SWE 721 / IT 821 Advanced Software Design:
Reusable Software Architectures

Architectural Patterns for Software Product Lines

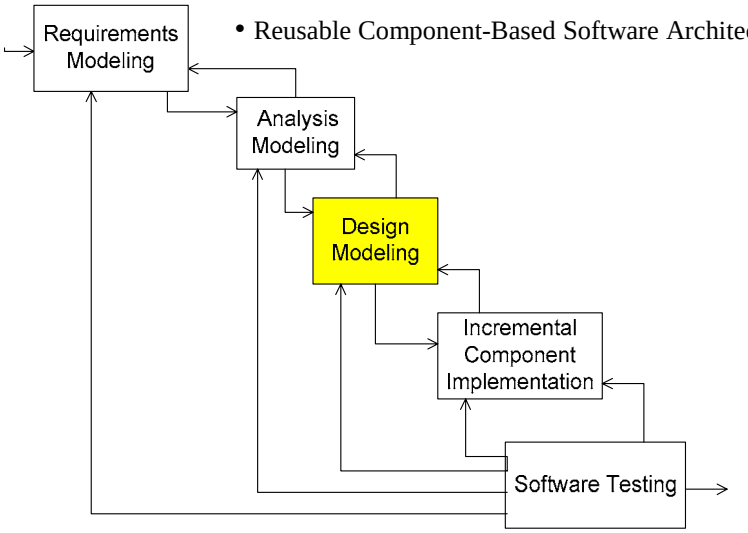
Hassan Gomaa
Department of Information and Software Engineering
George Mason University

References:
1) Hassan Gomaa, Chapter 10 in "Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures", Addison-Wesley Object Technology Series, 2005
2) F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, "Pattern Oriented Software Architecture: A System of Patterns", John Wiley & Sons, 1996

Software Product Line Design Modeling

What should SPL Design Method provide?

- Software Architectural Patterns
- Reusable Component-Based Software Architectures



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

Pattern Categories

- Design Patterns
  - Small group of collaborating objects
  - Gang of Four (Gamma, Helms, Johnson, Vlissides)
- Architecture Patterns
  - Address the structure of major subsystems of a system
  - Buschmann, etc.
- Analysis Patterns
  - Recurring patterns found in Analysis
- Domain Specific Patterns
  - Used in a specific application area (e.g., factory automation)
- Idioms
  - Low-level patterns specific to a programming language

Software Architectural Patterns

- Architectural Structure Patterns
  - Address static structure of software architecture
- Architectural Communication Patterns
  - Address dynamic communication between software components of architecture

Architectural Structure Patterns for Software Product Lines

- Layered patterns
  - Layers of Abstraction
  - Kernel
- Client/Server patterns
  - Basic Client/Server
  - Client/Broker/Server
  - Client/Agent/Server
- Control Patterns
  - Centralized Control
  - Distributed Control
  - Hierarchical Control

Architectural Structure Patterns for Software Product Lines

- Layers of Abstraction (Fig. 10.1)
  - Structure product line into hierarchical levels
  - Each layer provides services for higher layers
- Layers of Abstraction in Product Lines (Figs. 10.1, 10.2, 10.3)
  - Allows use of subsets and extensions
  - Lower layers do not depend on upper layers
    - Kernel components at lowest layer
  - Higher layers depend on lower layers
    - Optional and variant components at higher layers
- Variations in Layers of Abstraction
  - Strict Hierarchy (Fig. 10.1)
  - Flexible Hierarchy (Fig. 10.4)

Architectural Structure Patterns for Software Product Lines

- Kernel or Microkernel (Fig. 10.5)
  - 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 in Software Product Lines
  - Kernel of product line is determined
    - Always at lowest layer of hierarchy
  - Optional and variant components at higher layers depend on kernel components in kernel layer

Structural Architecture Patterns for Software Product Lines

- Client/Server (Figs. 10.6, 10.7)
  - **Client** requests services
  - **Server** is provider of services
  - Server services multiple clients
  - Client depends on server
  - Client at higher layer than server in layered architecture
- Variations:
  - Single Client / Multiple Server (Fig. 10.6)
  - Multiple Client / Multiple Server (Fig. 10.8)
    - Clients communicate with multiple servers
  - Multi-tiered Server (Fig. 10.9)
    - Servers can also be clients of other servers

Structural Architecture Patterns for Software Product Lines

- Broker Pattern (Fig. 10.11)
  - Decouples clients from servers
  - Provide location transparency
  - Supports heterogeneous systems
  - Allows for incremental expansion
- Client/Broker/Server
  - Servers registers with Broker
  - Clients communicate with Server via Broker
    - Different communication patterns
- Client/Agent/Server Pattern (Fig. 10.12)
  - Agent negotiates on behalf of clients and servers

Control Patterns

- Centralized Control Pattern
  - One control component
    - Executes statechart
  - Receives sensor input from input device interface components
  - Controls external environment via output device interface components that output to actuators
  - Entity object contain data that needs to be stored
- Examples
  - Cruise Control System
  - Microwave Oven Control System (Fig. 10.13)

Control Patterns

- Distributed Control Pattern
  - Several control components
    - Each component controls given aspect of system
    - Each component executes statechart
  - Control is distributed among the components
    - Components communicate with each other to provide overall control
    - Peer-to-peer communication
- Example
  - High Volume Manufacturing System (Fig. 10.14)

Control Patterns

- Hierarchical Control
  - Several control components
    - Each component controls given aspect of system
    - Each component executes statechart
  - Coordinator component
    - Coordinates several control components
      - Provides high level control
      - Communicates directly with each component
      - Determines next job for each control component
- Example
  - Flexible Manufacturing System (Fig. 10.15)
  - Elevator Control System (Fig. 10.16)

Architectural Communication Patterns for Software Product Lines

- Subsystems designed as components
  - Component
    - Active self-contained object with a well-defined interface, capable of being used in different applications from that for which it was originally defined
  - Components may be allocated to distributed nodes in distributed environment
- Components communicate with each other using communication patterns

Architectural Communication Patterns for Software Product Lines

- Peer-to-Peer Communication Patterns
  - Asynchronous message communication
  - Bi-directional asynchronous message communication
- Client/Server Communication Patterns
  - Synchronous message communication with reply
  - Asynchronous message communication with Callback
  - Synchronous message communication without reply
- Broker Communication Patterns
  - Broker forwarding
  - Broker handle
  - Discovery
- Group Communication Patterns
  - Broadcast
  - Subscription/notification (Multicast)

Peer-to-Peer Communication Patterns

- Asynchronous (loosely coupled) 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
  - Examples: Figs. 10.17, 10.18

Peer-to-Peer Communication Patterns

- Bi-directional asynchronous communication
  - 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
  - Examples: Figs. 10.19, 10.20

Client / Server Communication Patterns

- Synchronous communication with reply - Figs. 10.21, 10.22
  - Server
    - Responds to message requests from several clients
  - Client
    - Sends message to Server and waits for response
- Asynchronous communication with Callback - Fig. 10.23
  - Client
    - Sends message and callback handle to Server
  - Server sends remote response
- Synchronous communication without reply
  - Producer
    - Sends message to Consumer and waits for acceptance
  - Consumer
    - Waits for message from Producer (Figs. 10.24, 10.25)

Distributed Objects and Object broker

- Clients and Servers designed as distributed objects
- Object Broker
  - Mediates interactions between clients and servers
  - Frees client from having to maintain information
    - Where particular service provided
    - How to obtain service
- Servers register Services & Location with Broker
- Clients request information from Broker about Servers
- Broker provides different services

Object Broker Patterns

- White pages
  - Client knows name of service but not location
  - Broker Forwarding Pattern (Fig. 10.26)
    - Broker forwards client request to Server
    - Broker forwards Server response to Client
  - Broker Handle Pattern (Fig. 10.27)
    - Broker returns handle (remote reference) to Client
    - Client uses handle to communicate with Server
- Discovery Pattern (Yellow pages) - Fig. 10.28
  - Client knows service type but not specific server
  - Client makes yellow pages query
    - Request all services of a given type
  - Client selects service, then makes white pages query

Group Message Communication

- One-to-many message communication
  - Same message sent to several recipients
- Broadcast message communication (Fig. 10.29)
  - Message sent to all recipients
- Multicast message communication
  - Same message sent to all members of group
- Subscription/Notification (Fig. 10.30)
  - Client subscribes to group
  - Receives messages sent to all members of group
  - Sender sends message to group
    - Does not need to know recipients

Negotiation Pattern

- Client Agent
  - Acts on behalf of client
  - Negotiates with Server Agent
  - Proposes a service to Server Agent
- Server Agent
  - Negotiates with Client Agent
  - Queries servers for services provided
  - Offers client one or more server options
- Client Agent may
  - Request one of the offered options
  - Propose further options
  - Reject offer

Negotiation Pattern

- Client Agent (acts on behalf of client)
  - Propose service (negotiable)
  - Request service (non-negotiable)
  - Reject server offer
- Server Agent
  - Offers service (in response to client proposal)
  - Rejects client request/proposal
  - Accepts client request/proposal
- Example of Negotiation Pattern (Fig. 10.31)

Transaction Processing

- Transaction
  - Transaction is indivisible unit of work (atomic)
  - Two or more operations performed as one unit
- Example of Bank Transfer Transaction
  - From Savings account to Checking Account
  - Accounts maintained at two separate banks (servers)
  - One transaction consists of two operations
    - Debit Savings Account
    - Credit Checking Account
  - Transaction committed
    - Both credit and debit operations occur
  - Transaction aborted
    - Neither credit nor debit operation occurs

Transaction Pattern: Two-Phase Commit Protocol

- Used to synchronize updates on different nodes
- Result of Two-Phase Commit Protocol
  - Transaction is committed
    - Updates all succeed
  - Transaction is aborted
    - Updates all fail
- One server is designated Commit Server
- One Participant Server for each node
- In bank transfer transaction
  - Two participants

First Phase of Two-Phase Commit Protocol

- Commit Server
  - Sends Prepare to Commit message to each Participant Server
- Each Participant Server
  - Locks record
  - Performs update
  - Sends Ready to Commit (RTC) message to Commit Server
- Commit Server
  - Waits to receive RTC messages from all participants
- Example: Fig. 10.32

Second Phase of Two-Phase Commit

- Commit Server
  - Sends the Commit message to each Participant Server
- Each Participant Server
  - Makes update permanent
  - Unlocks record
  - Sends Commit Completed message to Commit Server
- Commit Server
  - Waits for all Commit Completed messages
  - If any participant sends Commit Refused message
    - Sends abort message to all participants
    - Participants roll back update
- Example: Fig. 10.33

Documenting a Design Pattern

- What a pattern must include (Buschmann)
  - Context
    - Situation leading to problem
  - Problem
    - Problem that often occurs in this context
  - Solution
    - Proven resolution to Problem

What Does a Pattern Include?

- 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

B.3.1 Compound Transaction Pattern

Pattern name	Compound Transaction.
Aliases	
Context	Distributed systems.
Problem	Client has a transaction requirement that can be broken down into smaller, separate flat transactions.
Summary of solution	Break down compound transaction into smaller atomic transactions, where each atomic transaction can be performed separately and rolled back separately.
Strengths of solution	Provides effective support for transactions that can be broken into two or more atomic transactions. Effective if a rollback or change is required to only one of the transactions.
Weaknesses of solution	More work is required to make sure that the individual atomic transactions are consistent with each other. More coordination is required if the whole compound transaction needs to be rolled back or modified.
Applicability	Transaction processing applications, distributed databases.
Related patterns	Two-Phase Commit Protocol, Long-Living Transaction.
Reference	Chapter 10, Section 10.4.2.

Example: Fig. 10.34

B.3.2 Long-Living Transaction Pattern

Pattern name	Long-Living Transaction.
Aliases	
Context	Distributed systems.
Problem	Client has a long-living transaction requirement that has a human in the loop and that could take a long and possibly indefinite time to execute.
Summary of solution	Split a long-living transaction into two or more separate atomic transactions such that human decision making takes place between each successive pair of atomic transactions.
Strengths of solution	Provides effective support for long-living transactions that can be broken into two or more atomic transactions.
Weaknesses of solution	Situations may change because of long delay between successive atomic transactions that constitute the long-living transaction, resulting in an unsuccessful long-living transaction.
Applicability	Transaction processing applications, distributed databases.
Related patterns	Two-Phase Commit Protocol, Compound Transaction.
Reference	Chapter 10, Section 10.4.3.

Example: Fig. 10.35

Applying Software Architectural Patterns

- Develop Software Architecture for product line
  - Component based software architecture
- Decide on software architectural patterns required
  - Patterns can be recognized during dynamic modeling
  - Decisions made during architectural design
    - First architectural structure patterns
    - Then architectural communication patterns
  - Different patterns can be combined
    - Start with layers of abstractions pattern
    - Consider incorporating client/server and control patterns
    - Consider architectural communication patterns
      - Synchronous, Asynchronous, Brokered, Group

Example of Applying Software Architectural Patterns Factory Automation product line

- Consider architectural structure patterns (Fig. 10.36)
  - Different patterns can be combined
- Start with layers of abstractions pattern
  - Incorporate kernel pattern
- Incorporate client/server patterns
  - Single Client / Multiple Server
  - Multiple Client / Multiple Server
- Incorporate control patterns
  - Distributed control pattern
    - High volume applications
  - Hierarchical control pattern
    - Flexible manufacturing applications

Example of Applying Software Architectural Patterns Factory Automation product line

- Consider architectural communication patterns
  - Synchronous message communication with reply
  - Asynchronous message communication
  - Bidirectional asynchronous message communication
  - Broker handle
  - Subscription/Notification (multicast)
- Example: Fig. 10.37