Architectural Design

Chapter 6

Architectural Design

- **Architectural Design:**
  the design process for identifying:
  - the subsystems making up a system and
  - the relationships between the subsystems

- what's a subsystem?
  - A subsystem is a sub-part of the system that provides (and/or consumes) services to other subsystems
- Architectural Design = subsystem decomposition
  - break the system into subparts with the goal of simplifying the overall system.

Software Architecture

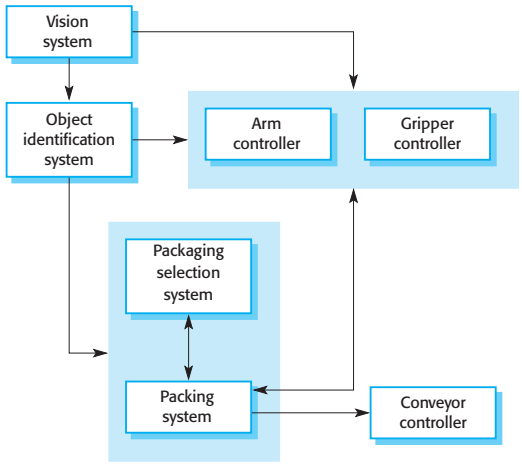
- **Software Architecture:**
  - a description of how a software system is organized (or decomposed)
  - an architectural model that is the output of architectural design
  - represents the subsystems and which ones communicate with each other

Architectural Models

- Simple box and line diagrams
- Each box is a component of the system (a subsystem)
- Boxes within boxes are subcomponents of a subsystem
- Arrows indicate data and/or messages are passed between components

Example: Architecture of a packing robot control system

The robot uses the vision system to pick out objects on a conveyor, identify the type of object, and select the right kind of packaging. It packs the object, and places it on another conveyor.



Architectural Design in the software engineering process

- Specification-Development-Verification-Evolution
- Development = Design+Implementation
- First step in Design is Architectural Design

- Critical link between requirements engineering and the design processes.
- In iterative development, system architecture is designed and implemented in the first iteration
  - Refactoring the overall structure is costly.

Use of Architectural Models

- Facilitating discussion about the system
  - For communication with stakeholders and project planners.
  - Use to discuss requirements with stakeholders
- Documenting the design of an architecture
  - Used as a basis for implementation, further design
  - Requires complete, detailed system model

6.1 Architectural Design Decisions

- Architectural Design is a creative process:
  - It is a series of decisions to be made.

Some issues to be considered:

- Will the system be standalone or distributed?
  - It may require having the subsystems distributed over different machines/processors.
  - This decision affects performance and other system-wide attributes (nonfunctional requirements).

Nonfunctional requirements affected by architecture

- To maximize Performance
  - Localize performance-critical operations within a few components on one processor
  - Minimize communications.
- To maximize security
  - Use a layered architecture with critical assets in the innermost layers (must be authorized to access layer).
- To maximize safety
  - Localize safety-critical features in a small number of sub-systems (simplifies validation).

Coupling and Cohesion

- Coupling: the number of dependencies (amount of communication) **between** two subsystems
- Cohesion: the number of dependencies **within** a subsystem (ie between classes in a subsystem).
- Goal: low coupling and high cohesion
  - subsystems should be independent, then modifications to one are unlikely to affect another.
  - subsystems should have internal units that are highly dependent on one another. These subsystems are easily understood and more reusable.

6.3 Architectural Patterns

- An architectural pattern is an abstract description of system organization that has been successful in previous projects (in various contexts)
- Patterns are a means of representing, sharing and reusing knowledge.
- Each pattern description should indicate in which contexts it is and is not useful.
- Architectural designers can browse pattern descriptions to identify potential candidates

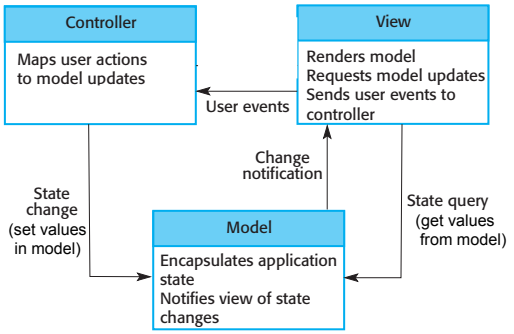
Model-View-Controller (MVC) Pattern

- Commonly used in desktop applications and web applications.
- Used to separate the data (the model) from the way it is presented to the user (the views)
- Model objects encapsulate the data.
- View objects present data to and receive actions from the user.
- Controller manages communication between Model and View (responds to user actions).

Model-View-Controller (MVC) Pattern Description

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. •**Model component** manages the system data and associated operations on that data. •**View component** defines and manages how the data is presented to the user. •**Controller component** manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to the View and the Model.
Example	Most web-based application systems, most desktop apps.
When 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.

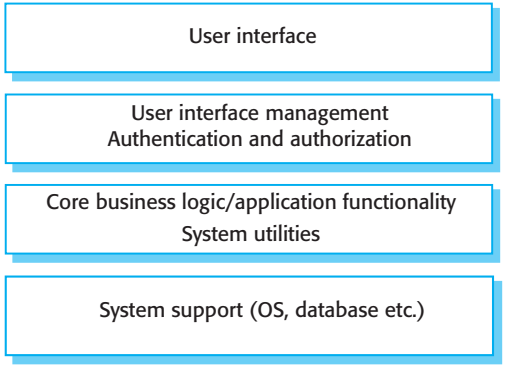
Model-View-Controller (MVC) Pattern Diagram



Layered Architecture Pattern

- System functionality is organized into separate layers.
- Each layer relies only on facilities and services of layer immediately beneath it.

Layered Architecture Pattern Diagram



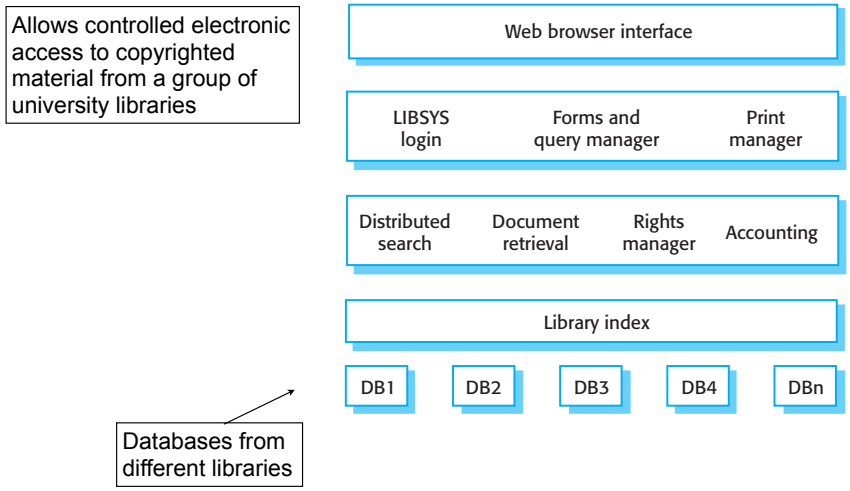
Layered Architecture Pattern Advantages

- Separation/independence: allows changes to be localized.
- Supports incremental development: as services are added to layers, expose them to the user.
- Changeability:
  - Easily replace one layer by another equivalent one (with same interface).
  - If interface changes, affects only layer above.
- Portability: need to change only bottom layer to port to different machine(s).

Layered Architecture Pattern Description

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.
Example	A layered model of a system for sharing copyright documents held in different libraries: LIBSYS
When used	Used when • building new facilities on top of existing systems • the development is spread across several teams with each team responsibility for a layer of functionality • 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.

Layered Architecture Pattern Example: LIBSYS



Repository Architecture

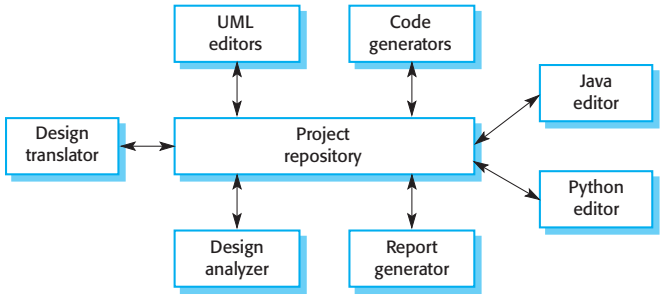
- Data is stored in a central shared repository.
- Components interact through the repository only.
- Suited to applications whose data is generated by one component and used by another.
- Advantages:
  - Components are independent/separate.
  - Changes to data are automatically available to other components.
- Communication between components may be inefficient.

Repository Architecture Description

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	An IDE where the components use a repository of system design information. Each software component generates information which is then available for use by other tools.
When used	•when large volumes of information are generated that has to be stored for a long time. •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.

Repository Architecture Example: IDE

Different “tools” supporting model-driven development



Client-Server Architecture

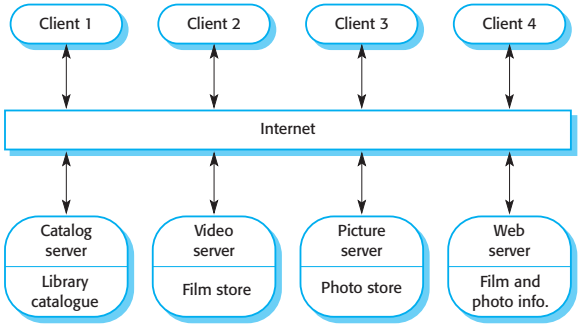
- Commonly used organization for distributed systems.
- Composed of:
  - A set of servers that offer specific (unique) services to other components.
  - A set of clients that call on services offered by the servers
  - A network that allows the clients to access the services.
- Could run on a single computer: separation/ independence.
- Clients make remote procedure calls to servers using a protocol like http, waits for reply.
- Several instances of client on different machines.

Client-Server Architecture Description

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	The 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 Architecture Example: Film Library

Serves information, videos, still photos. Catalog server handles searching. Clients are multiple instances of a user interface (in a web browser).



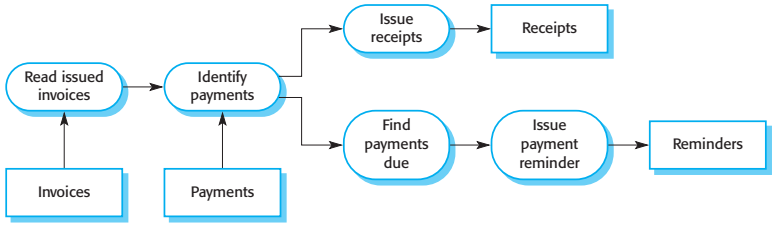
Pipe and Filter Architecture

- A series of transformations on data
- Composed of:
  - A set of "filters", each one transforming some input stream into an output stream.
  - Pipes connecting the filters.
- Data is transformed as it moves through the system.
- Transformations can be run concurrently.
- Commonly used in batch processing systems and embedded control systems.
- Difficult to use for interactive systems.

Pipe and Filter Architecture Description

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	The 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.

Pipe and Filter Architecture Example: Processing invoices



Once a week, payments are reconciled against invoices (issued at the beginning of the month). For paid invoices, it issues a receipt. For unpaid, it issues a reminder.

6.4 Application Architectures

- Systems in the same domain often have similar architectures that reflect domain concepts.
  - data collection systems
  - monitoring systems
  - billing systems
  - supply chain management
  - compilers
  - etc.
- If application reuse (COTS) is not possible, it may be possible to re-use the architecture.

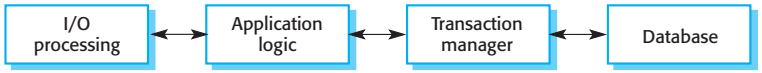
Transaction Processing Systems

- Database centered applications that
  - process user requests for information and
  - update information in a system database.
  - Prevent users actions from interfering with each other.
  - Preserve integrity of the database
  - Examples:
    - E-commerce systems
    - Reservation systems.

Transaction Processing Systems

- Process user requests for information from a database or requests to update the database.
- Transaction: sequence of operations treated as a single unit.
  - when all operations are done, they are made permanent
  - failure must not put database in inconsistent state.
- Example: transfer money between accounts
  - Really two operations: debit one account, credit the other.
  - If one succeeds but the other does not, the database is in an inconsistent state (the bank's books will not balance)

Transaction Processing Systems: Architecture



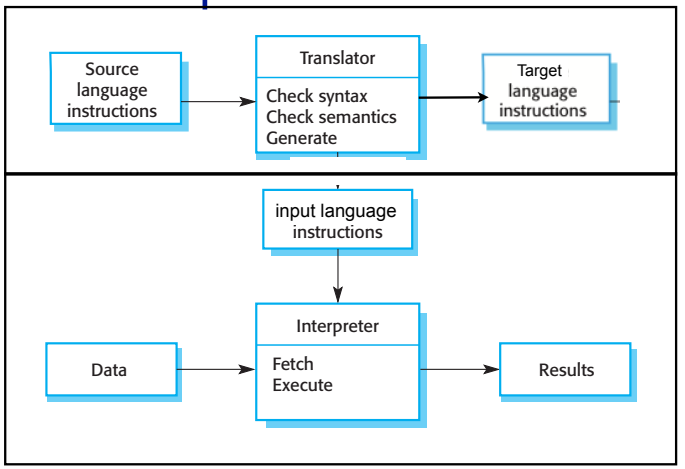
A simple layered architecture (this one is drawn sideways)

- User makes request through I/O processing
- Request is processed by application logic, creates a transaction.
- Transaction manager communicates with Database, makes sure transaction is completed as a unit.
- Result is passed back through to the user.

Language Processing Systems

- Process instructions in a given language.
  - natural or artificial language
- **Translators**
  - convert instructions in one language to another language
- **Interpreters**
  - execute instructions in a given language
- Examples
  - compilers: g++, javac
  - interpreters: sql evaluation, JVM (java)
  - browser: html, xml
  - simulator: iphone (for testing apps on mac)

Language Processing Systems Simple architectures



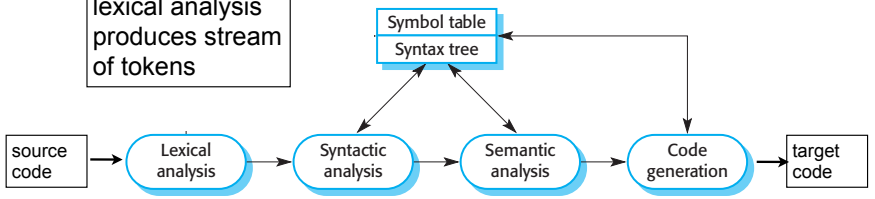
Language Processing Systems components

- Lexical analyzer (scanner), groups characters into tokens (identifiers, words, operators, numbers, etc.)
- Syntax analyzer (parser), groups tokens into phrases, sentences, etc. (produces syntax tree)
- Syntax tree, an internal structure representing the parsed input stream.
- Symbol table, holds info about the names of entities (variables, functions, objects,...) used in the text.
- Semantic analyzer: checks the semantic correctness of the input text (type checking)
- Code generator: 'walks' (traverses) the syntax tree and generates text in target language.

Language Processing Systems Pipe and Filter

Good for batch processing, run from the command line.

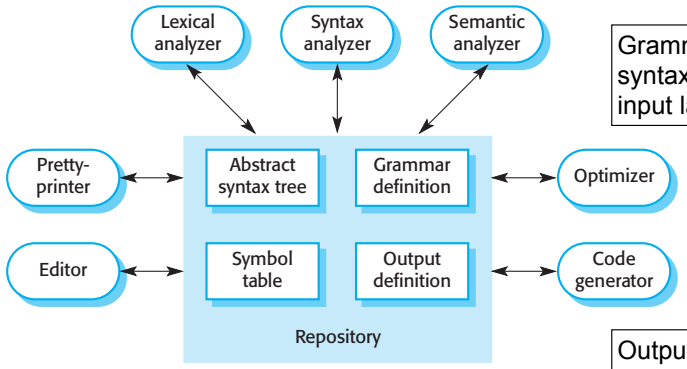
lexical analysis produces stream of tokens



Syntax tree is output of Syntactic analysis, which is transformed by Semantic analysis

Language Processing Systems Repository

Good for interactive, combined system, IDE



Grammar definition: syntax rules for input language

Output definition: syntax rules for output language