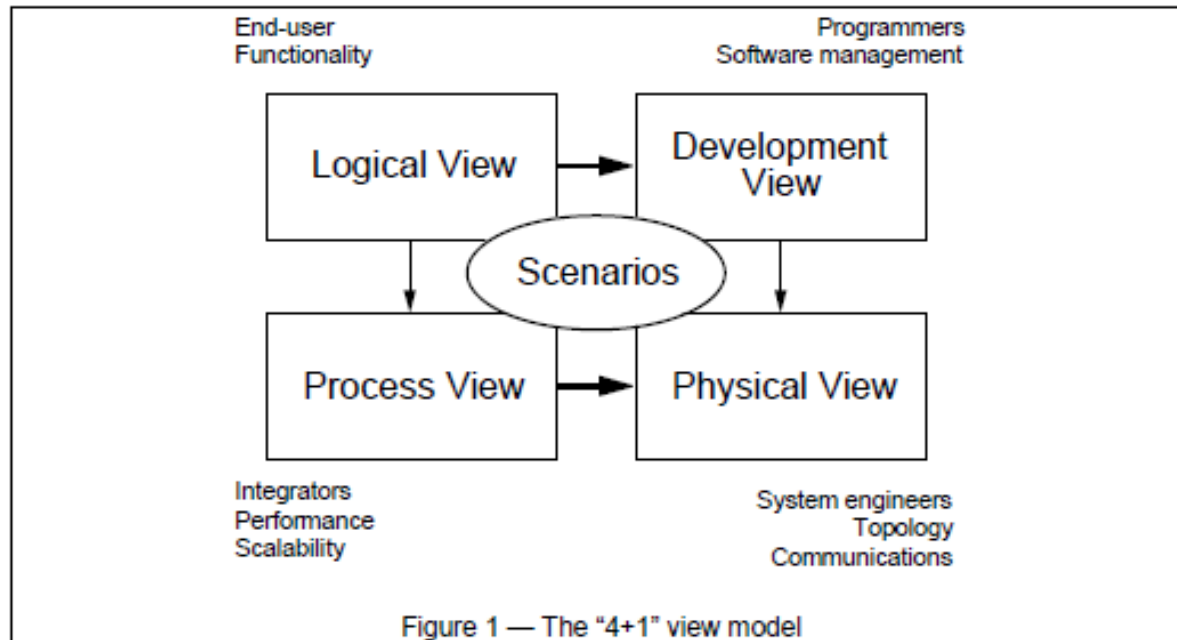
Kruchten, Philippe (1995, November).
Architectural Blueprints — The "4+1" View Model of Software Architecture.

Example – Process Blueprint

Figure 5 — Process blueprint for the Télic PABX (partial)

Example – Development View

Figure 6 — The 5 layers of Hughes Air Traffic Systems (HATS)

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.

Chapter 6 Architectural design
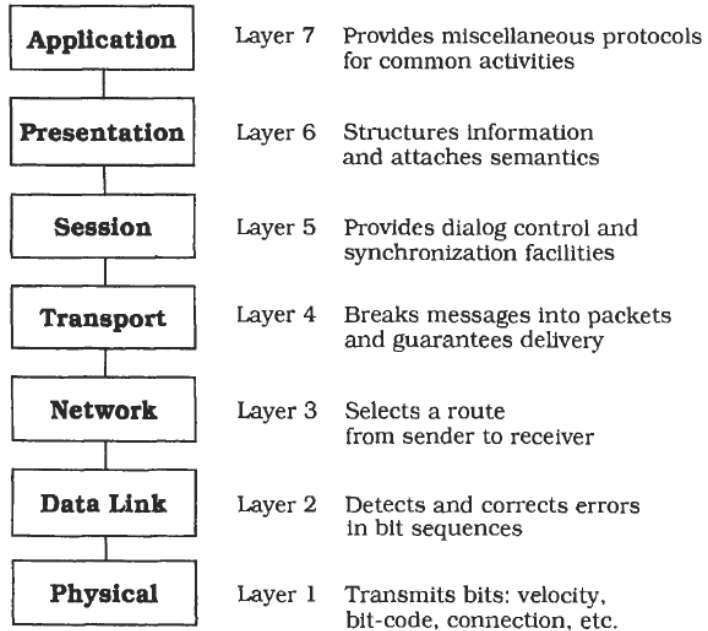
Architectural patterns

- Solutions for large scale, coarse grained design
- Same principles as Design Patterns

Layered architecture

- 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.

Chapter 6 Architectural design

Layer Example

Application	Layer 7	Provides miscellaneous protocols for common activities
Presentation	Layer 6	Structures information and attaches semantics
Session	Layer 5	Provides dialog control and synchronization facilities
Transport	Layer 4	Breaks messages into packets and guarantees delivery
Network	Layer 3	Selects a route from sender to receiver
Data Link	Layer 2	Detects and corrects errors in bit sequences
Physical	Layer 1	Transmits bits: velocity, bit-code, connection, etc.

The Layered architecture pattern
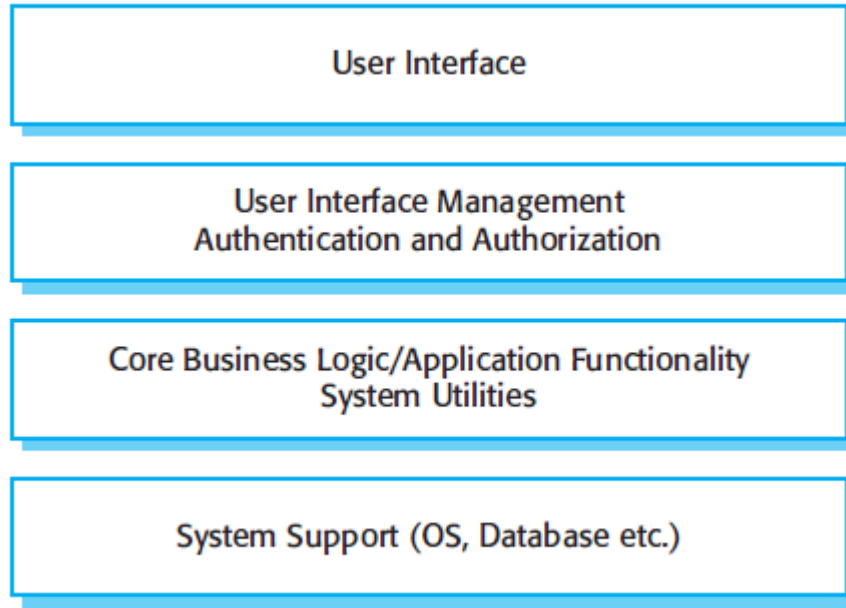
Name	Layered architecture
Description	Organizes the system into layers with related functionality associated with each layer. A layer provides services to the layer above it so the lowest-level layers represent core services that are likely to be used throughout the system. See Figure 6.6.
Example	A 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.

Chapter 6 Architectural design

Layer Structure

Class	Collaborator
Layer J	• Layer J-1
Responsibility • Provides services used by Layer J+1. • Delegates subtasks to Layer J-1.	

A generic layered architecture

User Interface

User Interface Management
Authentication and Authorization

Core Business Logic/Application Functionality
System Utilities

System Support (OS, Database etc.)

Chapter 6 Architectural design

The architecture of the LIBSYS system
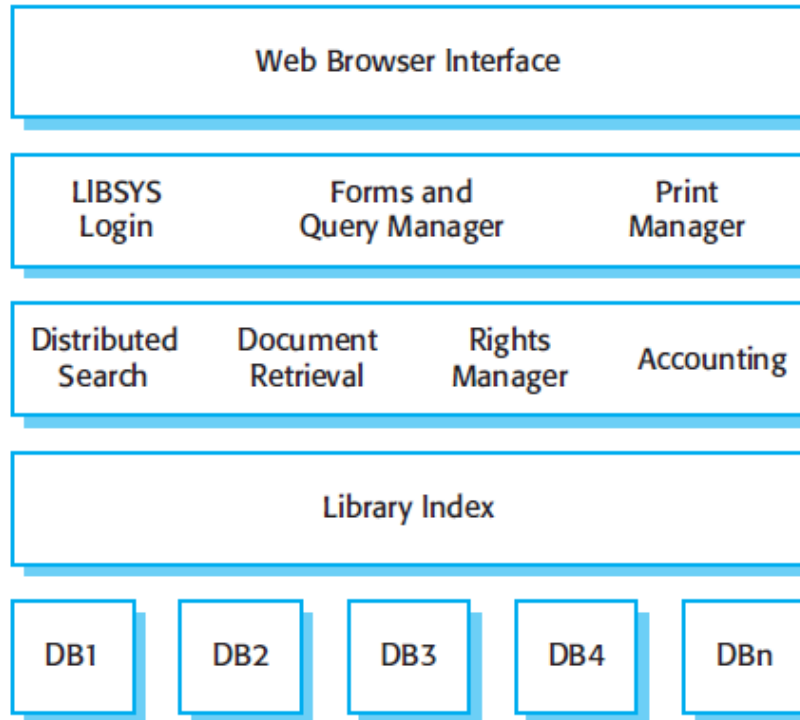
The Model-View-Controller (MVC)

| 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.
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.

Chapter 6 Architectural design

Organization of the Model-View-Controller

Chapter 6 Architectural design
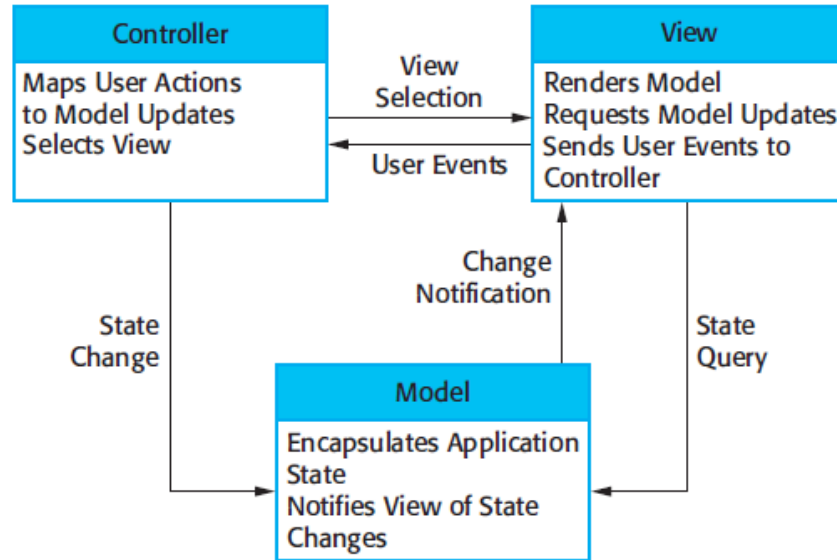
Web application architecture using the MVC pattern
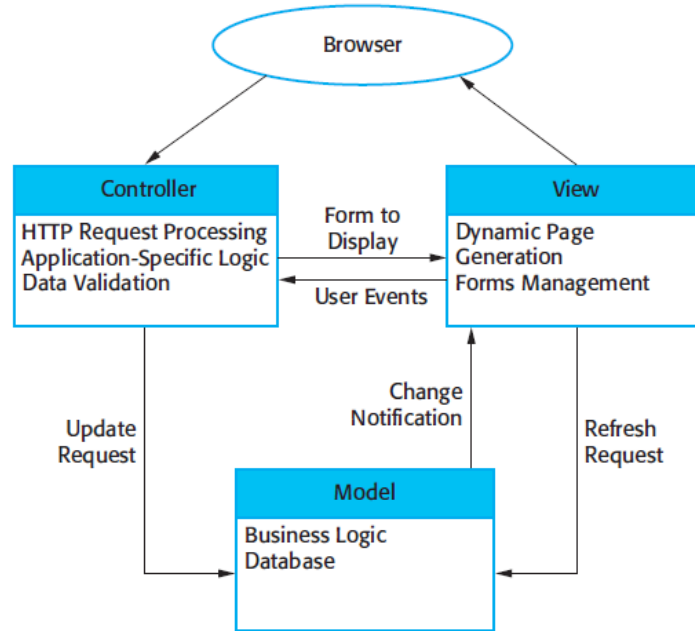
Key points

- A software architecture is a description of how a software system is organized.

- Architectural design decisions include decisions on the type of application, the distribution of the system, the architectural styles to be used.

- Architectures may be documented from several different perspectives or views such as a conceptual view, a logical view, a process view, and a development view.

- Architectural patterns are a means of reusing knowledge about generic system architectures. They describe the architecture, explain when it may be used and describe its advantages and disadvantages.

Chapter 6 Architectural design

Repository architecture

- 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 a this is an efficient data sharing mechanism.

Chapter 6 Architectural design

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.

Chapter 6 Architectural design

A repository architecture for an IDE

Chapter 6 Architectural design
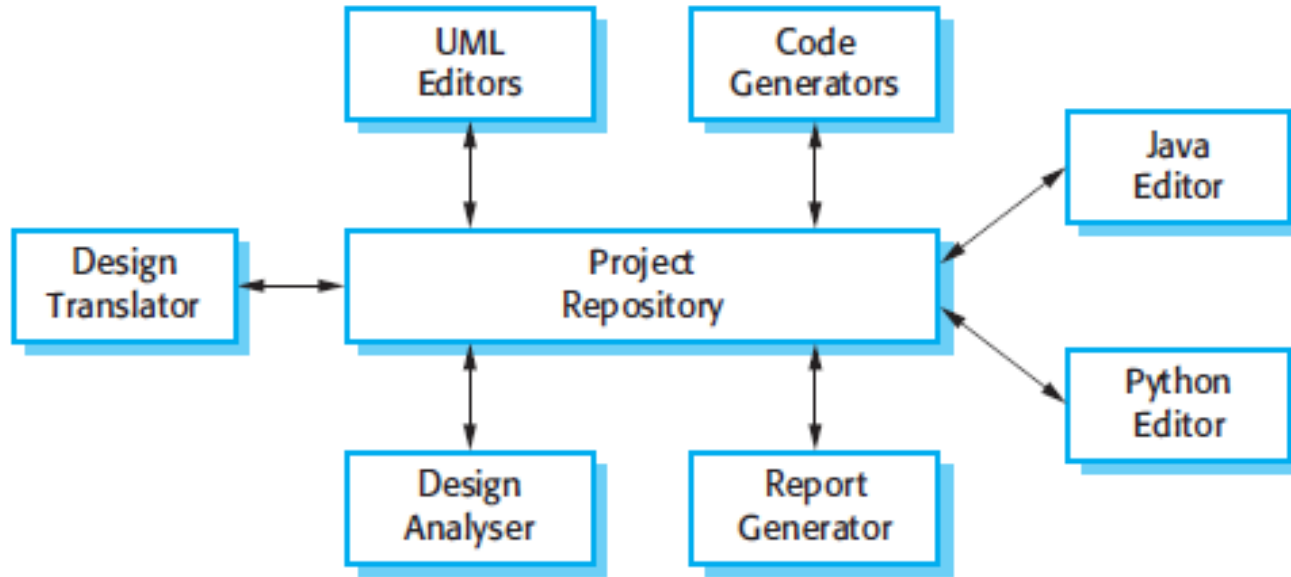
Client-server architecture

- Distributed system model which shows how data and processing is distributed across a range of components.
  - Can be implemented on a single computer.
- 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.

Chapter 6 Architectural design

The Client–server 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.

Chapter 6 Architectural design

A client–server architecture for a film library

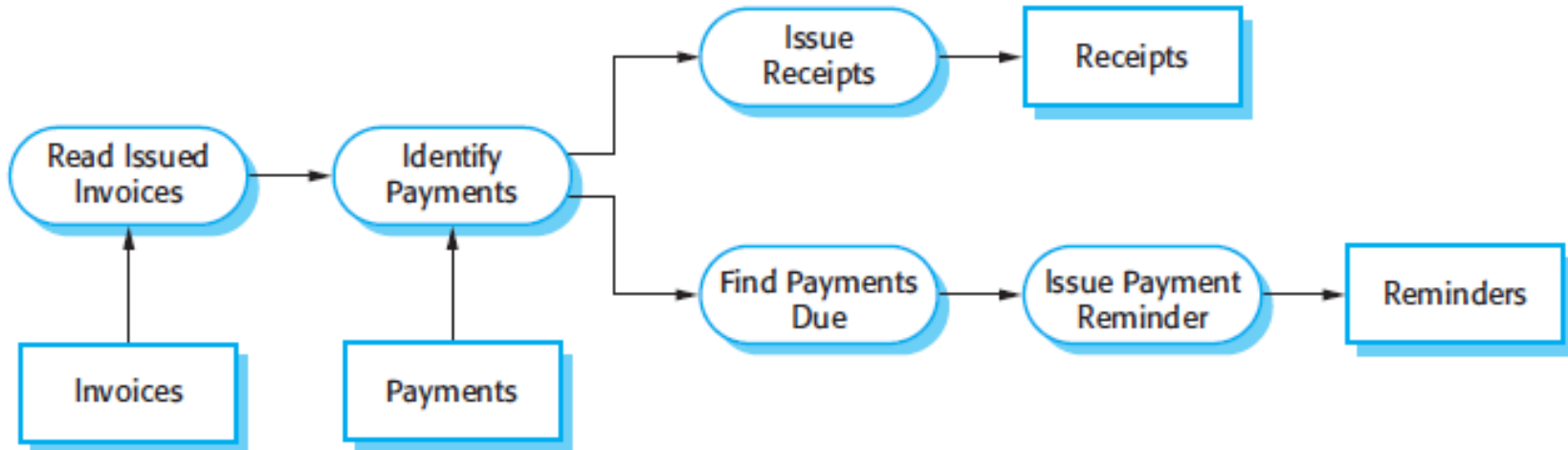Chapter 6 Architectural design
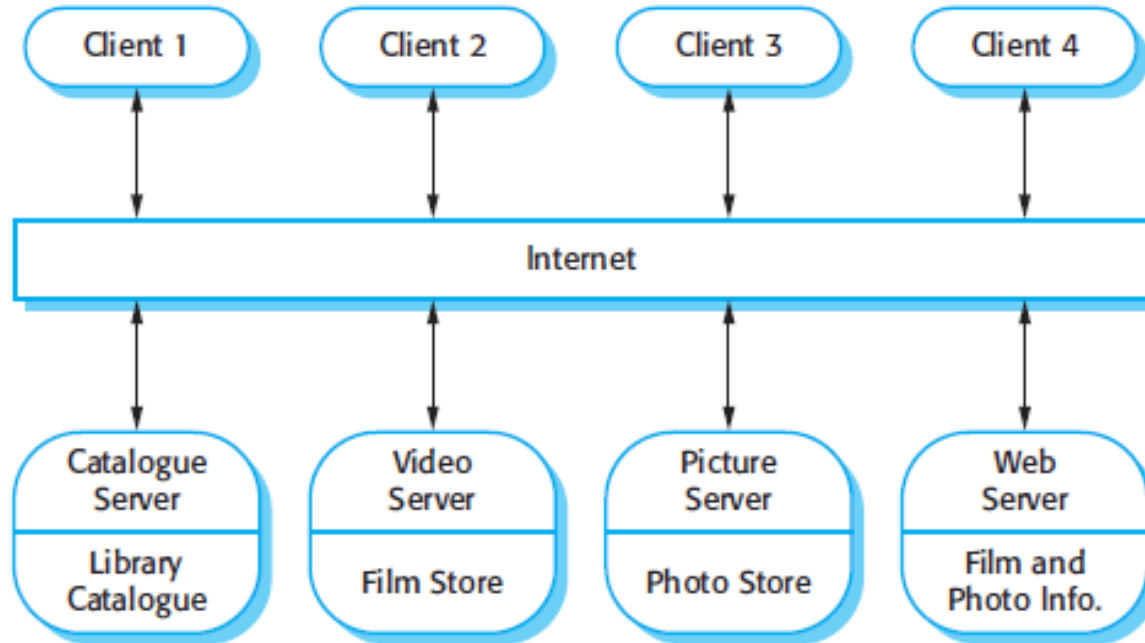
Pipe and filter architecture

- Functional transformations process their inputs to produce outputs.
- May be referred to as a pipe and filter model (as in 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.

Chapter 6 Architectural design

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	Figure 6.13 is 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) 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.

Chapter 6 Architectural design

An example of the pipe and filter architecture

Chapter 6 Architectural design

Responsibilities/Collaboration

Class	Collaborators
Filter	• Pipe
Responsibility	
• Gets input data.	
• Performs a function on its input data.	
• Supplies output data.	

Class	Collaborators
Pipe	• Data Source
	• Data Sink
	• Filter
Responsibility	
• Transfers data.	
• Buffers data.	
• Synchronizes active neighbors.	

Class	Collaborators
Data Source	• Pipe
Responsibility	
• Delivers input to processing pipeline.	

Class	Collaborators
Data Sink	• Pipe
Responsibility	
• Consumes output.	

Scenario 1
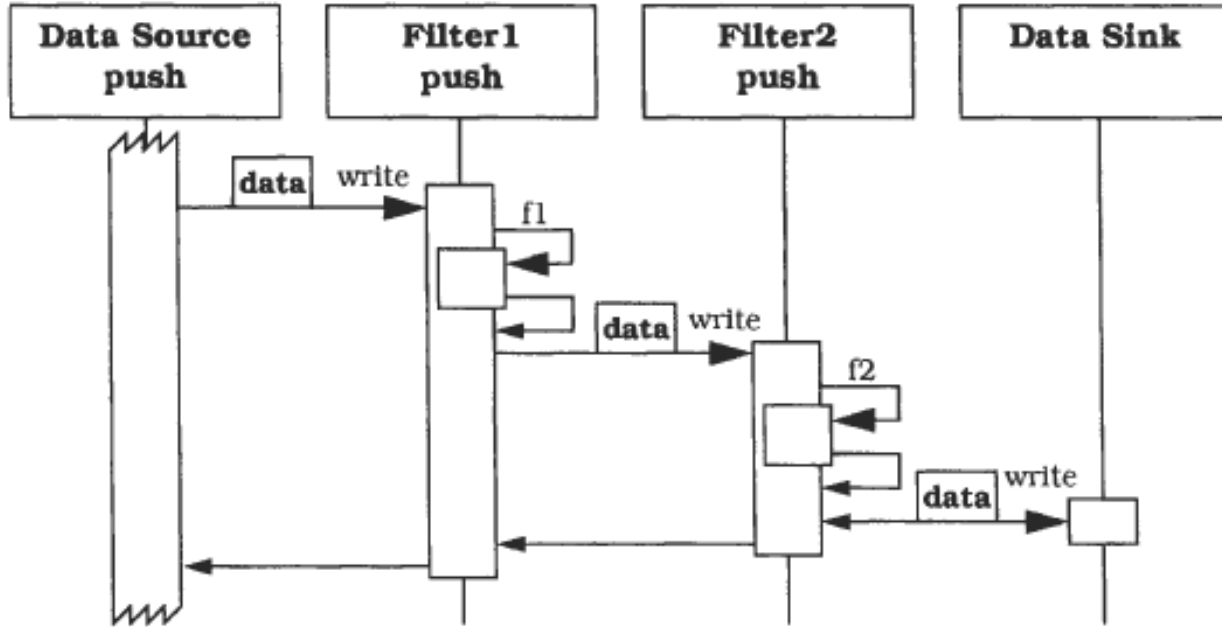
Scenario 2
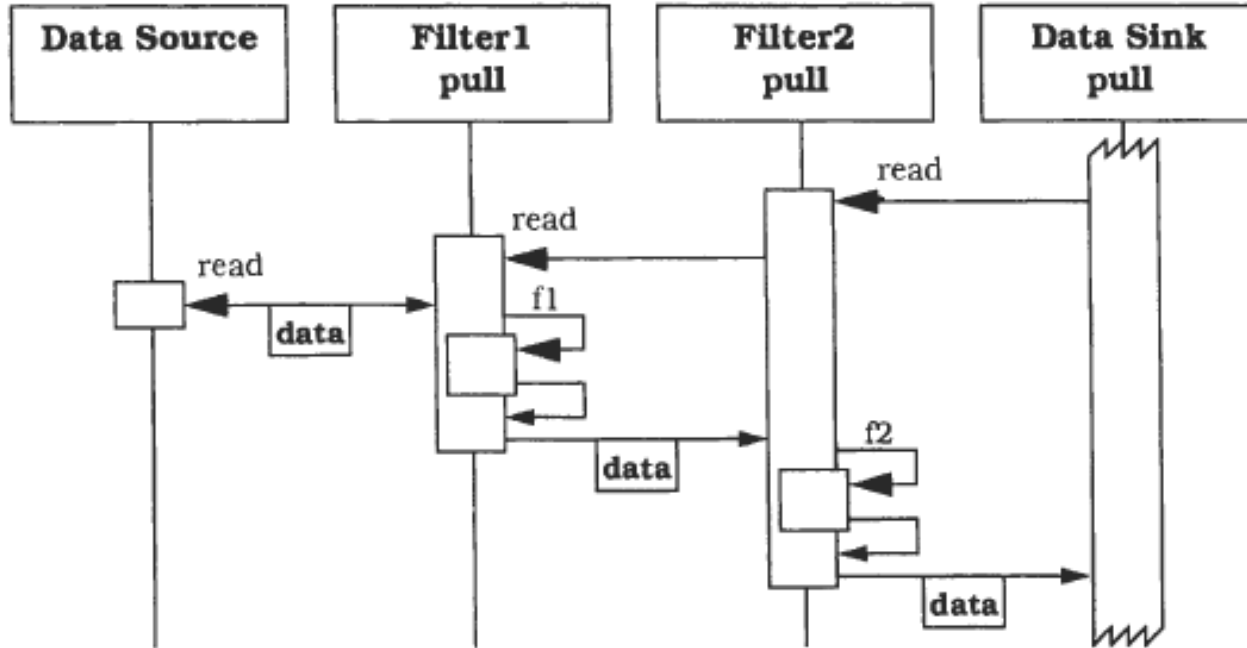
More complex Scenario

Broker

| Name	Broker
Description	Introduce a broker component to achieve better decoupling of clients and servers. Servers register themselves with the broker, and make their services available to cllents through method interfaces. Clients access the functionality of servers by sending requests via the broker. A broker's tasks include locating the appropriate server, forwarding the request to the server and transmitting results and exceptions back to the client..
Example	CORBA, COM
When used	Your environment is a distributed and possibly heterogeneous system with independent cooperating components..
Advantages	Location Transparency, Changeability and extensibility of components, Portability of a Broker system, Reusability.
Disadvantages	Restricted effiency, Lower fault tolerance, Testing and Debugging.

Collaboration - Responsibilities

Class	Collaborators
Client-side Proxy	• Client • Broker
Responsibility • Encapsulates system-specific functionality. • Mediates between the client and the broker.	

| Class	Collaborators
Server-side Proxy	• Server • Broker
Responsibility • Calls services within the server. • Encapsulates system-specific functionality. • Mediates between the server and the broker.	

Class	Collaborators
Broker	• Client • Server • Client-side Proxy • Server-side Proxy • Bridge
Responsibility • (Un-)registers servers. • Offers APIs. • Transfers messages. • Error recovery. • Interoperates with other brokers through bridges. • Locates servers.	

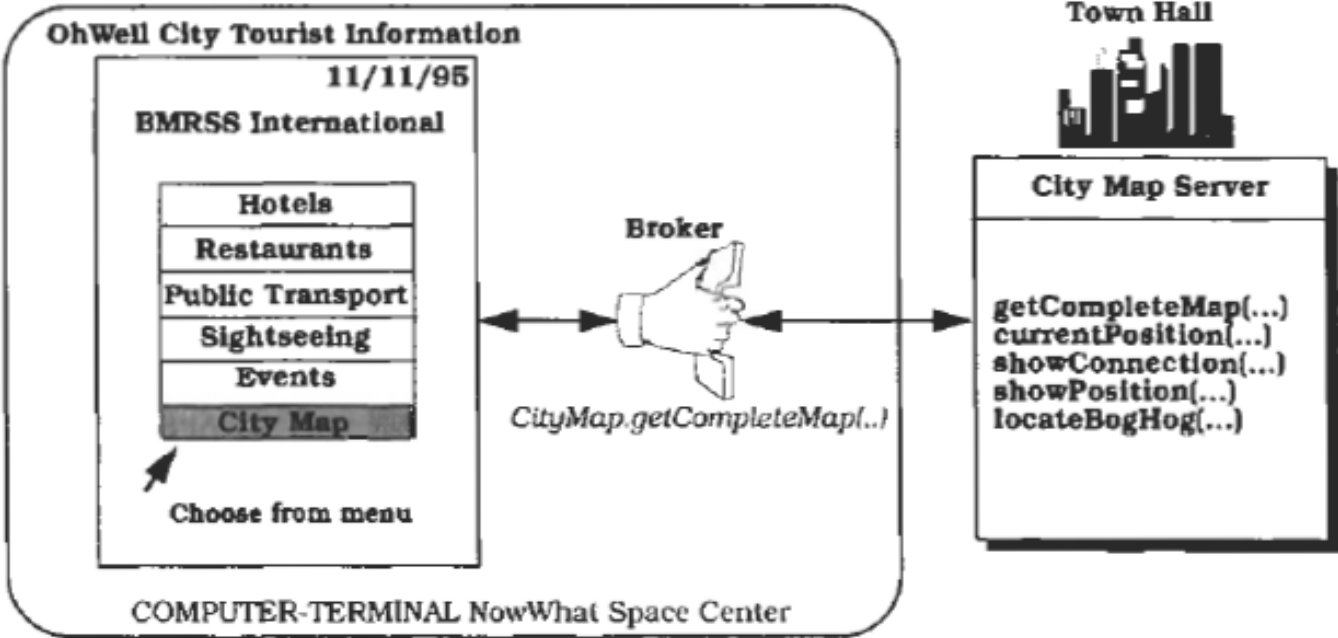| Class	Collaborators
Bridge	• Broker • Bridge
Responsibility • Encapsulates network-specific functionality. • Mediates between the local broker and the bridge of a remote broker.	

Communications
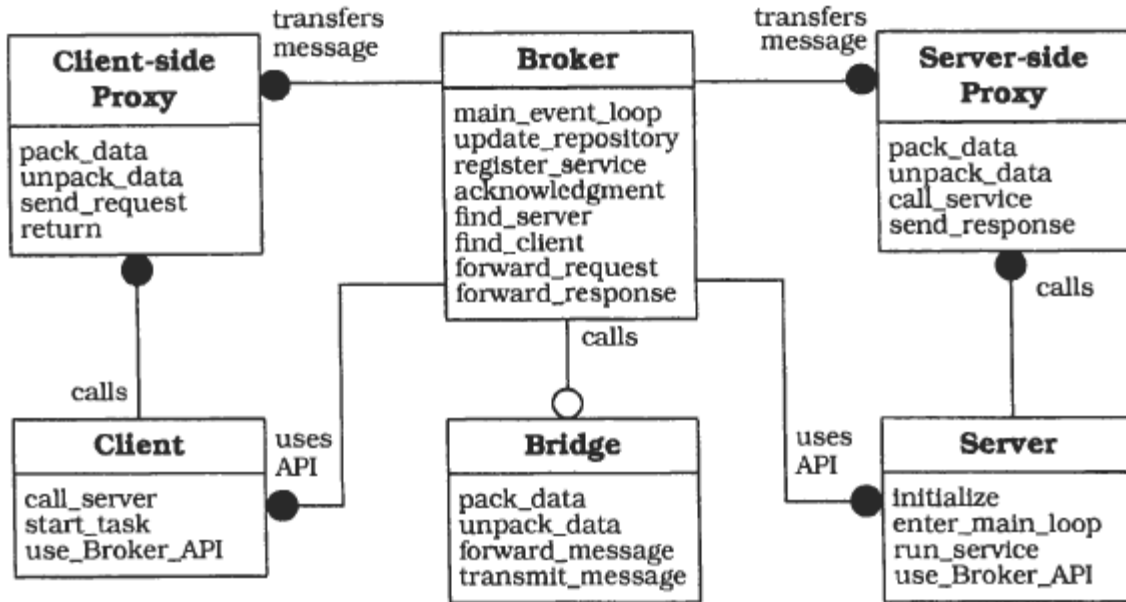
Scenarios
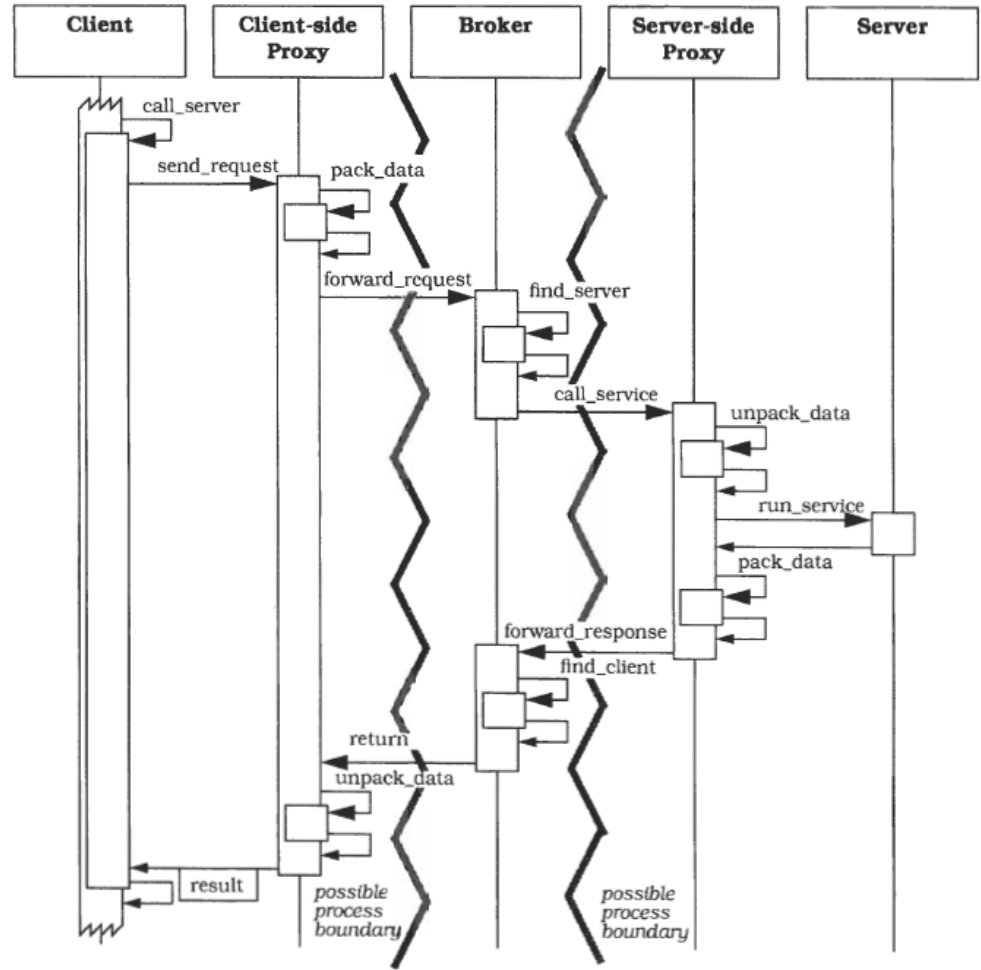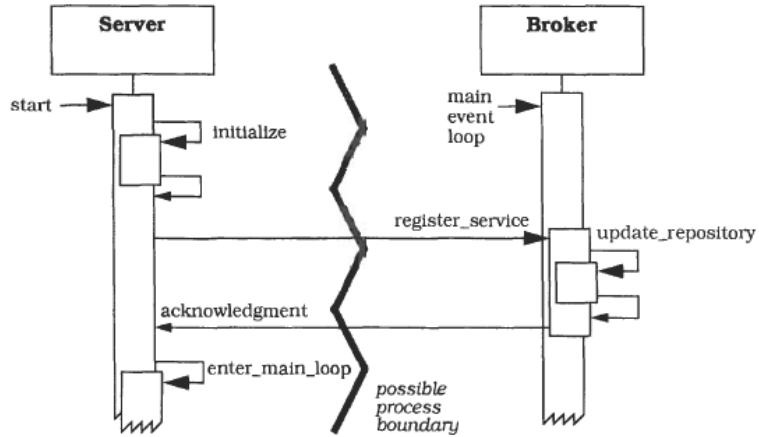
Example

Other patterns

- Blackboard
- Presentation-Abstraction-Control
- Microkernel
- Reflection

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.

Chapter 6 Architectural design

Use of application architectures

- 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

- Data processing applications
  - Data driven applications that process data in batches without explicit user intervention during the processing.
- Transaction processing applications
  - Data-centred applications that process user requests and update information in a system database.
- Event processing systems
  - Applications where system actions depend on interpreting events from the system's environment.
- Language processing systems
  - Applications where the users' intentions are specified in a formal language that is processed and interpreted by the system.

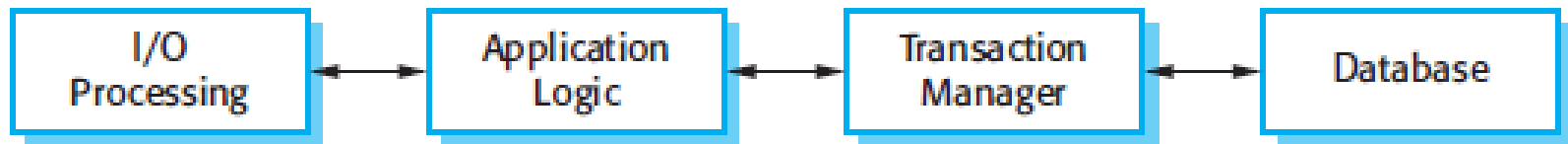Chapter 6 Architectural design

Application type examples

- Focus here is on transaction processing and language processing systems.
- Transaction processing systems
  - E-commerce systems;
  - Reservation systems.
- Language processing systems
  - Compilers;
  - Command interpreters.

Chapter 6 Architectural design

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.
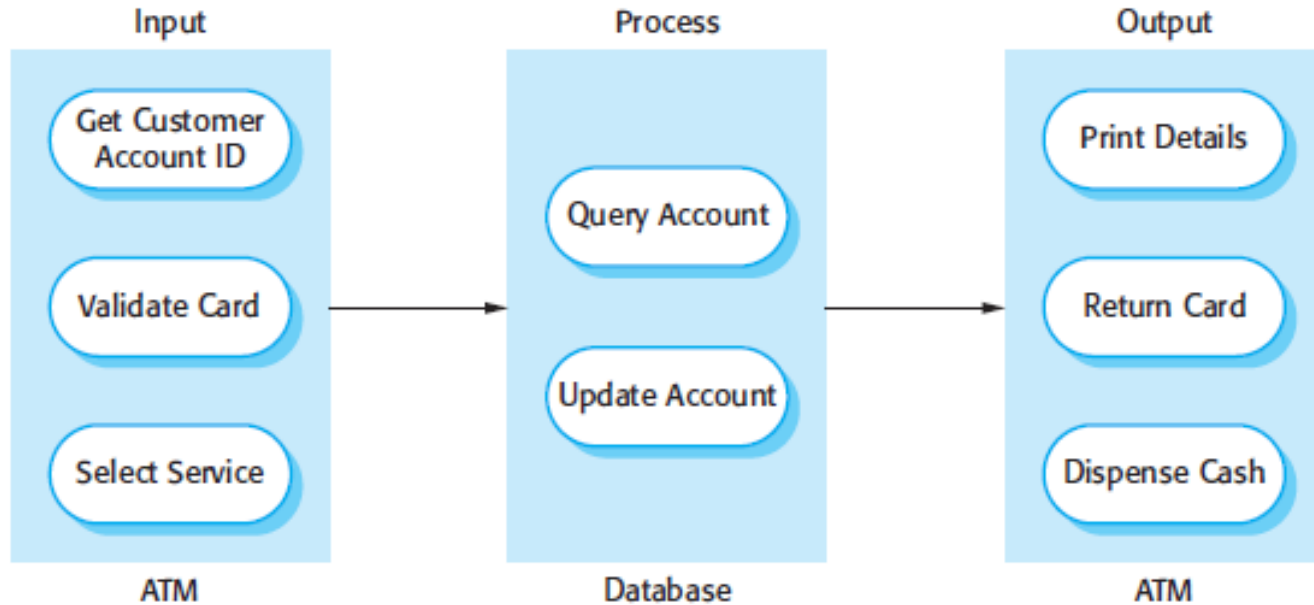
The structure of transaction processing applications

```
I/O              Application         Transaction         Database
Processing    ←→    Logic      ←→     Manager      ←→
```

Chapter 6 Architectural design

The software architecture of an ATM system

Chapter 6 Architectural design

Information systems architecture

- 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
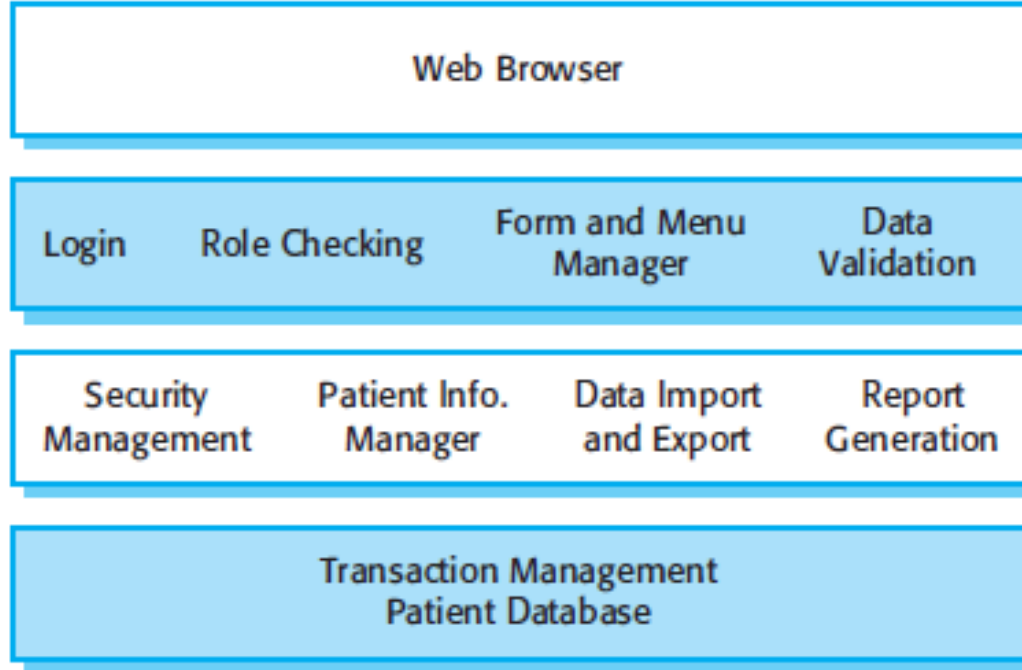 - Information retrieval
  - System database

Chapter 6 Architectural design

Layered information system architecture

Chapter 6 Architectural design
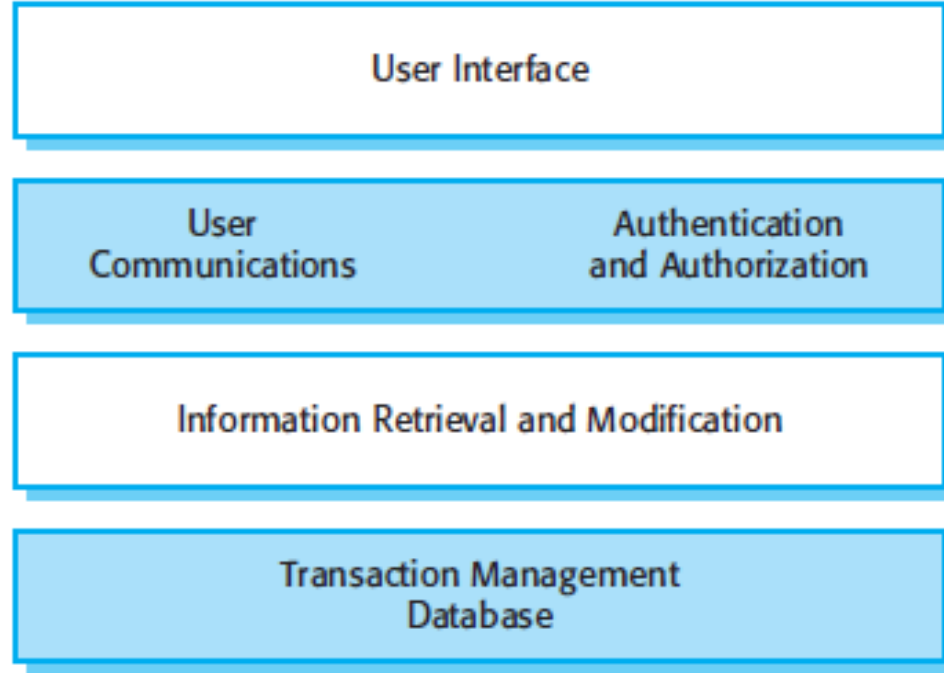
The architecture of the MHC-PMS

Chapter 6 Architectural design

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.

Chapter 6 Architectural design

Server implementation

◇ These systems are often implemented as multi-tier client server/architectures (discussed in Chapter 18)

- The web server is responsible for all user communications, with the user interface implemented using a web browser;

- The application server is responsible for implementing application-specific logic as well as information storage and retrieval requests;

- The database server moves information to and from the database and handles transaction management.

Chapter 6 Architectural design

Language processing systems

- Accept a natural or artificial language as input and generate some other representation of that language.
- May include an interpreter to act on the instructions in the language that is being processed.
- Used in situations where the easiest way to solve a problem is to describe an algorithm or describe the system data
    - Meta-case tools process tool descriptions, method rules, etc and generate tools.

Chapter 6 Architectural design

The architecture of a language processing system
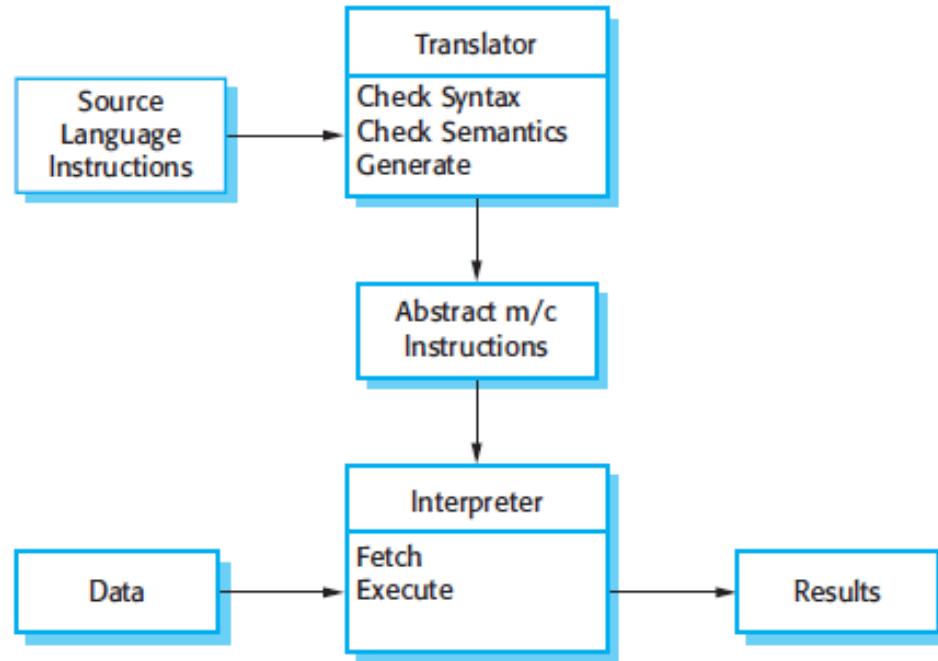
Compiler components

- 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.

Chapter 6 Architectural design

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.
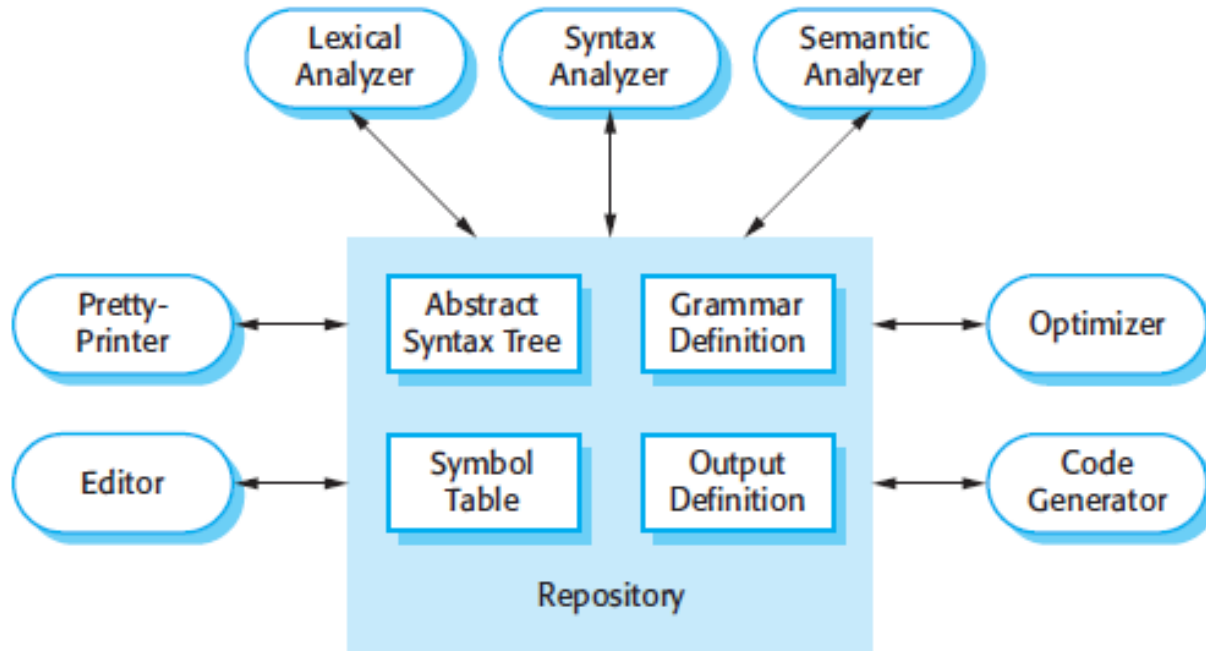
Chapter 6 Architectural design

A pipe and filter compiler architecture

Chapter 6 Architectural design
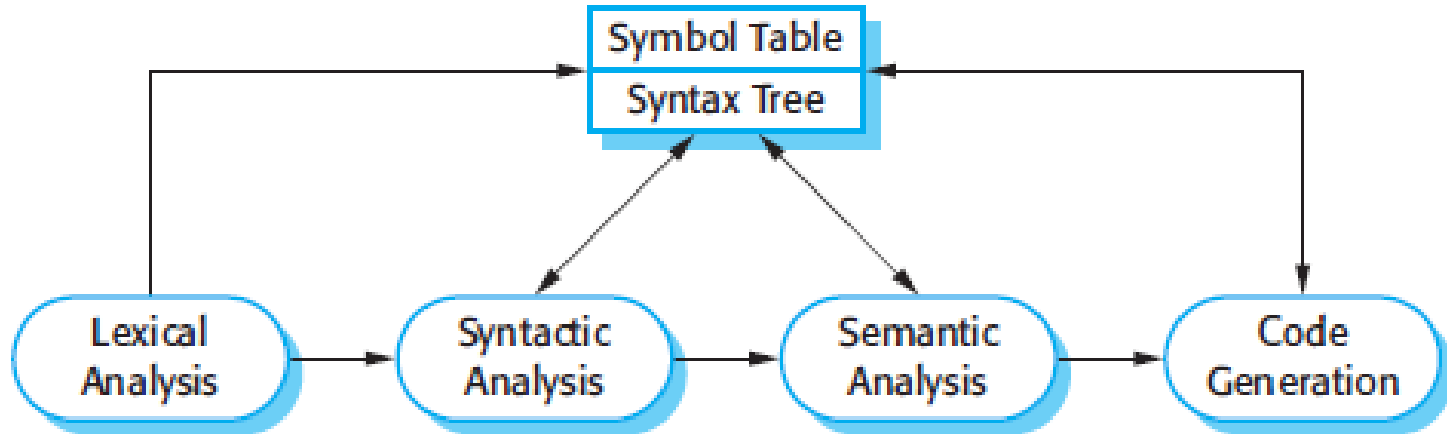
A repository architecture for a language processing system

Chapter 6 Architectural design

Key points

- Models of application systems architectures help us understand and compare applications, validate application system designs and assess large-scale components for reuse.

- Transaction processing systems are interactive systems that allow information in a database to be remotely accessed and modified by a number of users.

- Language processing systems are used to translate texts from one language into another and to carry out the instructions specified in the input language. They include a translator and an abstract machine that executes the generated language.

Chapter 6 Architectural design