



THIẾT KẾ KIẾN TRÚC (ARCHITECTURE DESIGN)

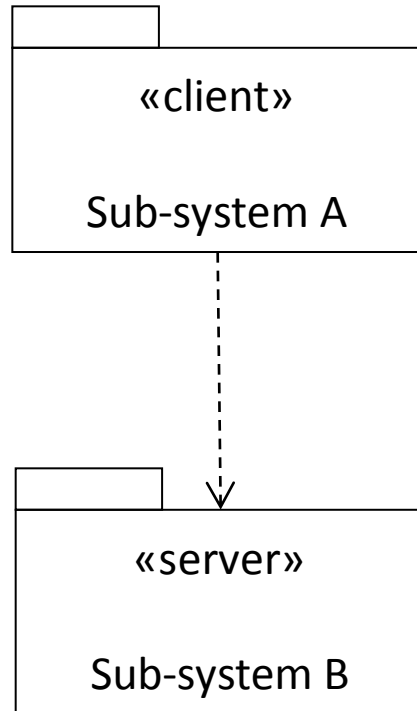
Software architecture

- 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.

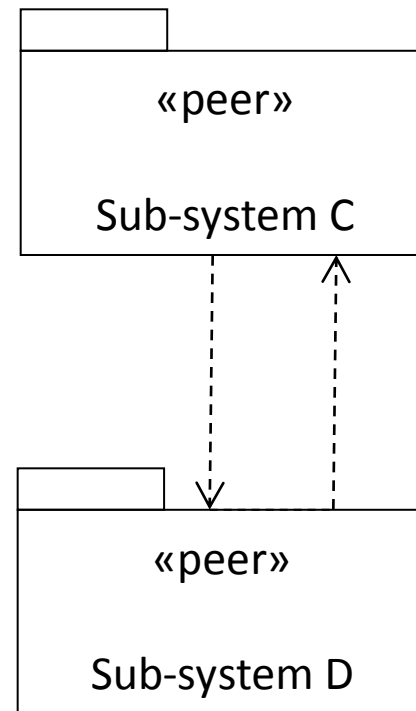
Software Architecture

- ‘A *software architecture* is a description of the sub-systems and components of a software system and the relationships between them.
- The architecture of the information system is first considered early in the project during the requirements capture and analysis activities.
- We will consider some of the major architecture styles
 - Client/Server Architecture
 - Peer-to-Peer Architecture
 - Layered Architecture
 - Model/View/Controller
 - Broker Architecture

Styles of communication between sub-systems



The server sub-system does not depend on the client sub-system and is not affected by changes to the client's interface.



Each peer sub-system depends on the other and each is affected by changes in the other's interface.

Styles of communication between sub-systems

- Client–server communication
 - Client–server communication requires the client to know the interface of the server sub-system, but the communication is only in one direction
 - The client sub-system requests services from the server sub-system and not vice versa
- Peer-to-peer communication
 - Peer-to-peer communication requires each sub-system to know the interface of the other, thus coupling them more tightly
 - The communication is two way since either peer sub-system may request services from the other

Layering and Partitioning

- Two general approaches to the division of a software system into sub-systems
 - *Layering*—so called because the different sub-systems usually represent different levels of abstraction
 - *Partitioning*, which usually means that each sub-system focuses on a different aspect of the functionality of the system as a whole
- Both approaches are often used together on one system

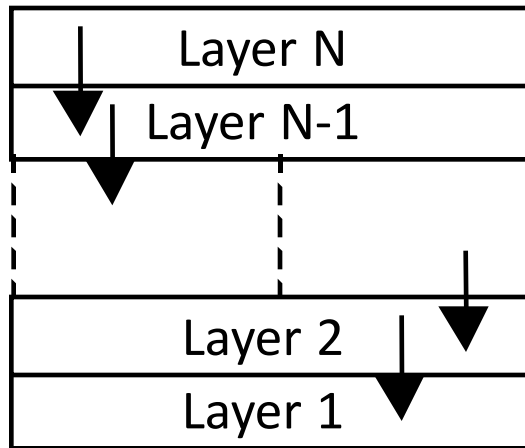
Layered architecture

- 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.

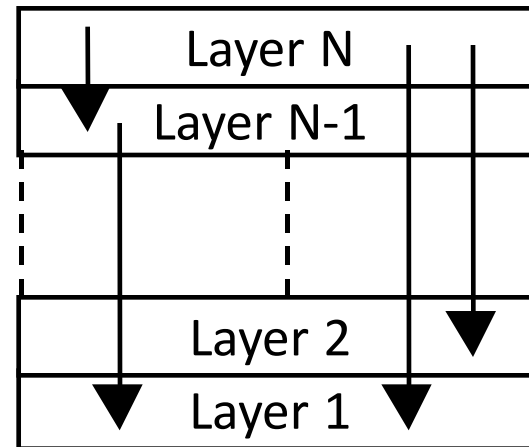
The Layered architecture pattern

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.

Schematic of a Layered Architecture

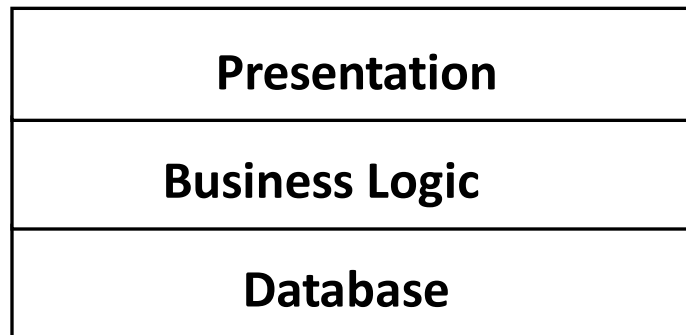


*Closed architecture—
messages may only be
sent to the adjacent
lower layer*

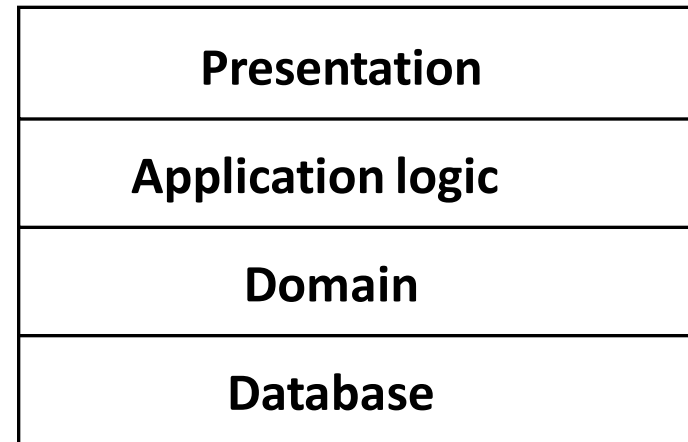


*Open architecture—
messages can be sent
to any lower layer*

Three & Four Layer Architectures.



-
-
-



Business logic layer
can be split into two layers

Partitioned Sub-systems

**Four layer
architecture
applied to part
of the Video
Rental Shop
system**

Loosely coupled sub-systems, each
delivering a single service or coherent
group of services

Presentation layer

Application layer

Video Catalog HCI Sub-system	Renting Video HCI Sub-system
Video Catalog Sub-system	Renting Video Sub System
VideoShop Domain	
VideoShop Database	

*A single domain
layer supports
two application
sub-systems*

A generic layered architecture

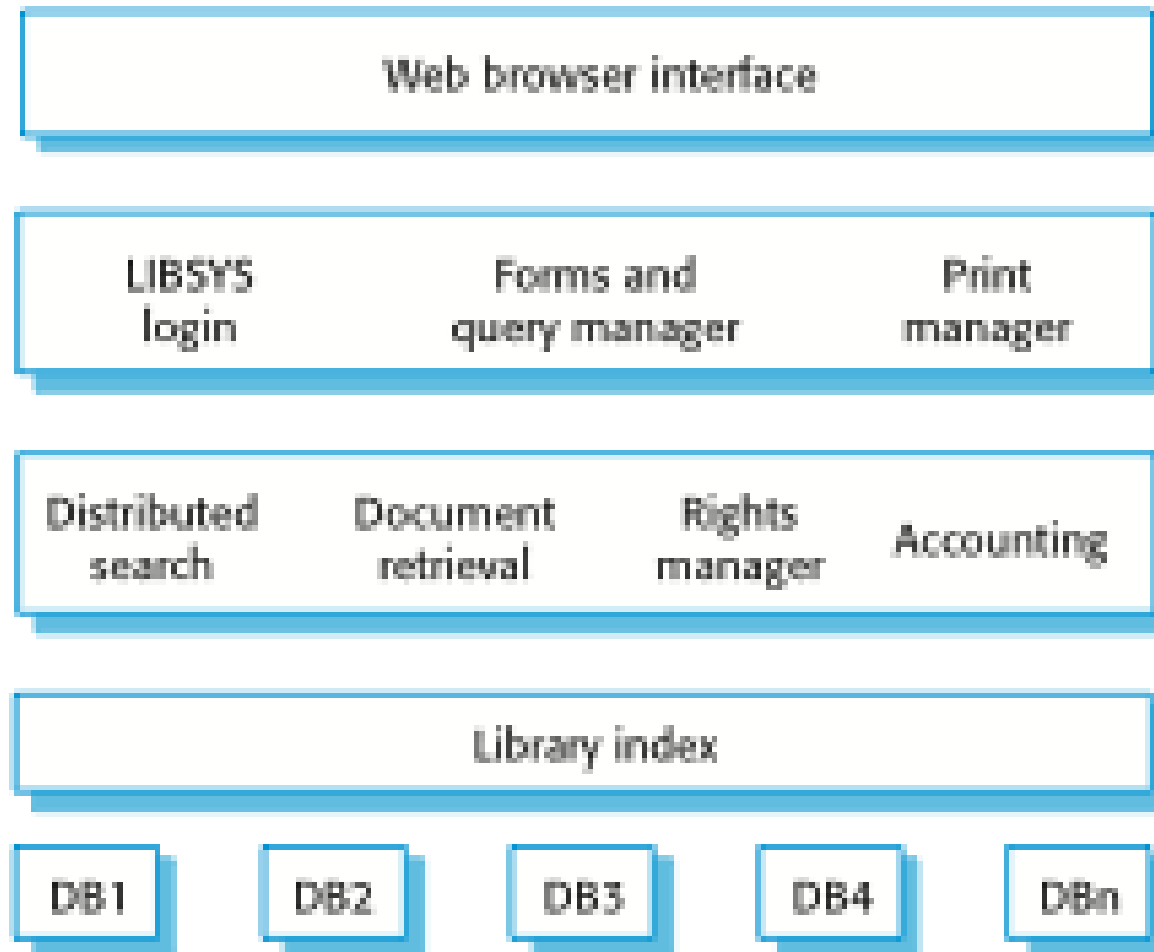
User interface

User interface management
Authentication and authorization

Core business logic/application functionality
System utilities

System support (OS, database etc.)

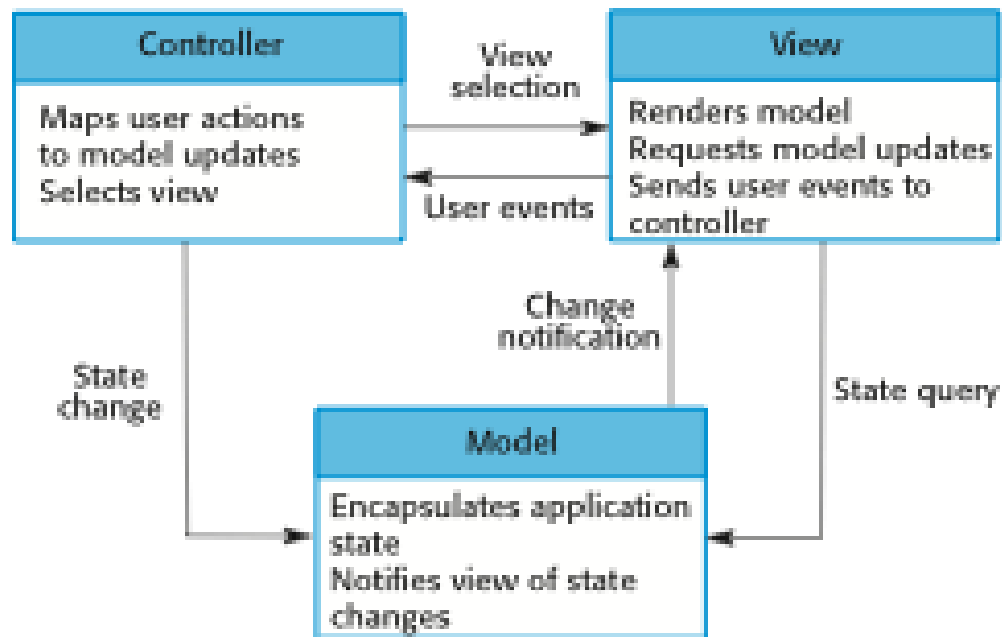
The architecture of the LIBSYS system



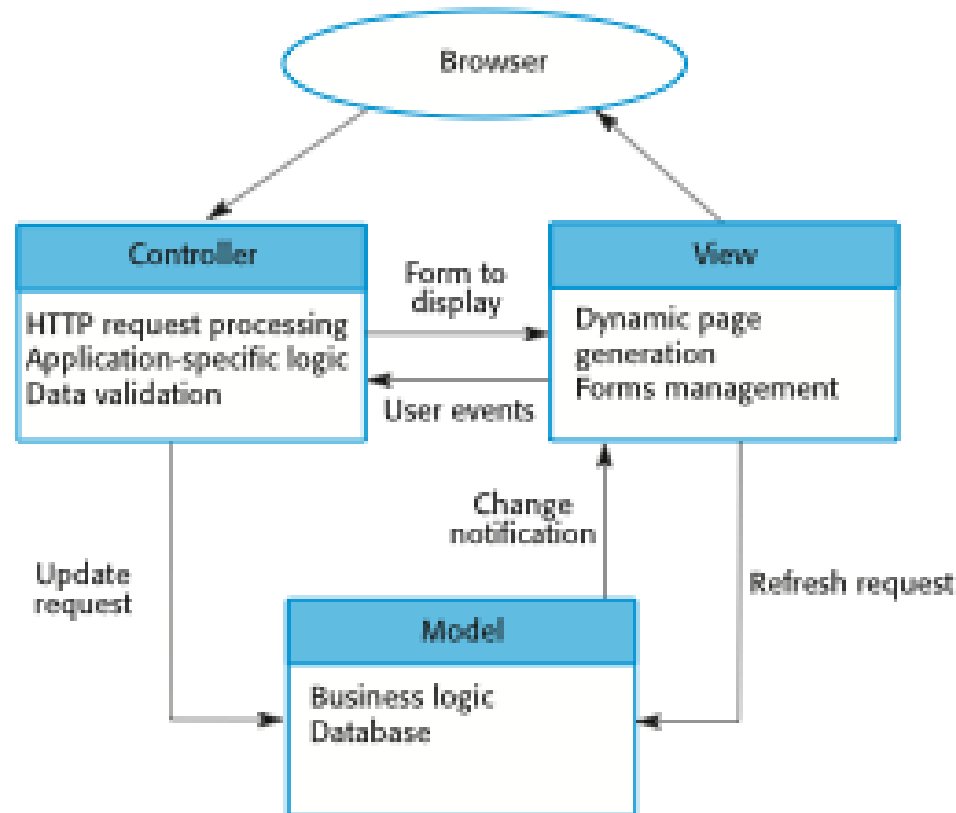
The Model-View-Controller (MVC) pattern

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.
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.

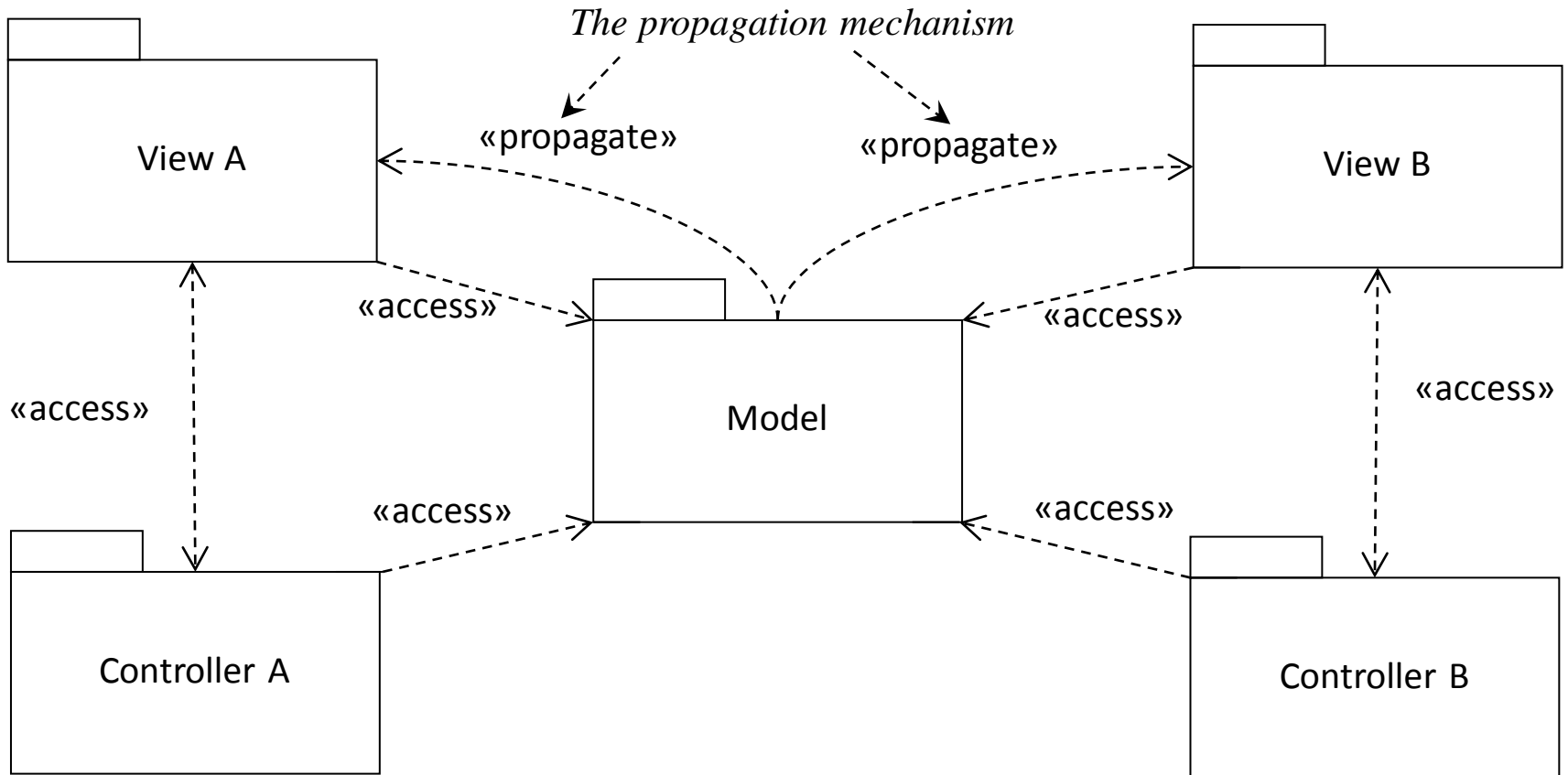
The organization of the Model-View-Controller



Web application architecture using the MVC pattern



Model-View-Controller

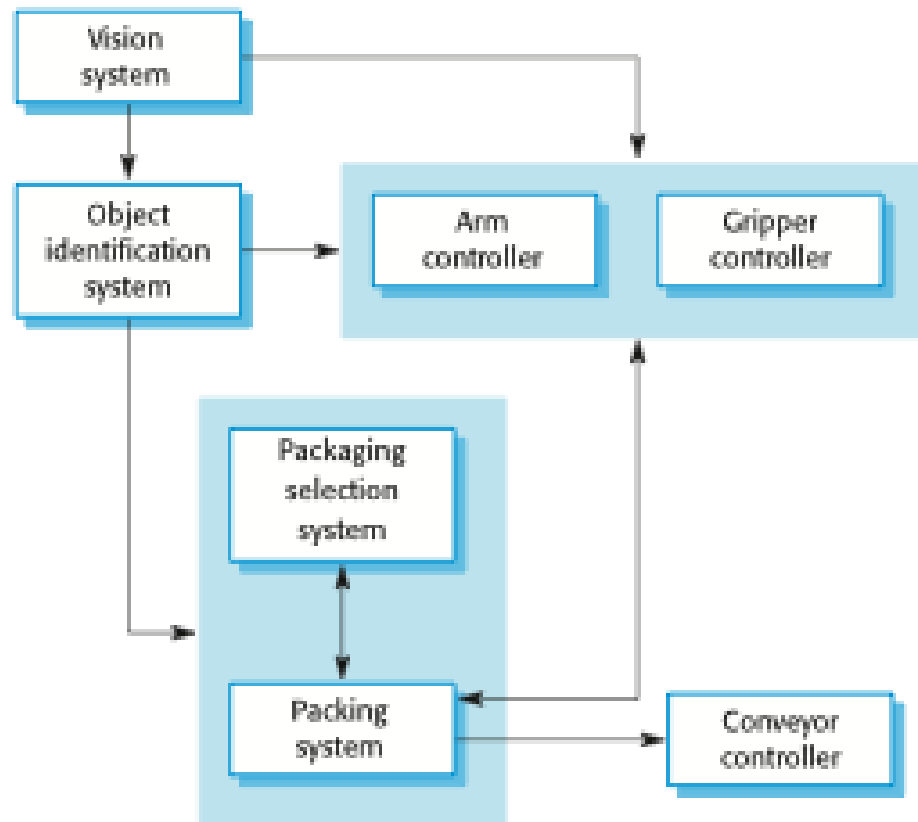


(Adapted from Hopkins and Horan, 1995)

What's architecture 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.

The architecture of a packing robot control system



Architectural design decisions

- Architectural design is a creative process so the process differs depending on the type of system being developed.
- However, a number of common decisions span all design processes and these decisions affect the non-functional characteristics of the system.

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?

Architecture and system characteristics

- Performance
 - Localize critical operations and minimize communications. Use large rather than fine-grain components.
- Security
 - Use a layered architecture with critical assets in the inner layers.
- Safety
 - Localize 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.

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)

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 they are not useful.
- Patterns may be represented using tabular and graphical descriptions.

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.

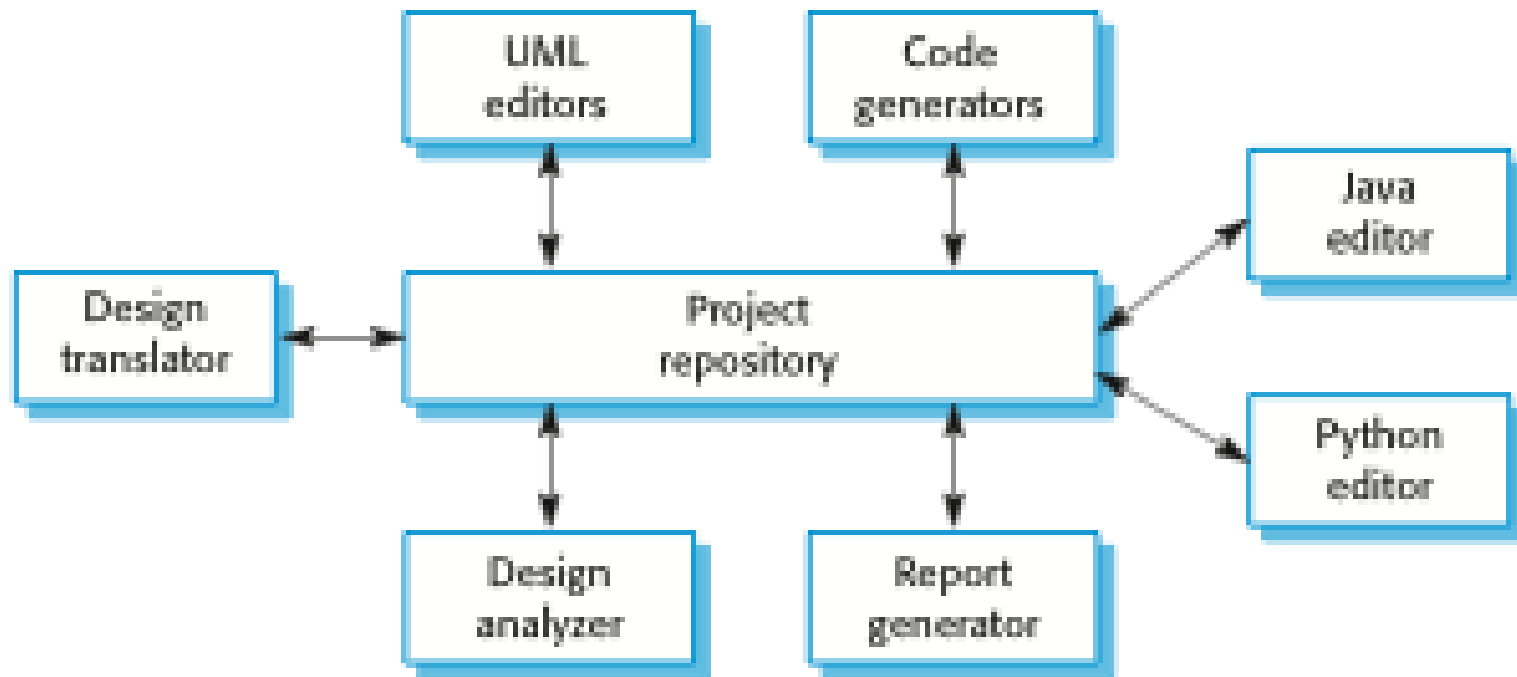
Repository architecture

- 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 as this is an efficient data sharing mechanism.

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.

A repository architecture for an IDE



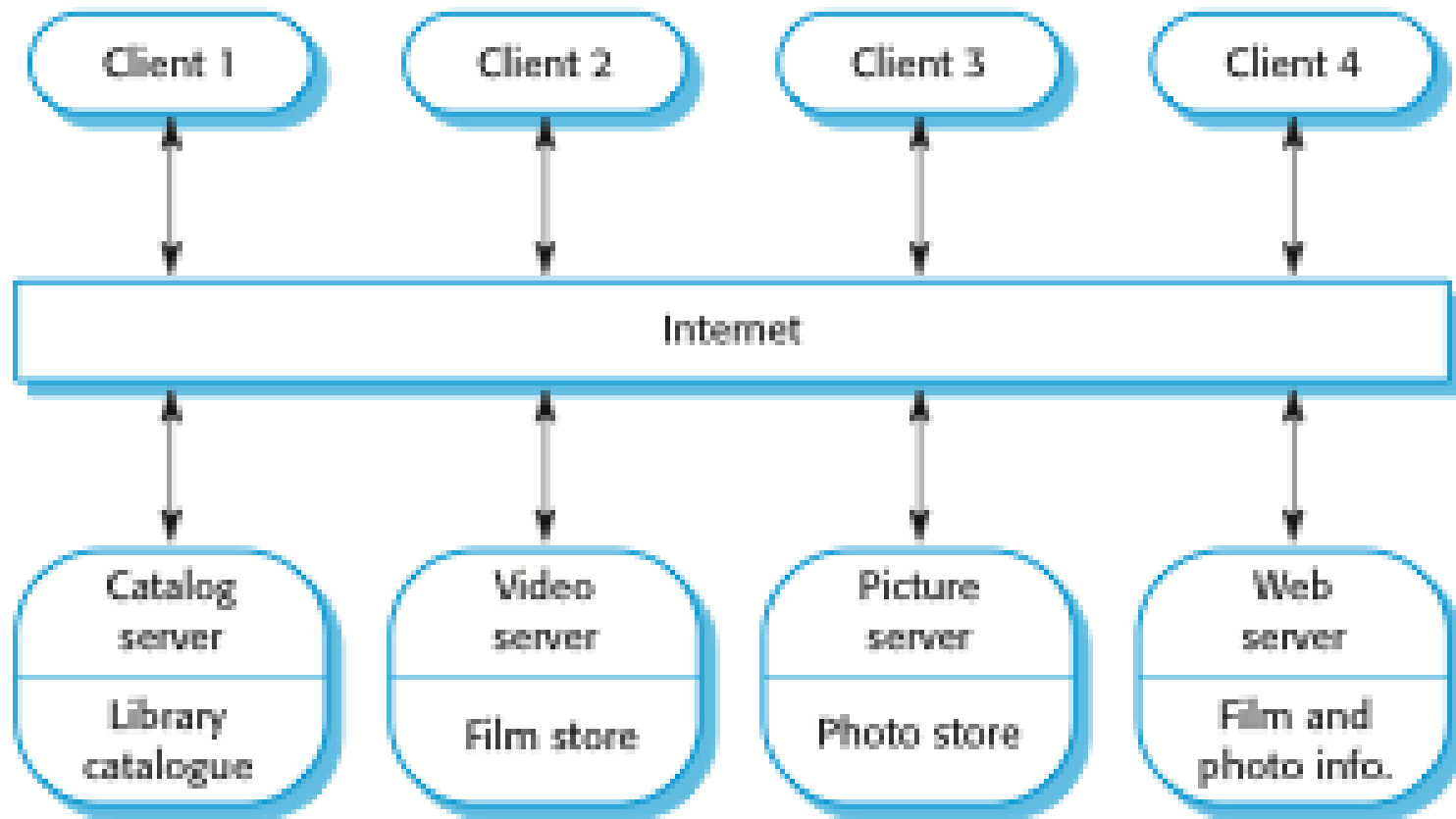
Client-server architecture

- 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.

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.

A client–server architecture for a film library



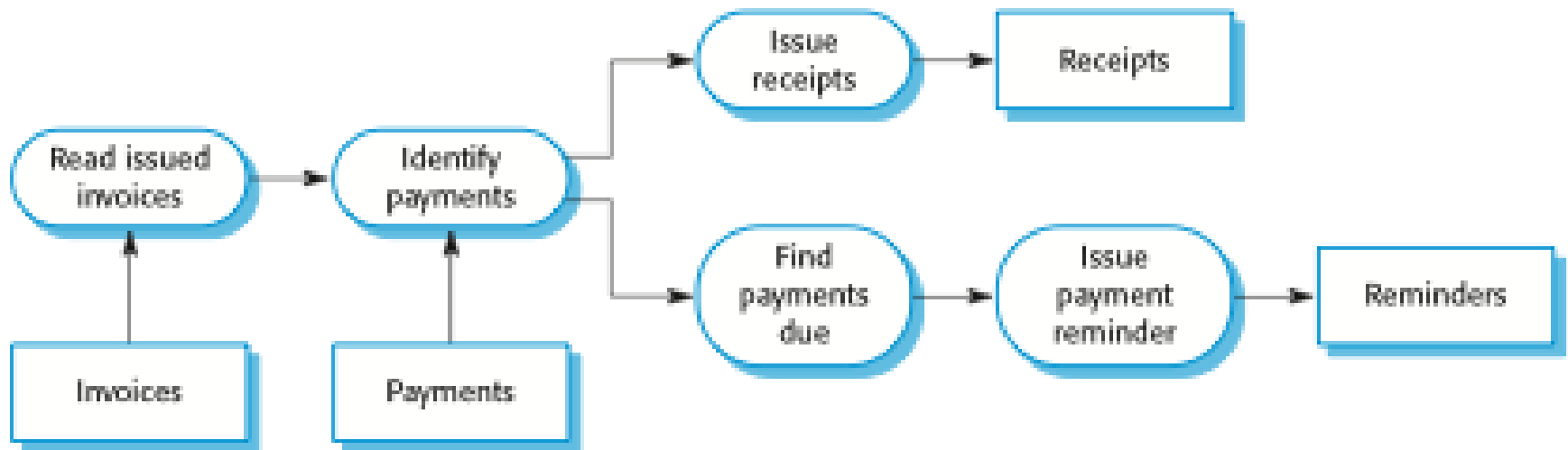
Pipe and filter architecture

- 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.

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.

An example of the pipe and filter architecture



Application architectures

- Application systems are designed to meet an organizational 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.

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.

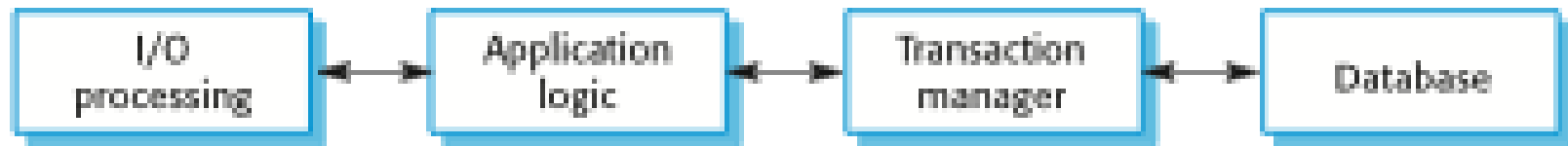
Application type examples

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

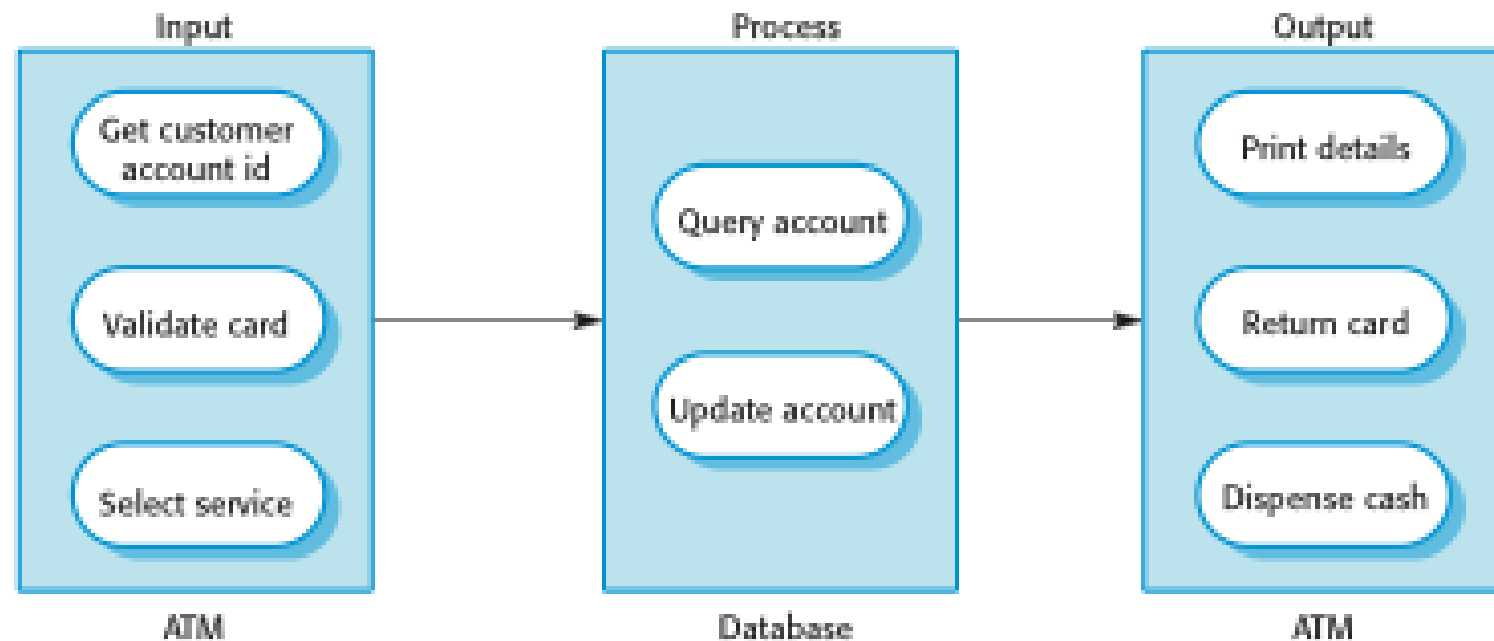
Transaction processing systems

- 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



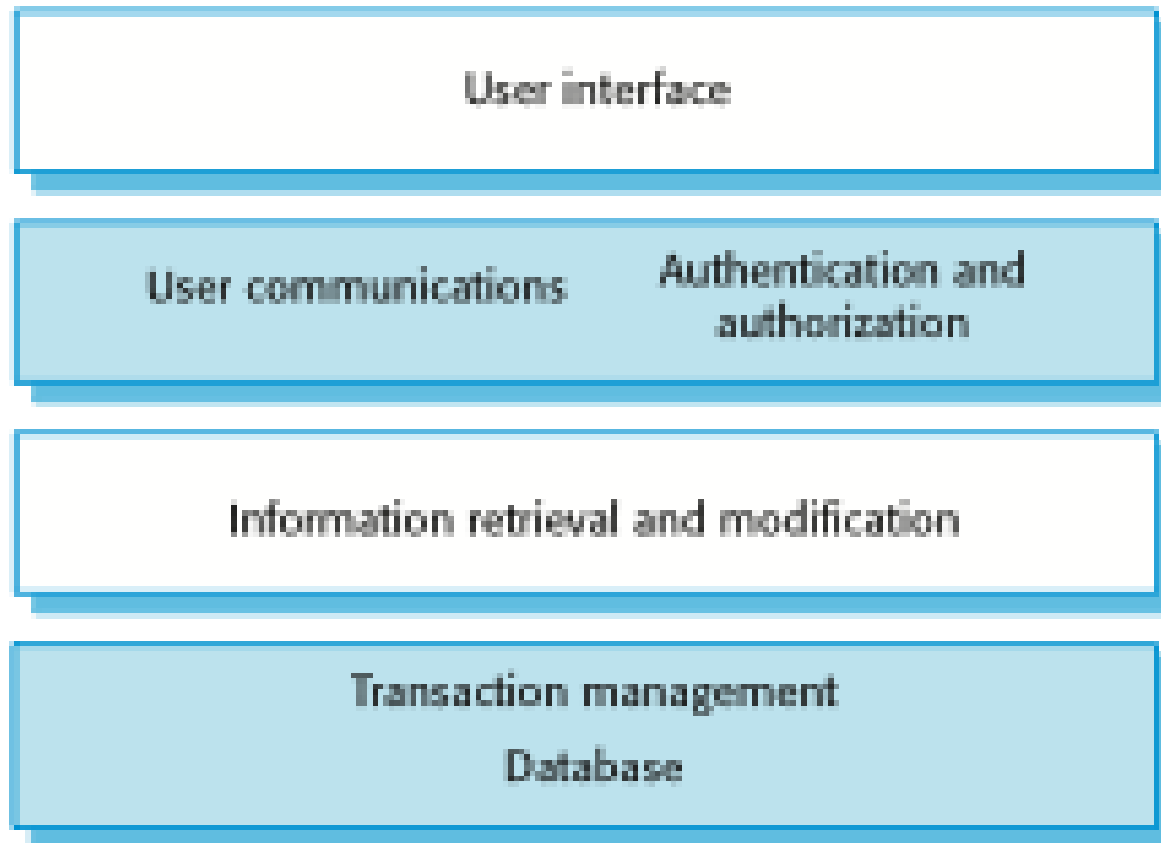
The software architecture of an ATM system



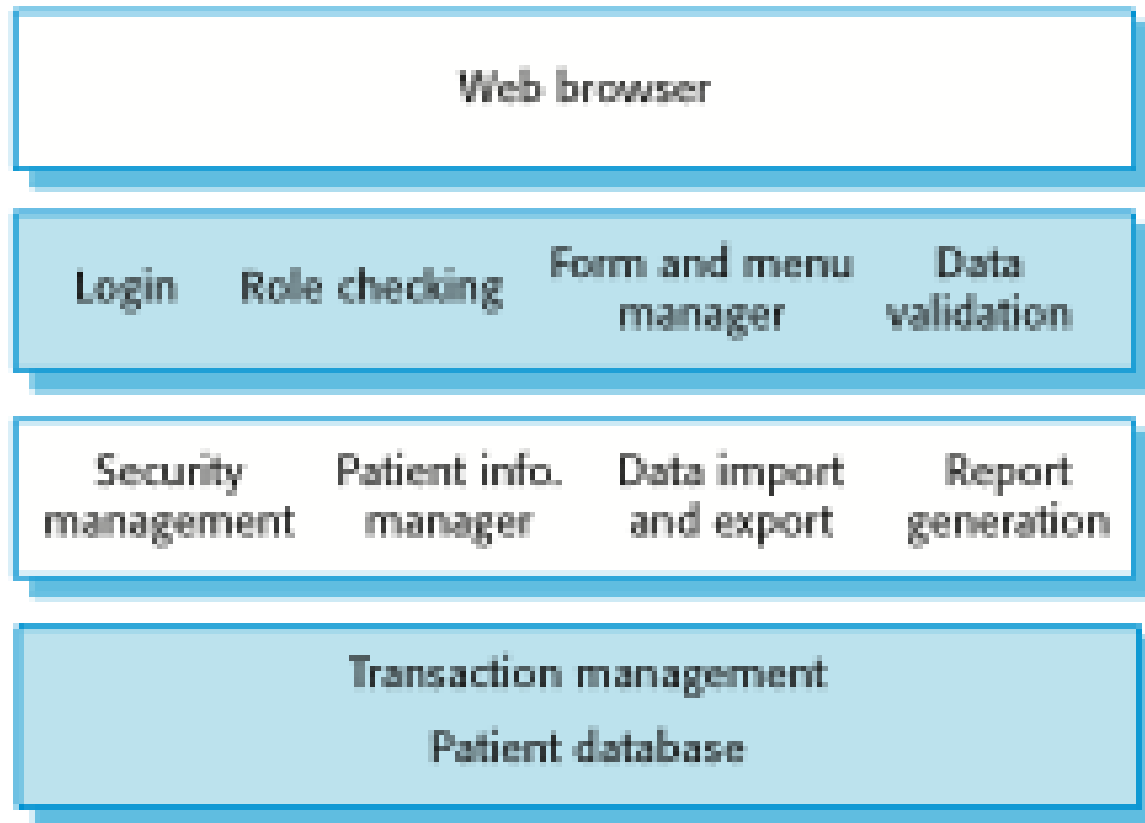
Information systems architecture

- Information systems have a generic architecture that can be organized 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 MHC-PMS



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.

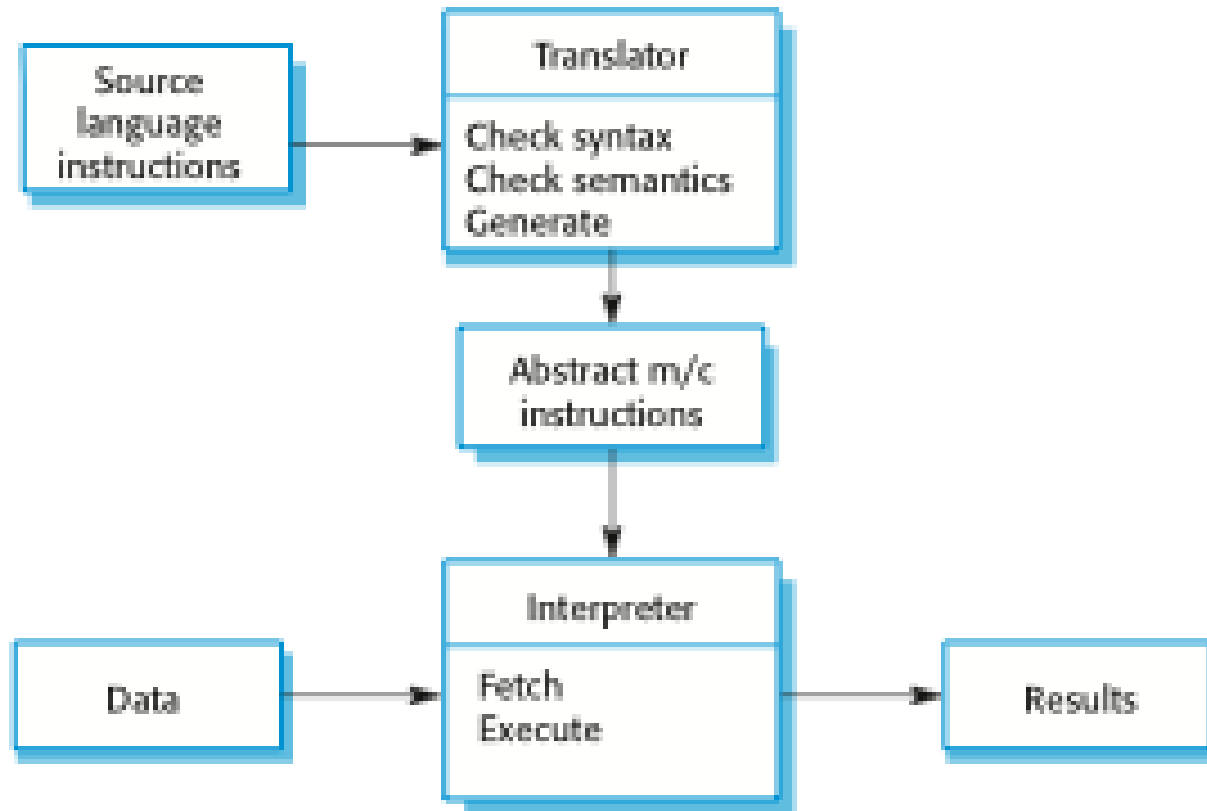
Server implementation

- 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.

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.

The architecture of a language processing system



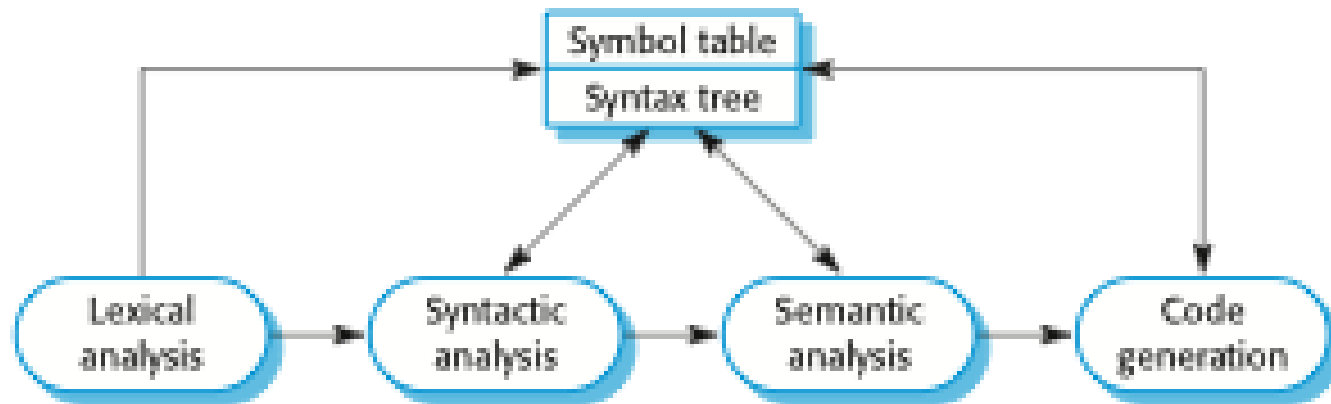
Compiler components

- 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.

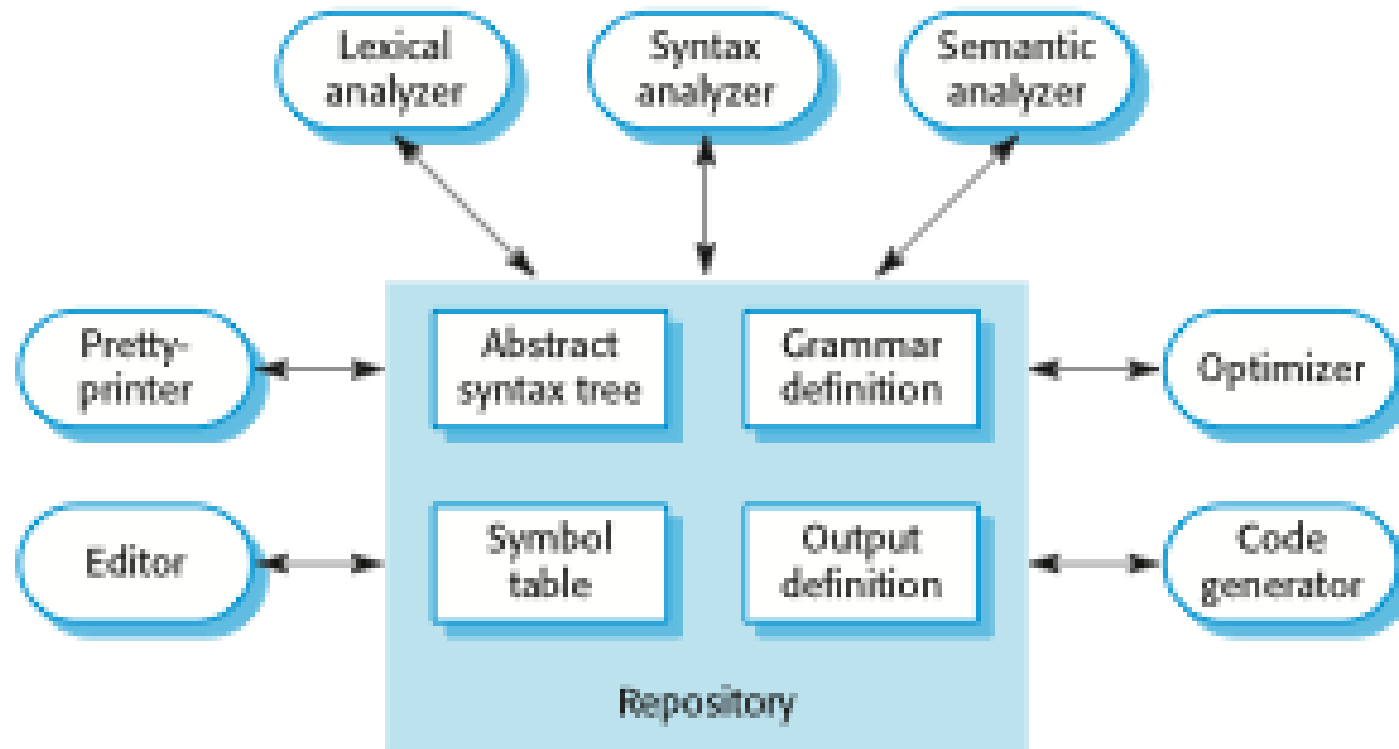
Compiler components

- 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 pipe and filter compiler architecture



A repository architecture for a language processing system



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.

References

- Software Engineering by Somerville, Ninth editor, chapter 6.