WVU, LCSEE
CS 430

Fall 2007

Architectural Design
Patterns
What are patterns?

- First discussions about design patterns in late 80’s
  - Most practical work started in mid 90’s
- A software design pattern is a model proposed for imitation in solving software design problems.
- Used at different levels of abstraction:
  - Architectural design
  - Mid level design (typically collaboration between classes)
  - Data structures and algorithms
  - Programming idioms
Layered architectural style

Typical organization for network protocols, operating systems, search and retrieval systems.
Pipe-and-Filter

Typically I/O transformation systems. If filters run concurrently, pipes needed for synchronization.
Split/Join data streams

Pipe-and-Filter comes in different topologies.
Pipe-and-Filter

**Advantages:**
- Filters easily modified
- Filters can be easily rearranged and reused
- Supports concurrency

**Disadvantages**
- Difficult to coordinate activities of filters
- Filters may have problems with varying and complex structures of data streams
- Error handling is difficult
- Synchronization, while possible, may be difficult
Shared Data Style

- Figure missing
Event-Driven

**Name:** Event-Driven or Implicit-Invocation

**Application:** Model a program that must react to unpredictable sequences of inputs.

**Form:** This style does not specify a static form. In its dynamic form, one or more event announcers send events to an event dispatcher. When an event occurs, the event dispatcher invokes procedures of components that have registered interest in the event.

**Consequences:** Components that announce events are independent of other components that announce events and of components that respond to them, which are themselves independent of one another. Hence, components are easy to change, replace, remove, or add, making Event-Driven-style programs changeable, maintainable, robust, and fault tolerant.

Interacting entirely through the event dispatcher is awkward. The independence of components may make it hard to establish program correctness. Program performance may degrade when many events occur in quick succession.

**Figure 15-3-9 Event-Driven Architectural Style**

---

**Figure 15-3-4 An Event-Driven Program**
Model-View-Controller Style

Models: problem-domain functions
Views: Data Displays
Controllers: receive and carry out commands

Name: Model-View-Controller (MVC)
Application: Model a program with interactive user interfaces.
Form: This style is a specialization of the Strict Layered style. The top user interface layer contains view modules for data display and controller modules for user input. The bottom application domain layer contains model modules embodying the data and operations implementing core program function. Controllers may change views or models, and models may change on their own. When a model changes, it notifies views and controllers, which respond by querying the model and updating themselves.
Consequences: The user interface is decoupled from the models, so it may be changed independently, increasing changeability and maintainability. Views and controllers may be changed even at runtime, increasing flexibility and configurability. However, views and controllers are highly dependent on models, decreasing application domain layer changeability and maintainability. User interface update performance may be a problem.
Brokers

**Figure 17-1-2 Broker Pattern Behavior**

Name: Broker Category

**Application:** Provide a class to mediate between a client and a supplier as a means to simplify or decompose the supplier or to facilitate communication between the client and the supplier.

**Form:** Broker patterns have three classes: a client, a broker, and a supplier. The client interacts with the broker, which in turn interacts with the supplier on the client’s behalf.

**Consequences:** Broker patterns simplify or decompose suppliers, or ease communication between them. Broker patterns thus increase changeability, modifiability, and reusability. By interposing additional processing, they may degrade performance.

**Figure 17-1-5 The Broker Category**
Façade Pattern

**Figure 17-2-1 Façade Pattern Structure**

**Name:** Façade  
**Application:** Add an interface class to a sub-system.  
**Form:** The façade class is a broker that mediates the interaction between a client class and a collection of supplier classes in a sub-system.  
**Consequences:** The façade class makes a sub-system easier for clients to use, lowers the coupling between the client and the sub-system, and may increase reusability by adapting a sub-system's interface to client needs.

**Figure 17-2-8 The Façade Pattern**
Mediator pattern

Name: Mediator
Application: Add an intermediate class to control the interaction of several collaborating classes.
Form: The mediator class is a broker that decouples several collaborating classes and facilitates their interaction. Each collaborator holds a reference to the mediator, which holds a reference to each collaborator. Collaborator interaction goes through the mediator.
Consequences: The mediator class decouples collaborating classes, making them more reusable and changeable. It encapsulates an interaction, making the interaction easier to change.

Figure 17-2-9 The Mediator Pattern

Figure 17-2-2 Unmediated and Mediated Collaborations
Proxy pattern

**Name:** Proxy

**Application:** Provide a stand-in for another object that is not yet created, is not locally available, has access restrictions, or is unavailable for other reasons.

**Form:** The proxy class is a broker with the same interface as the supplier for which it stands in. The proxy eases the interaction between the client and the supplier it replaces or augments the function of the supplier. The proxy may handle some requests; it passes on those it cannot handle to the supplier.

**Consequences:** The Proxy pattern makes it possible to defer expensive operations until they are necessary (virtual proxies), provides an elegant way to treat remote objects as if they were local (remote proxies), and provides a mechanism for implementing supplier access restrictions (protection or access proxies).