Chapter 6: Architectural Design
Key Takeaway Points

- The software architecture of a system or subsystem refers to the style of design of the structure of the system including the interfacing and interaction among its subsystems and components.

- Different types of systems require different design methods and architectural styles.

- Guidelines for Architectural Design
  1. Adapt an architectural style when possible.
  2. Apply software design principles.
  3. Apply design patterns.
  4. Check against design objectives and design principles.
  5. Iterate the steps if needed.
Architectural Design Process

1. Determine Design Objectives
2. Determine Type of System
3. Types of System & Characteristics
4. Architectural Styles Repository
5. Apply an Architectural Style
6. Specify Subsystem Functions, Interfaces & Interaction Behavior
7. Perform Custom Architectural Design
8. Review the Architectural Design

Partial design feedback
Architectural Design Considerations

- Ease of change and maintenance.
- Use of commercial off-the-shelf (COTS) parts.
- System performance – does the system require to process real-time data or a huge volume of transactions?
- Reliability.
- Security.
- Software fault tolerance.
- Recovery.
Four Common Types of Systems

(a) Interactive subsystem

(b) Event-driven subsystem

(c) Transformational subsystem

(d) Database subsystem
Characteristics of Interactive Systems

• The interaction between system and actor consists of a relatively fixed sequence of actor requests and system responses.
• The system has to process and respond to each request.
• Often, the system interacts with only one actor during the process of a use case.
• The actor is often a human being although it can also be a device or another subsystem.
• The interaction begins and ends with the actor.
• The actor and the system exhibit a “client-server” relationship.
• System state reflects the progress of the business process represented by the use case.
Characteristics of Event-Driven Systems

• It receives events from, and controls external entities.
• It does not have a fixed sequence of incoming requests; requests arrive at the system randomly.
• It does not need to respond to every incoming event. Its response is state dependent—the same event may result in different responses depending on system state.
• It interacts with more than one external entity at the same time.
• External entities are often hardware devices or software components rather than human beings.
• Its state may not reflect the progress of a computation.
• It may need to meet timing constraints, temporal constraints, and timed temporal constraints.
Characteristics of Transformational Systems

• Transformational systems consist of a network of information-processing activities, transforming activity input to activity output.
• Activities may involve control flows that exhibit sequencing, conditional branching, parallel threads, synchronous and asynchronous behavior.
• During the transformation of the input into the output, there is little or no interaction between system and actor—it is a batch process.
• Transformational systems are usually stateless.
• Transformational systems may perform number crunching or computation intensive algorithms.
• The actors can be human beings, devices, or other systems.
Characteristics of Object-Persistence Systems

• It provides object storage and retrieval capabilities to other subsystems.
• It hides the implementation from the rest of the system.
• It is responsible only for storing and retrieving objects, and does little or no business processing except performance considerations.
• It is capable of efficient storage, retrieval, and updating of a huge amount of structured and complex data.
Class Discussion

• Find examples of
  – interactive systems
  – event-driven systems
  – transformational systems, and
  – object-persistence systems

• Show that the example systems possess the properties listed on previous slides, respectively.

• Why do different types of systems require different design methods?
# System Types and Architectural Styles

<table>
<thead>
<tr>
<th>Type of System</th>
<th>Architectural Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interactive System</td>
<td>N-Tier</td>
</tr>
<tr>
<td>Event-Driven System</td>
<td>Event-Driven</td>
</tr>
<tr>
<td>Transformational System</td>
<td>Main Program and Subroutines</td>
</tr>
<tr>
<td>Object-Persistence Subsystem</td>
<td>Persistence Framework</td>
</tr>
<tr>
<td>Client-server</td>
<td>Client-server</td>
</tr>
<tr>
<td>Distributed, decentralized</td>
<td>Peer-to-peer</td>
</tr>
<tr>
<td>Heuristic problem-solving</td>
<td>Blackboard</td>
</tr>
</tbody>
</table>
N-Tier Architecture

Graphical User Interface Layer
Controller Layer
Business Objects Layer
Database Layer
Network Communication Layer

requests
requests
requests
requests
requests
Client-Server Architecture

<<client>>
Passenger Check-In
Client 1

<<remote procedure call>>

<<remote procedure call>>

<<remote procedure call>>

<<client>>
Passenger Check-In
Client 2

<<remote procedure call>>

<<remote procedure call>>

<<remote procedure call>>

<<client>>
Passenger Check-In
Client 3

<<server>>
Flight Information System

UML notation for a software component

Input/output port

UML stereotype for a user-defined modeling construct.
Main Program and Subroutine Architecture

subroutine 1

subroutine 2

subroutine 3

subroutine 4

subroutine 5

subroutine 6

subroutine 7
Event-Driven Architecture

Component 1

Component 2

State Based Controller

Component 3

Component 4
Object-Persistence Framework

**DB Manager**

It is responsible for storing and retrieving objects from different databases. It hides the different databases from the business objects.

- Business Object A
- Business Object B
- Business Object C

Communicate in the object-oriented implementation language

- DB Access 1
- DB Access 2
- DB Access 3

Communicate in the object-oriented implementation language

- DB 1
- DB 2
- DB 3

Communicate in a DBMS specific language
Perform Custom Architectural Design

• **Remember**: Not all application systems development projects can reuse an existing architectural style.

• Custom architectural design may be required to meet the needs of a *specific* system.

• *Design patterns* and *COTS products* are useful for custom architectural design.
Architectural Style and Package Diagram

Package diagram for an N-tier architectural view

Legend: + public - private
Applying Software Design Principles

• Design for Change – design with a “built-in mechanism” to adapt to, or facilitate anticipated changes.

• Separation of Concerns – focusing on one aspect of the problem in isolation rather than tackling all aspects simultaneously.

• Information Hiding – shielding implementation detail of a module to reduce its change impact to other parts of the software system.
Applying Software Design Principles

• High Cohesion – achieving a higher degree of relevance of the functions of a module to the module’s core functionality.

• Low Coupling – reducing the run-time effect and change impact of a subsystem to other subsystems.

• KISS: Keep It Simple/Stupid – designing “stupid objects.”