What is a Pattern?

- Pattern
  - Describes a recurring design problem
  - Arises in specific design contexts (i.e., situations)
  - Presents a well proven approach for its solution
- Micro-architecture (Gamma et al.)
  - Small number of collaborating objects that may be reused
- Design New Software Architectures using existing patterns
Software Architectural Patterns

- Software Architectural Patterns [Buschmann, Shaw]
  - Recurring architectures used in various software applications
- Goal: Design Software Architecture from
  - Software Architectural Patterns
- Architectural Structure Patterns
  - Address structure of major subsystems
- Architectural Communication Patterns
  - Reusable interaction sequences between components

Architectural Structure Patterns

- Layered patterns
  - Layers of Abstraction
- Client/Service patterns
  - Multiple Client / Single Service
  - Multiple Client / Multiple Service
  - Multi-tier Client / Service
- Control Patterns
  - Centralized Control
  - Distributed Control
  - Hierarchical Control
Layers of Abstraction Pattern

- Structure system into hierarchical levels
- Each layer provides services for higher layers
- Layers of Abstraction in RT systems (Figs. 11.1, 11.2, 11.3)
  - Allows use of subsets and extensions
  - Lower layers do not depend on upper layers
  - Higher layers depend on lower layers
- Variations in Layers of Abstraction
  - Strict Hierarchy (Fig. 11.1)
  - Flexible Hierarchy (Fig. 11.3)
- Incorporate other patterns into Layered Pattern
  - Client/Service pattern
  - Control Patterns

Figure 11.1 Five layers of Internet (TCP/IP) reference model
Architectural Structure Patterns

- Kernel or Microkernel (Fig. 11.4)
  - Separates minimal functional core from extended functionality and customer specific parts
  - Most commonly used in operating systems
  - Can be used as lowest layer in layered architecture pattern
- Kernel Pattern with Layered Architecture
  - Kernel of system is determined
    - Always at lowest layer of hierarchy
  - Used in many operating systems
Kernel Pattern

Figure 11.4 Example of kernel pattern within layered architecture

Layer 1
- Operating System Kernel Layer

Layer 2
- Operating System Services Layer

Layer 3
- Applications Layer

Control Patterns for RT Software Architectures

- Centralized Control Pattern
  - One control component
    - Executes state machine
  - Receives sensor input from input device interface components
  - Controls external environment via output device interface components that output to actuators
  - Can use entity objects to store data that needs to be stored
- Examples
  - Cruise Control System
  - Microwave Oven Control System (Fig. 11.5)
Variations on Centralized Control Pattern

- Centralized control with single state machine
- Centralized control with interacting state machines
- Distributed independent control with unidirectional communication to service (Fig. 11.7)
Distributed Collaborative Control Pattern
for RT Software Architectures

- Several control components
- Each component
  - Controls part of system
  - Executes state machine
- Control is distributed among the components
  - Components communicate with each other to provide overall control
  - Peer-to-peer asynchronous message communication
    • Communicating state machines
- Example
  - High Volume Manufacturing System
Distributed Collaborative Control Pattern

Hierarchical Control Pattern for RT Software Architectures

- Several distributed control components
- Each control component
  - Controls part of system
  - Executes state machine
- Coordinator (Hierarchical Controller) component
  - Coordinates several distributed controller components
    - Provides high level control
    - Sends commands to each control component
    - Determines next job for each control component
- Example - Figure 11.8
- Frequently used in factory automation systems
**Figure 11.8 Hierarchical Control Pattern**

- **«control» «component»**: Distributed Controller
- **«output» «component»**: ActuatorCmpt
- **«input» «component»**: SensorCmpt

**Master-Slave Pattern**

- Master sends command to Slave
- Slave sends completion acknowledgement to Master
- Master can divide a computational problem into parts
- Sends each part to a slave
- Receives responses
- Integrates responses
- Example of Master-Slave Pattern (Fig. 11.9)
Figure 11.9 Master-Slave Pattern

Client/Service Patterns

- **Client** requests services
- **Service** is provider of services for clients
- Client depends on service
- Client at higher layer than service in layered architecture
- Real-time clients can use service
  - During initialization
  - Update real-time status
- Variations:
  - Single Client / Multiple Service (Fig. 11.11)
  - Multiple Client / Multiple Service (Fig. 11.13)
    - Clients communicate with multiple services
Figure 11.11 Example of Multiple Client / Single Service Pattern

![Diagram of banking system](image1)

Figure 11.12 Example of client and service subsystems in Emergency Monitoring System

![Diagram of emergency monitoring system](image2)

Figure 11.13 Examples of Multiple Client / Multiple Service Pattern

![Diagram of monitoring and alarm services](image3)
Architectural Communication Patterns

- Asynchronous communication patterns
- Synchronous communication patterns

- Broker Communication Patterns
  - Broker forwarding
  - Broker handle
  - Discovery

- Group Communication Patterns
  - Broadcast
  - Subscription/notification

- Broker and group communication patterns
  - Facilitate software evolution and adaptation

UML notation for messages

- Simple message: No decision yet made about message type
- Asynchronous (loosely coupled) message communication: UML 1.3
- Synchronous (tightly coupled) message communication with reply
  - Option 1:
  - Option 2:
Synchronized Call/Return Pattern

- Synchronization of Tasks Interacting via Passive Objects
- Task interaction via shared data
  - Needs synchronization
- Task interaction via passive data abstraction object
  - Hides structure of data repository
  - Hides synchronization from tasks
    - Mutual exclusion
    - Multiple readers / multiple writers

Figure 11.14 Example of Synchronized Call/Return Pattern
Asynchronous Message Communication Pattern

- Producer sends message and continues
  - Consumer receives message
    - Suspended if no message is present
    - Activated when message arrives
  - Message queue may build up at Consumer

Figure 11.15 Asynchronous message communication pattern

Many-to-one Asynchronous Communication (Fig. 11.16)

- Several producers send messages to one consumer
- FIFO queue at consumer
Bi-directional Asynchronous Communication Pattern

- Producer sends asynchronous message and continues
- Consumer receives message
- Consumer generates and sends asynchronous response
- Message queue can build up at Consumer
- Response queue can build up at Producer
Example of Bidirectional asynchronous message communication pattern

Synchronous Message Communication with Reply Pattern (Client/Service Scenario)

- Service
  - Responds to message requests from several clients
- Client
  - Sends message to Service and Waits for response
- Remote Procedure Call
  - Client makes RPC to service on different node
  - Communication details hidden from client & service
- Remote method invocation (RMI)
  - Client object sends message to service object
**Synchronous Message Communication With Reply Pattern**  
*(Producer/Consumer Scenario)*

- Producer task sends message and waits for reply
- Consumer receives message
  - Suspended if no message is present
  - Activated when message arrives
  - Generates and sends reply
- Producer and Consumer continue

---

**Asynchronous communication with Callback Pattern**

- RT software architecture
  - Asynchronous message communication often preferred
- Send message and receiver response
  - Without delaying Producer
- Asynchronous communication with Callback
  - Producer
    - Sends message and callback handle (return address) to Consumer
  - Consumer
    - Sends response using callback handle
Asynchronous message communication pattern with callback pattern (Figure 11.21)

Synchronous Message Communication Without Reply Pattern

- Producer task sends message and waits for acceptance
  - Puts a brake on Producer
- Consumer receives message
  - Suspended if no message is present
  - Activated when message arrives
  - Accepts message, Releases producer
- Producer and Consumer continue
- Example – Fig. 11.23
Distributed Services and Service broker

- Service Broker
  - Mediates interactions between clients and services
  - Frees client from having to maintain information
    - Where particular service provided
    - How to obtain service
- Location transparency
  - Service can move to different servers
- Clients request information from Broker about Services
- Broker provides different services

Service Broker Patterns

- Service registration
- White pages Broker Patterns
  - Client knows name of service but not location
    - Broker Forwarding Pattern
      - Additional overhead at each service interaction
      - Rarely used in RT Design
    - Broker Handle Pattern
      - Used primarily during initialization
      - Additional overhead only at initialization
- Discovery Pattern (Yellow pages)
  - Client knows service type but not specific service
  - Used during initialization in RT design
  - Additional overhead only at initialization
Figure 11.24 Service registration with Broker

- Services registers service information with Broker
  - Service name
  - Service interface
  - Service description
  - Location
- Service re-registers after moving to different location

R1: register Service

R2: registrationAck

Figure 11.25: Broker Handle (White pages) pattern

- Broker Handle Pattern
  - Broker returns handle (remote reference) to Client
  - Client uses handle to communicate with Service

B1: serviceRequest

B2: serviceHandle

B3: serviceRequestWithHandle

B4: serviceReply
Figure 11.26: Discovery (Yellow pages) pattern

- Discovery Pattern
  - Client knows service type but not specific service
  - Client makes yellow pages query
    - Request all services of a given type
  - Client selects service
  - Client makes white pages query

<table>
<thead>
<tr>
<th>1: queryServices</th>
<th>2: serviceList</th>
<th>3: select aService</th>
</tr>
</thead>
<tbody>
<tr>
<td>aBroker</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4: service Handle</th>
<th>5: serviceRequestWithHandle</th>
<th>6: serviceReply</th>
</tr>
</thead>
<tbody>
<tr>
<td>aService Requester</td>
<td></td>
<td>aService</td>
</tr>
</tbody>
</table>

Group Message Communication

- One-to-many message communication
  - Same message sent to several recipients
- Broadcast message communication
  - Message sent to all recipients
- Multicast message communication
  - Same message sent to all members of group
- Subscription/Notification communication (Fig. 11.28)
  - Client subscribes to group
  - Receives messages sent to all members of group
  - Sender sends message to group
    - Does not need to know recipients
Figure 11.28 Example of subscription/notification (message multicast) communication

Documenting a Design Pattern

- Pattern describes
  - Pattern Name
  - Aliases
  - Context
    - When should pattern be used
  - Problem
  - Summary of solution
  - Strengths of solution
  - Weaknesses of solution
  - Applicability
    - When can you use the pattern
  - Related Patterns
  - Reference
## Subscription/Notification Pattern

<table>
<thead>
<tr>
<th>Pattern name</th>
<th>Subscription/Notification.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aliases</td>
<td>Multicast.</td>
</tr>
<tr>
<td>Context</td>
<td>Distributed systems.</td>
</tr>
<tr>
<td>Problem</td>
<td>Distributed application with multiple clients and services. Clients want to receive messages of a given type.</td>
</tr>
<tr>
<td>Summary of solution</td>
<td>Selective form of group communication. Clients subscribe to receive messages of a given type. When service receives message of this type, it notifies all clients who have subscribed to it.</td>
</tr>
<tr>
<td>Strengths of solution</td>
<td>Selective form of group communication. Widely used on the Internet and in World Wide Web applications.</td>
</tr>
<tr>
<td>Weaknesses of solution</td>
<td>If client subscribes to too many services, it may unexpectedly receive a large number of messages.</td>
</tr>
<tr>
<td>Applicability</td>
<td>Distributed environments: client/service and distribution applications with multiple services.</td>
</tr>
<tr>
<td>Related patterns</td>
<td>Similar to Broadcast, except that it is more selective.</td>
</tr>
<tr>
<td>Reference</td>
<td>Chapter 17, Section 17.6.2.</td>
</tr>
</tbody>
</table>

---

## Building Software Applications from Software Architectural Patterns

- Consider architectural structure patterns
  - Different patterns can be combined
- Start with layers of abstractions pattern
  - Incorporate client/service patterns
  - Incorporate control patterns
- Apply architectural communication patterns
  - Decouple sender components from receiver components
    - Broker patterns
    - Group communication patterns
Building Light Rail System
From Software Architectural Patterns

- Architectural Structure Patterns
  - Layered pattern
  - Client/Service pattern
  - Distributed Control

- Architectural Communication Patterns
  - Asynchronous
  - Synchronous
  - Broker
  - Subscription/Notification
Summary

• Pattern
  – Describes a recurring design problem
  – Arises in specific design contexts (i.e., situations)
  – Presents a well proven approach for its solution

• Categorization of software patterns
  – Architectural Structure Patterns
  – Architectural Communication Patterns

• Design New Software Architectures using existing patterns