Component-based Architectural Design for Software Product Lines

Hassan Gomaa
Department of Information and Software Engineering
George Mason University

Reference:
Hassan Gomaa, Chapter 11 in “Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures”, Addison-Wesley Object Technology Series, 2005

Copyright © 2005 Hassan Gomaa
All rights reserved. No part of this document may be reproduced in any form or by any means, without the prior written permission of the author.

Software Product Line Design Modeling
What should SPL Design Method provide?

- Software Architectural Patterns
- Reusable Component-Based Software Architectures
Component-based Distributed Software Architecture

- EXECUTES ON MULTIPLE NODES IN DISTRIBUTED CONFIGURATION
  - CONSISTS OF DISTRIBUTED COMPONENTS

- DISTRIBUTED COMPONENT
  - WELL-DEFINED PROVIDED AND REQUIRED INTERFACES
  - CONCURRENT OBJECT
  - LOGICAL UNIT OF DISTRIBUTION AND DEPLOYMENT
  - COMMUNICATES WITH OTHER COMPONENTS USING MESSAGES
  - STRUCTURE
    - COMPOSITE OBJECT CONSISTING OF OTHER OBJECTS
    - SIMPLE OBJECT
  - CAPABLE OF BEING REUSED

Steps in Designing Distributed Applications

- DESIGN SOFTWARE ARCHITECTURE
  - DECOMPOSE SYSTEM INTO CONSTITUENT COMPONENTS
  - EACH SUBSYSTEM DESIGNED AS CONFIGURABLE COMPONENT
  - DEFINE MESSAGE COMMUNICATION INTERFACES

- DESIGN CONSTITUENT COMPONENTS
  - STRUCTURE SUBSYSTEM INTO ACTIVE OBJECTS (TASKS) AND PASSIVE OBJECTS

- DEPLOY APPLICATION
  - DEFINE COMPONENT INSTANCES
  - MAP TO HARDWARE CONFIGURATION
Design Software Architecture

- Design Message Communication between Components
  - Synchronous messages
  - Asynchronous messages
  - Figure 11.1

- Integrate Communication Models
  - Integrate UML based communication diagrams
  - Generic UML communication diagram
  - Depict all possible interactions between objects
  - Figure 11.2

Separation of Concerns in Component Design

- **Composite object.**
  - Objects that are part of composite object
  - Structure in same component (e.g., Fig. 11.5)

- **Geographical location**
  - Objects at different locations are in separate components (e.g., Fig. 11.4)

- **Clients and Servers**
  - Place in separate components (e.g., Fig. 11.3)

- **User Interface**
  - Separate client component (e.g., Fig. 11.2)

- **Proximity to source of physical data** (e.g., Fig. 11.4)
  - Ensures fast access to physical data

- **Localized autonomy** (e.g., Fig. 11.2, 11.3, 11.4)
  - Performs specific site related service
  - Same service performed at multiple sites
  - Each instance of component on separate node
  - Operational if other nodes temporarily unavailable
Separation of Concerns in Component Design

- **Performance** (e.g., Fig. 11.2, 11.4)
  - Provides time critical service
  - More predictable performance
- **Specialized Hardware** (e.g., Fig. 11.4)
  - Special purpose hardware
  - Interface to special purpose I/O devices
- **Interface to external objects**
  - External real-world object should interface to one component (e.g., Fig. 11.4)
- **Scope of Control**
  - Control object & objects it controls are in same component (e.g., Fig. 11.4)

Component Structuring Criteria

- **Client Component**
  - Requestor of Services (e.g., Fig. 11.3)
- **User Interface Component**
  - Client provides user interface
  - Collection of objects supporting needs of user (e.g., Fig. 11.2)
- **Server Component**
  - Provides service for client components (e.g., Fig. 11.3, 11.6)
- **Control Component**
  - Controls given part of system (e.g., Fig. 11.7, 11.4)
- **Coordinator Component**
  - Coordinates several control components (e.g., Fig. 11.4)
- **Data Collection Component**
  - Collects data from external environment (e.g., Fig. 11.4)
- **Data analysis Component**
  - Provides reports and/or displays (e.g., Fig. 11.6)
- **I/O Component**
  - Interacts with external environment (e.g., Fig. 11.7)
  - One or more device interface objects
  - May contain localized control object and/or entity object
Design of Server Components
- Sequential Server Component

- Receives message requests from clients
  - One message type for each service type
- Sequential Server designed as one concurrent object
  - Services client requests sequentially
  - Server completes one request before starting next
- Server Coordinator
  - Acts as server stub
  - Unpacks incoming message
  - Invokes server operation
  - Packs response in server response message
- Example: Figure 11.3

Design of Server Components
- Concurrent Server Component

- Services shared among several concurrent objects
  - Server Coordinator coordinates activities of other objects
- Synchronization algorithm is needed
  - Mutual exclusion
    - Only one reader or writer may access data repository at any one time
  - Multiple readers and writers algorithm (Fig. 11.8)
    - Multiple readers access shared data repository concurrently
    - Only one writer can updates data repository at any one time
Concurrent Server with Subscription and Notification

• Uses Subscription / Notification pattern
• Real-Time Event Monitor looks at external events
  – Records external events of interest
• Client may subscribe to Subscription Server
  – Client requests to be notified of events
  – Subscription server maintains subscription list of clients who wish to be notified of monitored events
  – Fig 11.9, S prefix
• When significant event occurs (Fig 11.9, E prefix)
  – Real-Time Event Monitor updates event archive
  – Sends message to Event Distributor
  – Event Distributor multicasts event notification to clients on subscription list

Component Interfaces in UML

• Interface
  – Collection of operations used to specify service of class or component
  – Similar to abstract class but cannot have attributes
• UML notation
  – Interface can be modeled separately from component
  – Two ways to depict (simple and expanded)
• Component can provide one or more interfaces
  – Use different interfaces if clients require different services
• Component can require one or more interfaces
Example of Component Interfaces

• Alarm Handling Server provides 2 interfaces, requires 1
  – Provides (Fig. 11.10)
    • IAlarmServer interface to receive alarm requests and subscriptions
    • IAlarmStatus interface to receive new alarms
  – Requires IAlarmNotification interface to send alarm notifications
• Operator Interface component (Fig. 11.10)
  – Requires IAlarmServer interface to make alarm requests and subscriptions
  – Provides IAlarmNotification interface to receive alarm notifications
• Workstation Controller component
  – Requires IAlarmStatus interface to post new alarms

Modeling Components in UML 2.0

• Components
  – Modeled as UML 2.0 structured classes
  – Depicted on UML 2.0 composite structure diagrams
• Component provided and required interfaces are explicitly modeled
• Provided interface
  – Collection of operations that specify the services that a component must fulfill
• Required interface
  – Services that other components provide for this component
• Components communicate with each other via ports
• Connectors join the ports of communicating components
Component Interfaces and Ports

• Components
  – Communicate with each other through ports

• Port
  – Consists of provided and/or required interfaces

• Ports and interfaces
  – Provided Port supports provided interface
  – Required Port supports required interface
  – Complex Port supports both provided and required interfaces

• Connector
  – Joins required port of one component to provided port of another component

• Example: Figures 11.11, 11.12

Design of Components

• Design of Composite Component
  – Contains nested components
  – Part components depicted as instances
  – Delegation connector
    • Provided port of composite component connected to Provided port of inner component
  – Example: Figure 11.13

• Design of “plug compatible” component
  – Producer component can be connected to multiple alternative consumer components
  – Consumer components all have same interface
  – Example: Figure 11.14

• Design of Variable Component Architectures
  – Component involved in different component architectures in different SPL members
  – Example: Figure 11.15

• Design of Component Interface Inheritance
  – Interface specialized to allow additional functionality
  – Components providing interface and requiring interface need to be modified
  – Example: Figure 11.16