Chapter 6: Architectural Design
Objectives

- Understand why architectural styles are important.
- Understand the decisions that have to be made about the system architecture.
- Understand different architectural styles.
Overview

Part I. Architectural Design

- System organisation
  - Repository model
  - Client/server model
  - Layered model
- Decomposition style
  - Object oriented
  - Function oriented (pipeline)
- Control style
  - Centralised
  - Event-driven

Part II. Generic application architectures
The software design process is composed of four main design activities:

- Architectural design → this chapter
- Data design
- Component design
- Interface design → discussed in chapter 7

Covered in SA course
The main deliverables of this phase are:

- Software architecture
- Data structures
- Pseudo code
- Interface designs

Software Design Document
Design covered in SA course

- **Data Design**
  - Translate ER diagrams and data dictionary created during requirements analysis into data structures or database schema.

- **Component design**
  - Data structures and algorithms must be represented in sufficient detail to guide in the generation of programming language source code.
  - Use structured English (pseudo code), and flowcharts
Software Architecture

Requirements

Design
What is architectural design?

- An early stage of the system design process.
- Link between the *specification* and *design* processes.
- It involves identifying major system components and their communications.
- The output of the architectural design process is a description of the *software architecture*.
Why do we need architectural design?

- **Stakeholder communication**
  Architecture may be used as a focus of discussion by system stakeholders.

- **System analysis**
  Architectural design decisions have a profound effect on whether the system can meet the critical requirements (performance, etc).

- **Large-scale reuse**
  The architecture may be reusable across a range of systems.
## Software architecture and non-functional requirements

<table>
<thead>
<tr>
<th>Non-functional Requirement</th>
<th>Architectural considerations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>Localize critical operations and minimize communications. Use large rather than fine-grain components.</td>
</tr>
<tr>
<td>Security</td>
<td>Use a layered architecture with critical assets in the inner layers.</td>
</tr>
<tr>
<td>Safety</td>
<td>Localize safety-critical features in a small number of sub-systems.</td>
</tr>
<tr>
<td>Availability</td>
<td>Include redundant components and mechanisms for fault tolerance.</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Use fine-grain, replaceable components.</td>
</tr>
</tbody>
</table>
We can have more than one architectural style within a single software system.

Do you agree?
The architectural model of a system may conform to a generic architectural model or style. An awareness of these styles can simplify the problem of defining system architectures. However, most large systems are heterogeneous and do not follow a single architectural style.
Architectural design decisions

- The software architect must make decisions about:
  - System organisation (structure) style
  - Decomposition style
  - Control style

- Evaluating different architectures for a software is sometimes refereed to as *Trade-off analysis*. 
Architectural design decisions

The software architect should answer the following...

- Is there a generic application architecture that can be used?
- What architectural styles are appropriate?
- 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?
1. System organisation

Architectural design decisions
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 client/server style;
3. An abstract machine or layered style.
System organisation

1. Shared repository style

- This style is used when large amounts of data are to be shared.
- Shared data is held in a central database or repository and may be accessed by all sub-systems.
System organisation

1. Shared repository style

- CASE toolset architecture
System organisation

1. Shared repository style

Advantages

- 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.

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.
System organisation

2. Client/Server style

Distributed system model, which contains:

1. A set of stand-alone servers which provide specific services such as printing, data management, etc.

2. A set of clients which call on these services.

3. A network which allows clients to access servers.
System organisation

2. Client/Server style

Film and picture library:

- Client 1
- Client 2
- Client 3
- Client 4

- Catalogue server
- Library catalogue
- Video server
- Film clips files
- Picture server
- Digitised photographs
- Webserv
- Film and photo info.
System organisation

2. Client/Server style

Advantages

- 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;
- No central register of names and services - it may be hard to find out what servers and services are available.
System organisation

3. Abstract machine / layered style

- Used to model the interfacing of sub-systems.
- Organises the system into a set of layers (or abstract machines)
- Each layer provides a set of services to the layer above and serves as a client to the layer below.
- Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.
3. Abstract machine / layered style

Layered Architecture

- Usually procedure calls
- Users
- Composites of various elements
- Core Level
- Basic Utility
- Useful Systems
System organisation

3. Abstract machine / layered style

- Communication protocols (OSI reference model for network protocols).
- Operating systems
- Database Management systems (storage, transaction, query, optimizer, etc)

Information Systems

- User Interface
- User Communications
- Information Retrieval
- Transaction Management Database
3. Abstract machine / layered style

Advantages

- Supports incremental development
- Changeable (if a layer changes, only adjacent layers are affected)

Disadvantages

- Structuring systems into layers is difficult
- Inner layers may provides facilities required by all layers (e.g. file management)
- Performance is degraded.
Architectural design decisions

- The software architect must make decisions about:
  - System organisation (structure) style
  - **Decomposition style**
  - Control style
2. Decomposition style

Architectural design decisions
Decomposition style

- Sub-systems are decomposed into modules.
- Two modular decomposition models covered
  1. An **object** model where the system is decomposed into interacting objects;
  2. A **pipeline** or data-flow model where the system is decomposed into functional modules which transform inputs to outputs.
Decomposition style

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.
Decomposition style

1. Object models

- Invoice processing system

<table>
<thead>
<tr>
<th>Customer</th>
<th>Receipt</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer#</td>
<td>invoice#</td>
</tr>
<tr>
<td>name</td>
<td>date</td>
</tr>
<tr>
<td>address</td>
<td>amount</td>
</tr>
<tr>
<td>credit period</td>
<td>customer#</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Payment</th>
<th>Invoice</th>
</tr>
</thead>
<tbody>
<tr>
<td>invoice#</td>
<td>invoice#</td>
</tr>
<tr>
<td>date</td>
<td>date</td>
</tr>
<tr>
<td>amount</td>
<td>amount</td>
</tr>
<tr>
<td>customer#</td>
<td>customer</td>
</tr>
</tbody>
</table>
1. Object models

Advantages

- 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.
2. Pipeline models

- Functional transformations process their inputs to produce outputs.
- *Pipe and filter* model (as in UNIX shell).
- When transformations are sequential, this is a batch sequential model which is extensively used in data processing systems.
- Not really suitable for interactive systems.
Decomposition style

2. Pipeline models

- Invoice processing system
2. Pipeline models

Advantages

- Supports reuse.
- Intuitive organization. Many people think of their work in terms of input and output processing.
- Easy to add new transformations.
- Relatively simple to implement as either a concurrent or sequential system.

Disadvantages

- Requires a common format for data transfer along the pipeline and difficult to support event-based interaction.
Architectural design decisions

- The software architect must make decisions about:
  - System organisation (structure) style
  - Decomposition style
  - Control style
Architectural design decisions

3. Control style
3. Control styles

- Are concerned with the control flow between sub-systems. Distinct from the system decomposition model.

- **Centralised** control
  - One sub-system has overall responsibility for control and starts and stops other sub-systems.

- **Event-Driven** control
  - Each sub-system can respond to externally generated events from other sub-systems or the system’s environment.
Control style

1. Centralised Control

- A control sub-system takes responsibility for managing the execution of other sub-systems.

**Call-Return Models**
- Top-down subroutine model.
- Control starts at the top of a subroutine hierarchy and moves downwards.
- Applicable to sequential systems

**Manager Models**
- Applicable to concurrent systems.
- One system component controls the stopping, starting and coordination of other system processes.
Control style

1. Centralised Control

🔹 Call-Return Model
Control style

1. Centralised Control

✧ Manager Model

Diagram showing the Manager Model with connections to Sensor processes, Actuator processes, System controller, User interface, and Fault handler.
Control style

2. Event-Driven Control

- Driven by externally generated events where the timing of the event is outside the control of the sub-systems which process the event. (event vs. input)

**Broadcast** Models

- An event is broadcast to all sub-systems. Any sub-system which can handle the event may do so

**Interrupt-driven** Models

- Used in real-time systems where interrupts are detected by an interrupt handler and passed to some other component for processing.
Control style

2. Event-Driven Control

- **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.
Control style

2. Event-Driven Control

- **Interrupt-driven model**
  - 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 with a memory location and a hardware switch causes transfer to its handler.
  - Allows fast response but complex to program and difficult to validate.
Control style

2. Event-Driven Control

- **Interrupt-driven model**
Part II. Generic Application Architecture

- Chapter 12 in textbook
  - Distributed systems architectures
- Chapter 13 in textbook
  - Application specific architectures
    - Data processing
    - Transaction processing
    - Event processing
    - Language processing