Concurrent Object-Oriented Design
with UML

Hassan Gomaa
Dept of Information & Software Engineering
George Mason University

Reference: H. Gomaa, “Chapter 6 - Designing Concurrent,
Distributed, and Real-Time Applications with UML”,
Addison Wesley Object Technology Series, July, 2000

Concurrent Object-Oriented Design
Method

• Introduction to Method
• Software Architecture Design
• Task Structuring
• Information Hiding Class Design
• Detailed Software Design
• Relational Database Design
Overview

• Concurrent Object Modeling Technology (COMET)
  – Object Oriented Analysis and Design Method
  – Uses UML notation
• Provide criteria for structuring concurrent, real-time, distributed applications
• Provides steps and procedures for mapping
  – From object-oriented analysis model
  – To a concurrent object-oriented design
• See Fig. 6.1

Object-Oriented Software Life Cycle
Requirements & Analysis Modeling

• Requirements Modeling
  • Use Case Modeling
    – Define software functional requirements in terms of use cases and actors
• Analysis Modeling
  • Static Modeling
    – Define structural relationships between classes
    – Depict classes and their relationships on class diagrams
  • Dynamic Modeling
    – Define statecharts for state dependent objects
    – Defines how objects participate in use cases using collaboration diagrams or sequence diagrams
Object-Oriented Software Life Cycle

Architectural Design

• Maps analysis model (emphasis on problem domain) to design model (emphasis on solution domain)
• Structure system into subsystems
• Design each subsystem
• Sequential Applications
  – Emphasis on OO concepts
  – Information hiding, classes, inheritance
• Concurrent, Distributed and Real-Time Applications
  – Emphasis on
    • OO concepts
    • Concurrent tasking

COMET OO Analysis and Design

• UML Notation
  – Supports both Analysis and Design concepts
• COMET/UML method
  – Separate requirements activities, analysis activities and design activities
• Requirements Modeling
  – Consider system as black box
  – Develop Use Case Model
COMET OO Analysis and Design

• Analysis modeling
  – Consider analysis of problem domain
  – Determine problem oriented objects and classes
  – Analyze static viewpoint in Static Model
    • Classes, relationships, attributes
  – Analyze dynamic viewpoint in Dynamic Model
    • Statecharts
    • Object interaction model
      – Consider objects supporting each use case
      – Analyze sequence of interactions between objects
      – Analyze information passed between objects

COMET OO Analysis and Design

• Design Model
  – Consider solution domain
  – Make decisions about overall software architecture
  – Make decisions about distributed component-based subsystems
  – Make decisions about characteristics of objects
    • Active or Passive
  – Make decisions about characteristics of messages
    • Asynchronous or Synchronous (with/without reply)
  – Make decisions about class interfaces
    • Operations and parameters
  – Make detailed design decisions
Steps in Using COMET/UML

1. Develop Object-Oriented Requirements Model
   - Develop Use Case Model (Chapter 7)

2. Develop Object-Oriented Analysis Model
   - Develop static model of problem domain (Chapter 8)
   - Structure system into objects (Chapter 9)
   - Develop statecharts for state dependent objects (Chapter 10)
   - Develop object interaction diagrams for each use case (Chapter 11)

3. Develop Object-Oriented Design Model
   - Design Overall Software Architecture (Chapter 12)
   - Design Distributed Component-based Subsystems (Chapter 13)
   - Structure Subsystems into Concurrent Tasks (Chapter 14)
   - Design Information Hiding Classes (Chapter 15)
   - Develop Detailed Software Design (Chapter 16)
Transition from Analysis to Design: Consolidation of Collaboration Diagrams

• Used to determine overall structure of system
• Merger of Collaboration Diagrams
  – Start with first Collaboration Diagram
  – Superimpose other Collaboration Diagrams
    • Add new objects and new message interactions from each subsequent diagram
    • Objects and interactions that appear on multiple diagrams are only shown once
    • Consider alternative scenarios for each use case
• Consolidated Collaboration Diagram
  – Shows all objects and their interactions
  – Example: Fig. 12.5

Consolidation of Collaboration Diagrams

• Subsystem collaboration diagram
  – High-level collaboration diagram
  – Shows subsystems and their interactions
  – Example: Fig. 12.6
• Consolidated collaboration diagram
  – If there are too many objects for one consolidated collaboration diagram
  – Develop subsystem collaboration diagram
  – Develop consolidated collaboration diagram for each subsystem
Design of Software Architecture

• Software Architecture
  – Define overall structure of system
    • Component interfaces and interconnections
  – Separately from component internals
• Each subsystem performs major service
  – Contains highly coupled objects
  – Relatively independent of other subsystems
  – May be decomposed further into smaller subsystems
  – Subsystem is aggregate or composite object
• Candidates for subsystem
  – Objects that participate in same use case

Separation of Subsystem Concerns

• Aggregate/composite object.
  – Objects that are part of aggregate/composite object
  – Structure in same subsystem (e.g., Fig. 12.7)
• Interface to external objects
  – External real-world object should interface to 1 subsystem (e.g., Fig. 12.4)
• Scope of Control
  – Control object & objects it controls are in same subsystem (e.g., Fig. 12.8)
• Geographical location
  – Objects at different locations are in separate subsystems (e.g., Fig. 12.4)
• Clients and Servers
  – Place in separate subsystems (e.g., Fig. 12.1)
• User Interface
  – Separate client subsystem (e.g., Fig. 12.9)
Subsystem Structuring Criteria

- Control
  - Subsystem controls given aspect of system (e.g., Fig. 12.8)
- Coordination
  - Coordinates several control subsystems (e.g., Fig. 12.4)
- Data Collection
  - Collects data from external environment (e.g., Fig. 12.9)
- Data analysis
  - Provides reports and/or displays (e.g., Fig. 12.9)
- Server
  - Provides service for client subsystems (e.g., Fig. 12.9, 12.10)
- User Interface
  - Collection of objects supporting needs of user (e.g., Fig. 12.9)

Static Modeling at Design Level

- Design Modeling
  - More detailed static model is developed
  - Start from conceptual static model if there is sufficient detail (e.g., Fig. 12.11)
  - Otherwise, start from consolidated collaboration diagrams (e.g., Fig. 12.13)
    - Design class for each object
    - Design relationship for each link between objects
    - Show direction of navigability on class diagram
    - Generalization/specialization relationships are only shown on class diagram
  - Can combine the two approaches
  - Example: Fig. 12.11 – 12.12
Architectural Design of Distributed Applications

Hassan Gomaa
Dept of Information & Software Engineering
George Mason University


• Distributed processing environment
  – Multiple computers communicating over network
• Typical applications
  – Distributed real-time data collection
  – Distributed real-time control
  – Client / server applications
• COMET/UML for Distributed Applications
  – Addresses structuring application into distributed subsystems
  – Examples: Figs. 4.6 – 4.8
Active and Passive Objects

- Objects may be active or passive
- Active object
  - Concurrent Task
  - Has thread of control
- Passive object
  - a.ka. Information Hiding Object
  - Has no thread of control
  - Operations of passive object are executed by task
  - Operations execute in task’s thread of control
    - Directly or indirectly
- Software Design terminology
  - Task refers to active object
  - Object refers to passive object

Sequential and Concurrent Systems

- Sequential Systems
  - Sequential program
  - One thread of control (task)
  - Several passive objects
  - Synchronous communication
    - Operation (procedure or function) call
- Concurrent Systems
  - Several active objects (tasks)
  - Each task has its own thread of control
    - Asynchronous communication
    - Synchronous communication
Characteristics of Distributed Applications

• Structure of distributed application
  – Consists of one or more subsystems
  – Execute on multiple nodes in distributed configuration
  – Subsystems determined using subsystem structuring criteria
• Structure of Subsystem
  – Consists of one or more objects
  – Objects all execute on same node
• Communication between subsystems
  – Message communication
  – Example: Fig. 4.14

Steps in Designing Distributed Applications

• System Decomposition
  – Decompose system into distributed subsystems
    • Design as configurable components
  – Define message communication interfaces
  – Examples: Figs. 13.1 – 13.3
• Subsystem Decomposition
  – Structure subsystem into active objects (tasks) and passive objects
• System Configuration
  – Define component subsystem instances of target system
  – Map to hardware configuration
  – Example: Fig. 13.14
Define Subsystem Interfaces

- Message Communication between distributed subsystems
  - Loosely coupled (asynchronous) message communication
  - Multiple Client / Server message communication
    - Tightly coupled (synchronous) message communication
    - Remote Procedure Call
  - Group Message Communication
    - Broadcast message communication
    - Multicast message communication
  - Transaction management
  - Object Broker Architecture

Loosely Coupled Message Communication
(Asynchronous)

Producer
Sends message and continues
Message queue may build up

Consumer receives message
Suspended if no message present
Activated when message arrives
Processes message
Not suspended if message present

Example: Fig. 13.2
Client / Server Message Communication

- Server
  - Often encapsulates data store(s)
  - Does not initiate any message communication
  - Responds to message requests from clients
  - Usually services many clients
- Client
  - Sends message to server
  - Usually has to wait for response
- Tightly Coupled Message Communication
  - Remote Procedure Call
- Examples: Figs. 13.1, 4.15, 4.16

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
  - Client subscribes to group
  - Receives messages sent to all members of group
  - Sender sends message to group
    - Does not need to know recipients
  - Example: Figs. 13.4
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
- Example: CORBA, Fig. 4.17

Object Broker Architecture

- White pages
  - Client knows name of service but not location
  - Forwarding design (Fig. 13.5)
    - Broker forwards client request to Server
    - Broker forwards Server response to Client
  - Handle-driven design (Fig. 13.6)
    - Broker returns handle (remote reference) to Client
    - Client uses handle to communicate with Server
- Yellow pages
  - Client knows service type but not specific server
  - Client makes yellow pages query (Fig. 13.7)
    - Request all services of a given type
  - Client selects service, then makes white pages query
Task Structuring

Hassan Gomaa
 Dept of Information & Software Engineering
 George Mason University


Copyright © 2002 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.

Structure System into Tasks

• Concurrent Design with UML
• Concurrent task structuring criteria
  – Structure analysis model into concurrent tasks
    – Task is an active object
    – Task has thread of control
  – Consider concurrent nature of system activities
  – Determine concurrent tasks
• Define task interfaces
• Develop concurrent collaboration diagrams
• Develop task behavior specifications
Operating System Support for Concurrent Tasks

Multi-tasking Kernel (nucleus, executive)
Services
  Task creation and deletion
  Priority pre-emption task scheduling
  Inter-task synchronization using events
  Mutual exclusion using semaphores
  Inter-task communication using messages
Examples
  Unix, VRTX, Windows/NT, Linux

Language Support for Concurrent Tasks

Multi-tasking capabilities
  Concurrent tasking constructs
  Task creation and deletion
  Support for inter-task communication and synchronization
  Run time system handles scheduling and dispatching of tasks
Examples
  Ada
  Java
Task Structuring -
Task Structuring Categories

- I/O task structuring criteria
  - How device interface objects are mapped to I/O tasks
- Internal task structuring criteria
  - How internal objects are mapped to internal tasks
- Task priority criteria
  - Importance of executing task relative to others
- Task clustering criteria
  - Whether and how objects should be combined into concurrent tasks

I/O Task Structuring Criteria

- Asynchronous I/O device interface task
  - Task for each asynchronous I/O device
  - Asynchronous device generates interrupt
  - Example: Fig. 14.1
- Periodic I/O device interface task
  - Task for each polled I/O device
  - I/O device (usually input) sampled at regular intervals
  - Example: Fig. 14.2
- Passive I/O device interface tasks
  - Task for each passive I/O device (usually output)
  - Computation overlapped with output
  - Example: Fig. 14.3
- Resource Monitor task
  - Task for each I/O device that receives requests from multiple sources
  - Example: Fig. 14.4
Internal Task Structuring Criteria

- Periodic task
  - Task for each periodic activity
  - Example: Fig. 14.5
- Asynchronous task
  - Task for each asynchronous internal activity
  - Example: Fig. 14.6
- Control task
  - Task executes statechart
  - Example: Fig. 14.7, 14.8
- User interface task
  - Task for each sequential user activity
  - Example: Fig. 14.9

Task Priority Criteria

- Important consideration
  - Performance Analysis
  - Real-Time Scheduling
    - Rate Monotonic Analysis
- Time critical
  - Activity that has hard deadline
  - Map to time critical task
  - Example: Fig. 14.1
- Non-time-critical computationally intensive
  - Low priority activity
  - Example: Fig. 14.3
Task Clustering Criteria

- Temporal clustering
  - Activities activated by same event
  - Example: Fig. 14.10
- Sequential clustering
  - Activities must be executed sequentially
  - Example: Fig. 14.11
- Control clustering
  - Control object grouped with objects it activates
  - Example: Fig. 14.12
- Mutually exclusive clustering
  - Activities cannot execute concurrently
  - Example: Fig. 14.13
- Task Inversion
  - Map all objects of same type to one task
  - Example: Fig. 14.14

Define Task Interfaces

- Map Analysis Model Interaction Diagram interfaces to task interfaces
- Simple messages with data transfer between concurrent tasks
  - Need to determine type of message communication
- Simple messages without data transfer between concurrent tasks (synchronization only)
  - Event synchronization
  - Message communication
- Passive objects
  - Information hiding objects
- Update task architecture on concurrent collaboration diagrams
Task Interfaces

Producer task sends data to consumer task
  Loosely coupled message communication
  Tightly coupled message communication
    With reply
    Without reply
  Event synchronization
  Task interface to information hiding object

Loosely Coupled Message Communication
(Asynchronous)

Producer
  Sends message and continues
  Message queue may build up
Consumer receives message
  Suspended if no message present
  Activated when message arrives
    Processes message
  Not suspended if message present

Example: Fig. 14.18
Tightly Coupled (Synchronous) Message Communication With Reply

Producer
  Sends message
  Waits for reply

Consumer
  Suspended if no message present
  Activated when message arrives
  Accepts message
  Generates and sends reply

Producer and Consumer continue
Example: Fig. 14.19

Tightly Coupled (Synchronous) Message Communication Without Reply

Producer
  Sends message
  Waits for acceptance by Consumer

Consumer
  Suspended if no message present
  Activated when message arrives
  Accepts message
  Releases producer

Producer and Consumer continue
Example: Fig. 14.20
Information Hiding Object

- Passive object
- Encapsulates data store
  - Hides contents of data store
  - Data store accessed indirectly via operations
    - Access procedures
    - Access functions
- Data store
  - Accessed by two or more tasks
  - Access procedures/functions
    - Must synchronize access to data
    - Example: Fig. 14.24

Event Synchronization

Types of events
- External event (interrupt) – Fig. 14.21
- Timer event – Fig. 14.22
- Internal event – Fig. 14.23

Two tasks may need to synchronize their operations

- If message contains no data, can use internal event
- Source task signals event
  - Signal (Event)
- Destination task waits for event
  - Wait (Event)
  - Suspended until event signaled
**Task Behavior Specifications (TBS)**

TBS evolves as task design is refined
- First developed during Task Structuring
- Refined during Detailed Software Design

Describes concurrent task's
- Interface
- Structure
- Timing characteristics
- Relative priority
- Event sequencing logic

**Task Behavior Specification**

Task interface
- Message communication
  - Type of interface
  - Message names and parameters
- Events signaled
  - Name and Type of event
- External inputs or outputs

Task structure information
- Task structuring criterion used to design task

Task's event sequencing logic
- Response to each message or event input
- Described informally in Pseudocode