Architectural Design

Chapter 6

Outline:

• Definitions
• Models

• Architectural Patterns
  1. MVC: Model-View-Controller
  2. Layered (or tiered)
  3. Pipe and Filter
  4. Repository
  5. Client-Server

Software Design

• Process of converting the requirements into the design of the system.
• Definition of how the software is to be structured or organized.

• For large systems, this is divided into two parts:
  - Architectural design defines main components of the system and how they interact.
  - Detailed design components are decomposed and described at a much finer level of detail.

What is a component?

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
  - subsystem = component of the system.

• Architectural Design = subsystem decomposition
  - break the system into subparts with the goal of simplifying the overall system.
Services and subsystem interfaces

- **A service** is a set of related operations that share a common purpose.

- **Subsystem interface**: the set of one subsystems’ operations that are available to other subsystems.
  - specification includes the signature of each operation: name of the operation, types of each parameter.

- Why specify interfaces to components?
  - so components may be developed in parallel

- Helps promote independence (separation).
  - can make changes behind the interface without affecting components using that interface.

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 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

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.
  - Architectural designers can browse pattern descriptions to identify potential candidates for their project

- Each pattern description should indicate in which contexts it is and is not useful.

- Design goals: **simplicity** (reduce complexity) and **independence** (reduce dependence).
1. Model-View-Controller (MVC) Pattern

- Commonly used in desktop, mobile phone, 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 Responds to user actions (from View) by updating Model (and View).

Model-View-Controller (MVC) Pattern Diagram

2. Layered Architecture Pattern

- System functionality is organized into separate layers.
- Each layer relies only on facilities and services of layer immediately beneath it.
- Advantages:
  - Changes to one layer do not affect layers above.
  - If interface changes, affects only layer above.
  - Easily replace one layer by another equivalent one (with same interface).
  - Portability: need to change only bottom layer to port to different machine(s).
Layered Architecture Pattern Description

<table>
<thead>
<tr>
<th>Name</th>
<th>Layered architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>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.</td>
</tr>
<tr>
<td>Example</td>
<td>A layered model of a system for sharing copyright documents held in different libraries: LIBSYS</td>
</tr>
<tr>
<td>When used</td>
<td>Used when</td>
</tr>
<tr>
<td></td>
<td>* building new facilities on top of existing systems</td>
</tr>
<tr>
<td></td>
<td>* the development is spread across several teams with each team responsibility for a layer of functionality</td>
</tr>
<tr>
<td></td>
<td>* there is a requirement for multi-level security.</td>
</tr>
<tr>
<td>Advantages</td>
<td>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.</td>
</tr>
<tr>
<td>Disadvantages</td>
<td>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.</td>
</tr>
</tbody>
</table>

3. 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
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.

Pipe and Filter Example: Compiler (g++)

Pipe and Filter Architecture
Description

<table>
<thead>
<tr>
<th>Name</th>
<th>Pipe and filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>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.</td>
</tr>
<tr>
<td>Example</td>
<td>The pipe and filter system used for processing invoices.</td>
</tr>
<tr>
<td>When used</td>
<td>Commonly used in data processing applications (both batch- and transaction-based) where inputs are processed in separate stages to generate related outputs.</td>
</tr>
<tr>
<td>Advantages</td>
<td>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.</td>
</tr>
<tr>
<td>Disadvantages</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

- 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.
5. 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.
- Clients make remote procedure calls to servers using a protocol like http, then wait for reply.
- Several instances of client on different machines.
# Client-Server Architecture

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>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.</td>
</tr>
<tr>
<td>Example</td>
<td>The film and video/DVD library organized as a client–server system.</td>
</tr>
<tr>
<td>When used</td>
<td>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.</td>
</tr>
<tr>
<td>Advantages</td>
<td>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.</td>
</tr>
<tr>
<td>Disadvantages</td>
<td>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.</td>
</tr>
</tbody>
</table>