SWE 621:
Software Modeling and Architectural Design

Lecture Notes on Software Design

Lecture 12 - Software Design Patterns

Hassan Gomaa
Dept of Computer Science
George Mason University
Fairfax, VA

Introduction to
Architecture and Design Patterns

Hassan Gomaa

Reference: H. Gomaa, Chapters12, 15,16 - *Software Modeling and Design*, Cambridge University Press, February 2011

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. at Siemens
- Analysis Patterns
  - Recurring patterns found in Analysis
  - Fowler
- Domain Specific Patterns
  - Used in a specific application area (e.g., factory automation, Internet terminal)

Software Architectural Patterns

- Architectural Structure Patterns
  - Address static structure of software architecture
  - E.g., layers of abstraction, client/service
- Architectural Communication Patterns
  - Address dynamic communication between software components of architecture
  - E.g., asynchronous message communication, broker forwarding

Software Architectural Patterns

- Also called Software Architectural Styles
  - Recurring architectures used in various software applications
- Client/Server Architecture pattern (Fig. 15.1, 15.4)
  - **Client** requests services
  - **Server** is provider of services
- Layers of Abstraction pattern (Fig. 12.4)
  - Hierarchical architecture
  - Each layer provides services for layers above it
  - Operating systems, network communications software
- Centralized Control Pattern (Fig. 18.2)
  - One control component executes statechart
  - Receives sensor input from input components
  - Controls external environment via output components

Figure 15.1 Multiple Client / Single Service Pattern

Client/Service (Figs. 15.1)
Client requests services
Service is provider of services

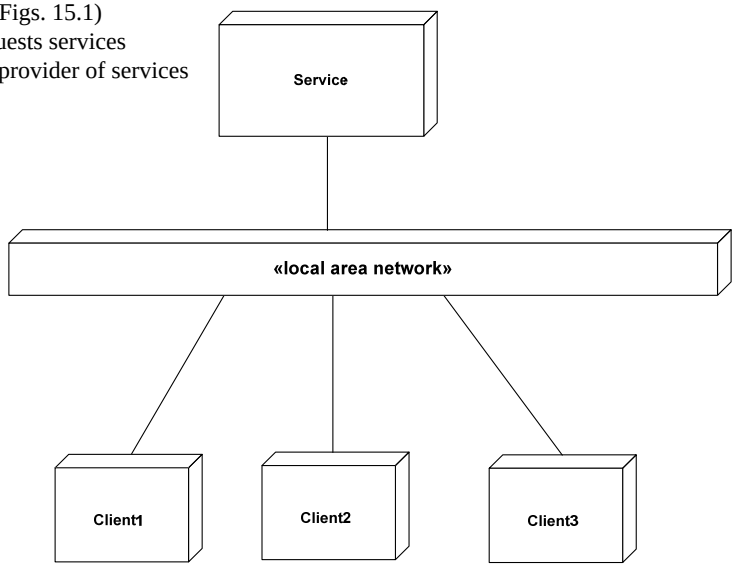


Figure 15.4 Multiple-client /multiple-server pattern

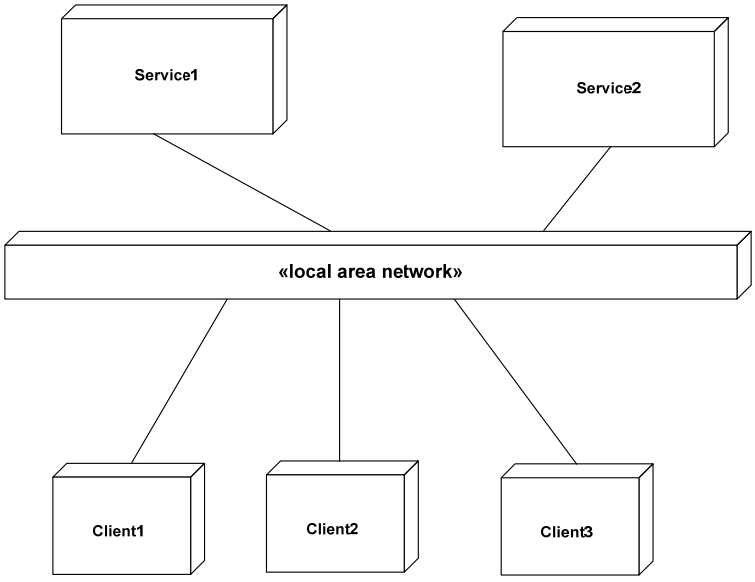
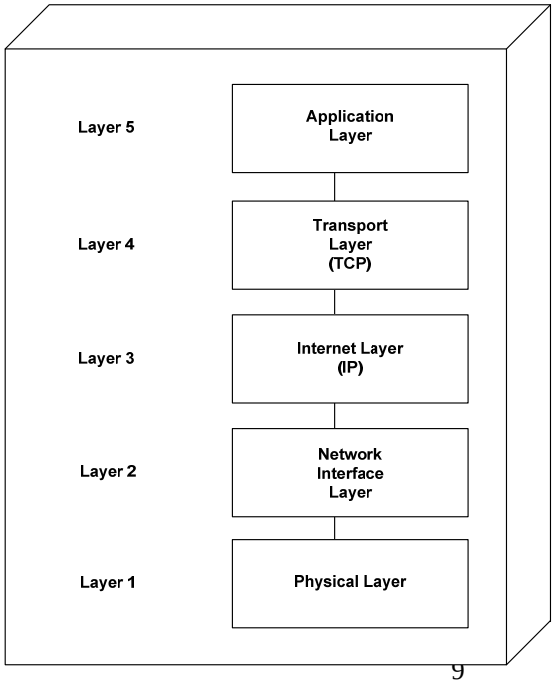


Figure 12.4 Example of layered pattern - Five layers of Internet (TCP/IP) reference model



Example of hierarchical architecture - Cruise Control and Monitoring System

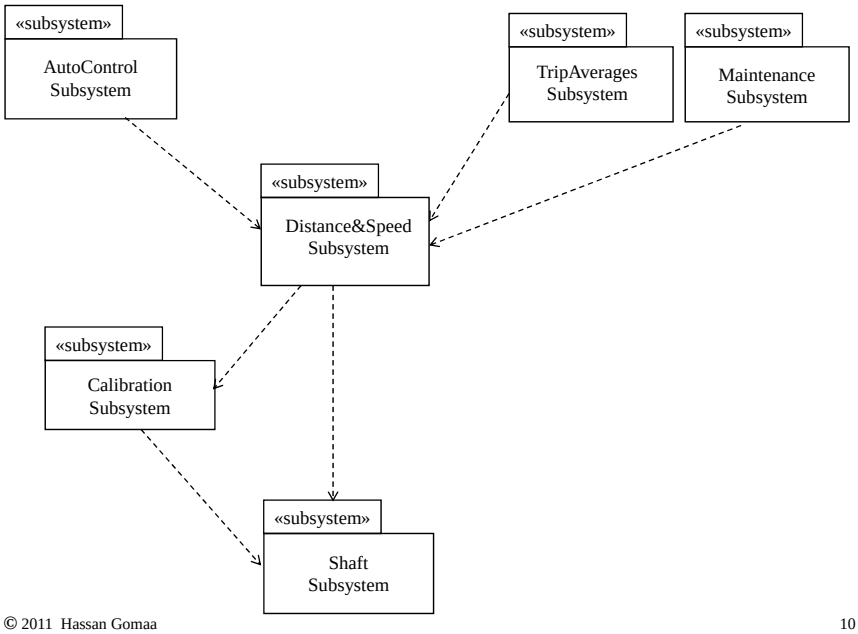
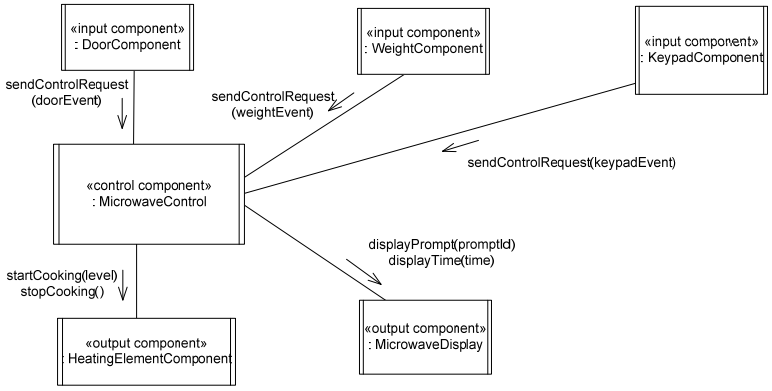


Figure 18.2 Example of centralized control architectural pattern



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

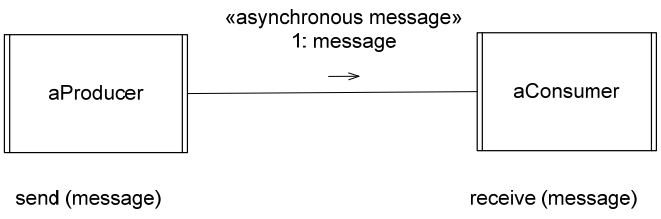
What Does a Pattern Include?

- Pattern describes
  - Pattern Name
  - Alias
  - 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

FIFO Queue Pattern -

Alias: Loosely Coupled Message Communication
Alias: Asynchronous Communication

- 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



Asynchronous Message Communication Pattern

- **Pattern Name**: Asynchronous message communication.
- **Alias**: Loosely coupled message communication, FIFO Queue.
- **Context**: Concurrent systems.
- **Problem**: Concurrent application with concurrent tasks that need to communicate with each other. Producer does not need to wait for consumer. Producer does not need reply.
- **Summary of solution**: Use message queue between producer task and consumer task. Producer sends message to Consumer and continues. Consumer receives message. Messages may be queued FIFO (first-in-first-out) if Consumer is busy. Consumer is suspended if no message is available.
- **Strengths**: Consumer does not hold up Producer.
- **Weaknesses**: If Producer produces messages more quickly than Consumer can consume them, the message queue will eventually overflow.
- **Applicability**: Centralized and distributed environments: Real-time systems, client/server and distribution applications.
- **Related Patterns**: Tightly coupled message communication with/without reply.

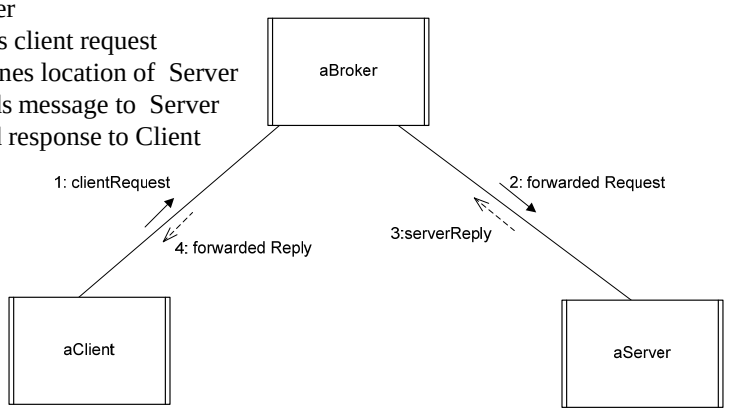
Figure 13.5 Object broker architecture
(White pages - forwarding design)

Object Broker Architecture - Forwarding Design

- Client queries Broker for services provided

Client sends message to Server via Broker

- Identifies Server name and service required
- Object Broker
  - Receives client request
  - Determines location of Server
  - Forwards message to Server
  - Forward response to Client



Broker Forwarding Pattern

- **Pattern Name**: Broker Forwarding
- **Alias**: Object Broker with Forwarding
- **Context**: Distributed Systems
- **Problem**: Distributed application with multiple clients communicating with multiple servers. Clients do not know location of servers.
- **Summary of solution**: Use Object Broker. Servers register their services with the Object Broker. Clients send service request to Broker. Broker forwards request to Server. Server services request and sends reply to Broker. Broker forwards reply to Client.
- **Strengths**: Location transparency - Servers may relocate easily. Clients do not need to know location of Servers.
- **Weaknesses**: Additional overhead because Object Broker is involved in all message communication. Broker can become a bottleneck if there is a heavy load at the Broker.
- **Applicability**: Distributed environments: Client/server and distribution applications with multiple servers.
- **Related Patterns**: Broker Handle.

Review of Design Patterns

- 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