Software Design Refinement Using Design Patterns

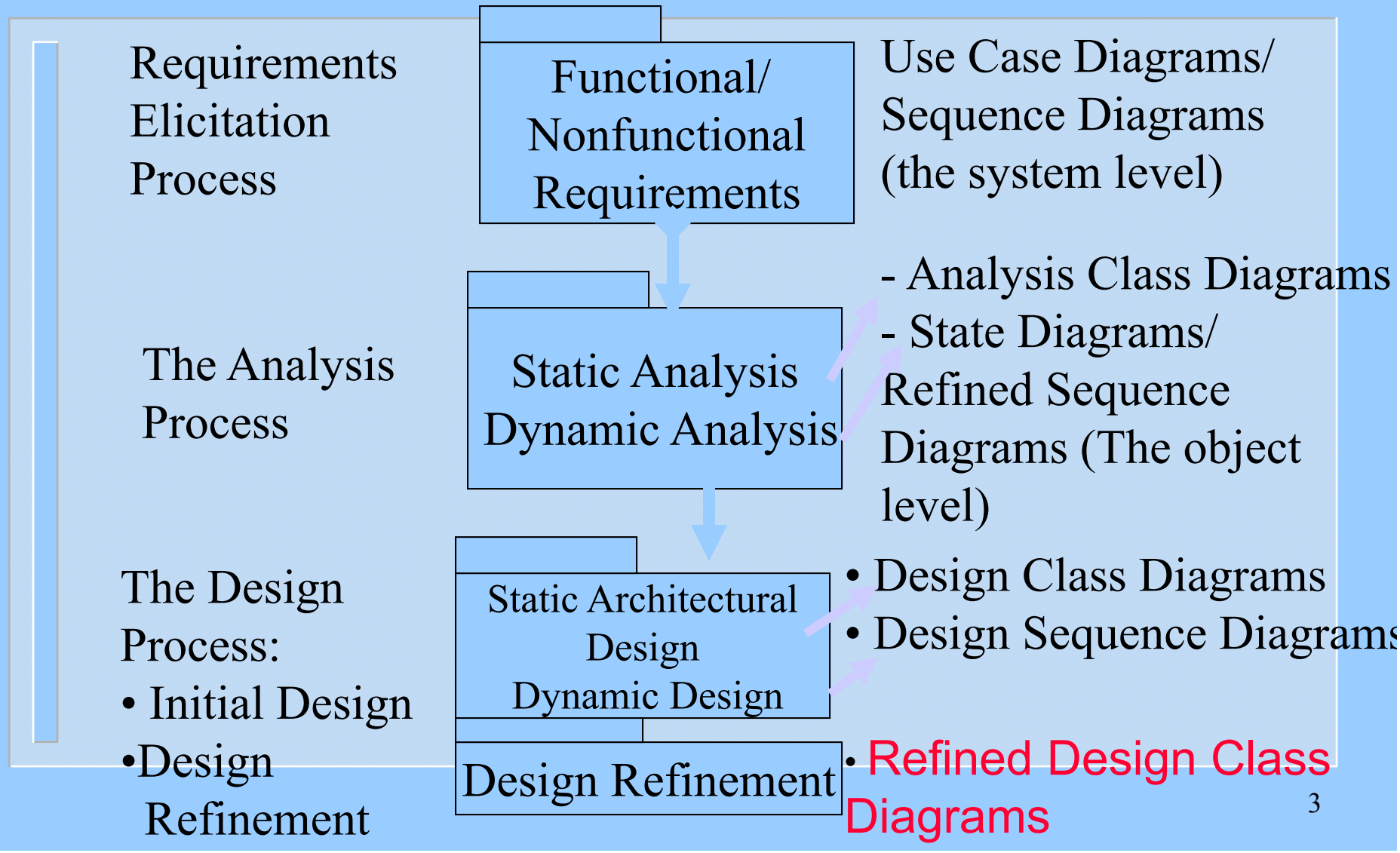
Instructor: Dr. Hany H. Ammar

Dept. of Computer Science and Electrical Engineering, WVU

Outline

- The Requirements, Analysis, Design, and Design Refinement Models
- Class diagram refinement using design patterns
- Design patterns examples
  - The Facade pattern
  - The Strategy Pattern
  - The State Pattern
  - The Command Pattern
  - The Observer Pattern
  - The Proxy Pattern
- Design Patterns Tutorials

The Requirements, Analysis, Design, and Desgin Refiement Models



Design Refinement

- It is difficult to obtain a quality design from the initial design
- The initial design is refined to enhance design quality using the software design criteria of modularity, information hiding, complexity, testability, and reusability.
- New components (or new classes) are defined and existing components (or classes) structures are refined to enhance design quality
- The design refinement step is an essential step before implementation and testing.

Outline

- The Requirements, Analysis, Design, and Design Refinement Models
- Class diagram refinement using design patterns
- Design patterns examples
  - The Facade pattern
  - The Strategy Pattern
  - The State Pattern
  - The Command Pattern
  - The Observer Pattern
  - The Proxy Pattern
- Design Patterns Tutorials

Class Diagram Refinement Using Design Patterns

- Design Class Diagrams are further refined to enhance design quality (i.e., reduce coupling, increase cohesion, and reduce component complexity) **using design patterns**
- A design pattern is a documented good design solution of a design problem
- Repositories of design patterns were developed for many application domains (communication software, agent-based systems, web applications)
- Many generic design patterns were defined and can be used to enhance the design of systems in different application domains

What is a Design Pattern

- What is a Design Pattern?

  A design pattern describes a design problem which repeatedly occurred in previous designs, and then describes the core of the solution to that problem
- Solutions are expressed in terms of classes of objects and interfaces (object-oriented design patterns)
- A design pattern names, abstracts, and identifies the key aspects of a high quality design structure that make it useful for creating reusable object-oriented designs

Defining a Design Pattern

- Design Patterns are documented in the literature by a template consisting of the following

A Design Pattern has 5 basic parts:

1. Name
2. Problem
3. Solution
4. Consequences and trade-of of application
5. Implementation: An architecture using a design class diagram

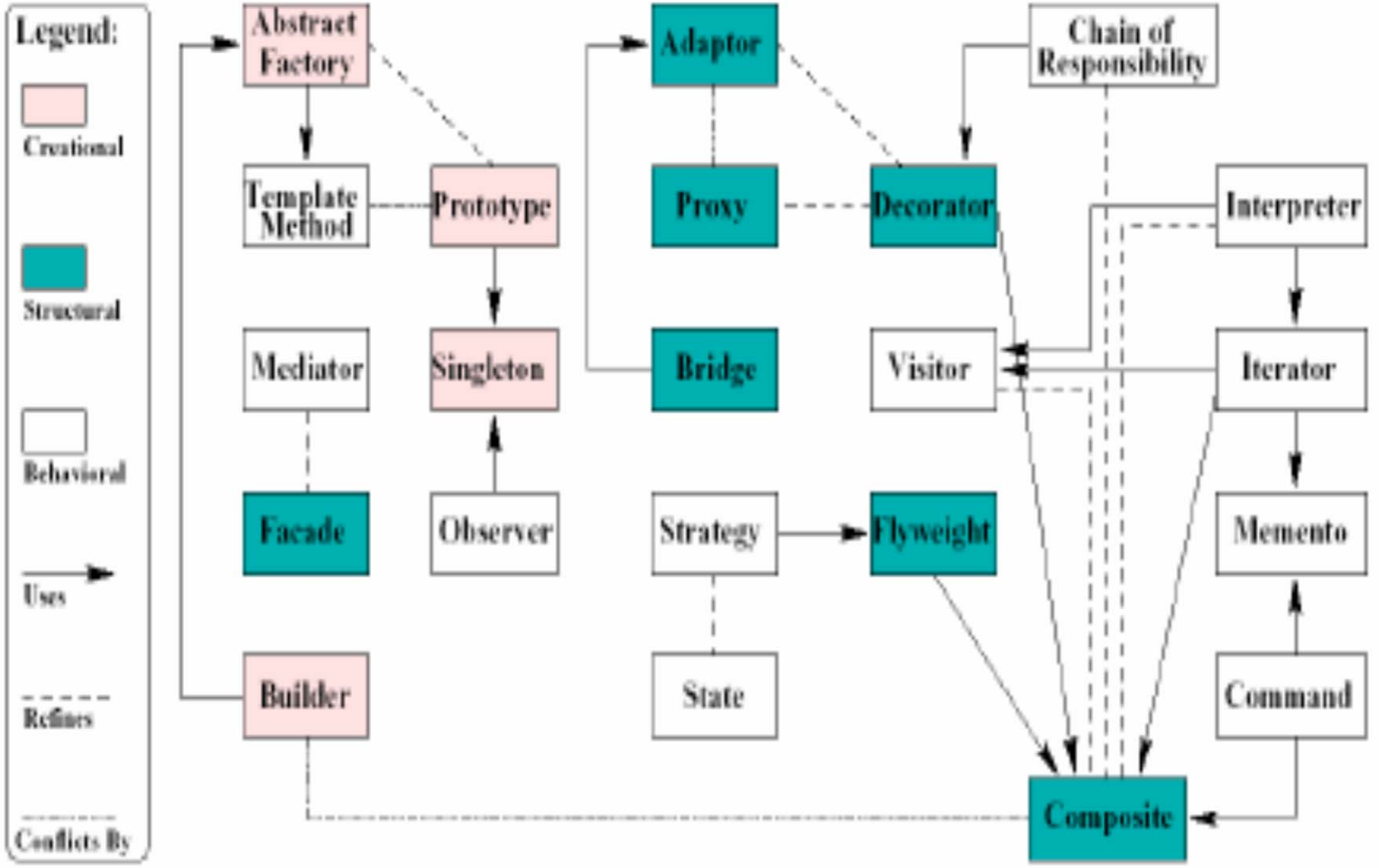
Types of Design Patterns

The Gang of Four (GoF) Patterns (Gamma et al 1995)

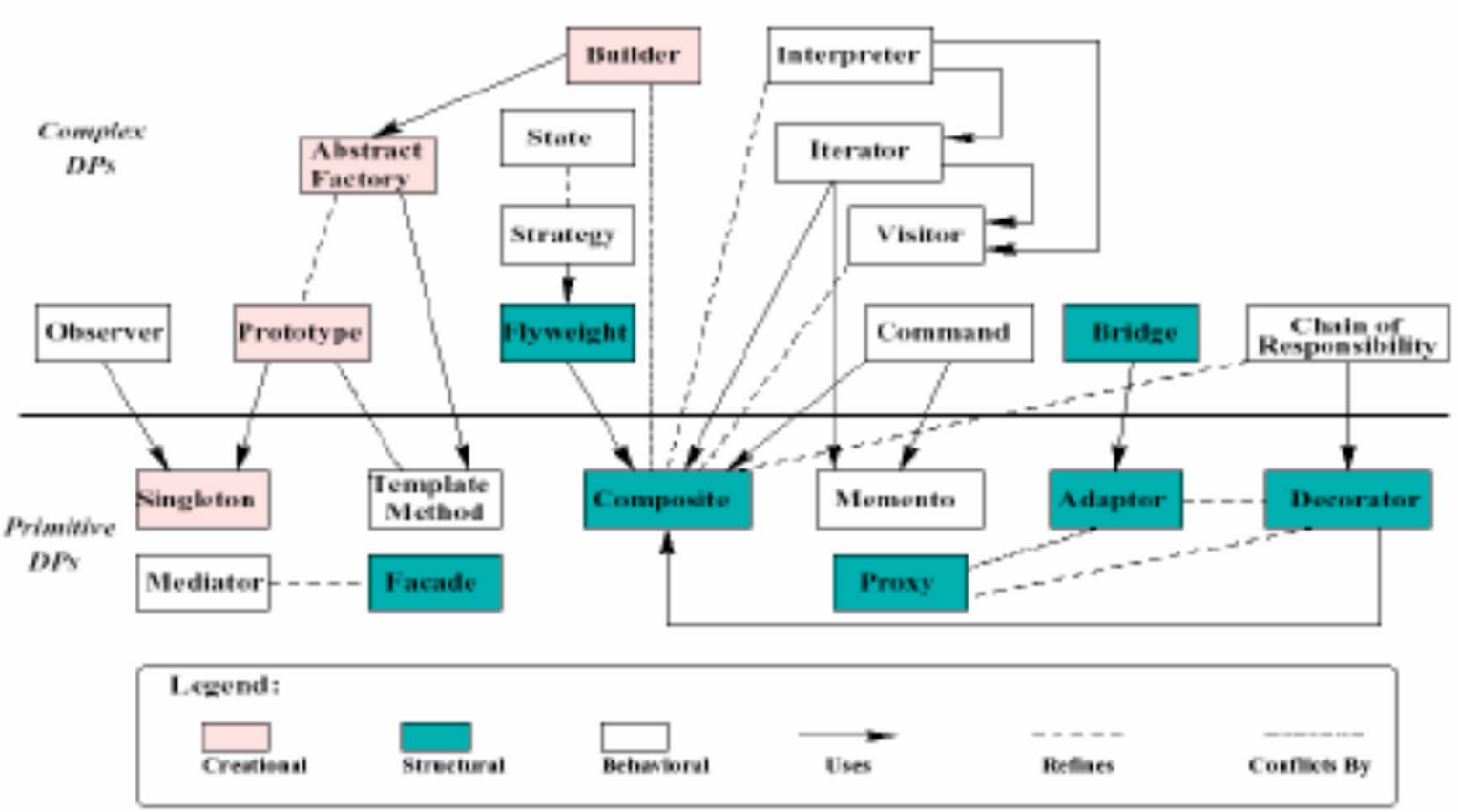
Design Pattern Space

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method	Adapter (class)	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Flyweight Facade Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

2. Relation among patterns



A layered version



Outline

- The Requirements, Analysis, Design, and Design Refinement Models
- Class diagram refinement using design patterns
- Design patterns examples
  - The Facade pattern
  - The Strategy Pattern
  - The State Pattern
  - The Command Pattern
  - The Observer Pattern
  - The Proxy Pattern
- Design Patterns Tutorials

Example of Pattern Definition: The Façade Pattern Provides An Interface To a Subsystem

The Facade Pattern: Key Features

Intent	You want to simplify how to use an existing system. You need to define your own interface.
Problem	You need to use only a subset of a complex system. Or you need to interact with the system in a particular way.
Solution	The Facade presents a new interface for the client of the existing system to use.
Participants and Collaborators	It presents a specialized interface to the client that makes it easier to use.
Consequences	The Facade simplifies the use of the required subsystem. However, since the Facade is not complete, certain functionality may be unavailable to the client.
Implementation	• Define a new class (or classes) that has the required interface. • Have this new class use the existing system.
GoF Reference	Pages 185–193.

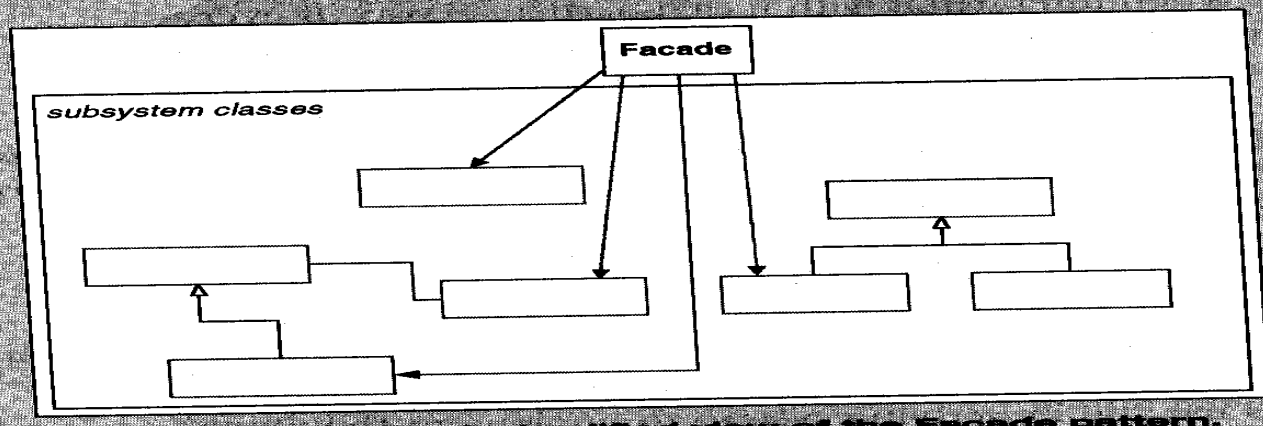


Figure 6-3 Standard, simplified view of the Facade pattern.

The Facade Pattern

- The class Facade is introduced as an interface to the whole subsystem.
- Any client class needs a service from any of the subsystem classes will send the request to the facade class.
- All the subsystem interfaces are combined in one class

Example of Using the Design Pattern

- Design Patterns produce quality designs by reducing coupling
- Example of how a Façade Pattern reduces coupling

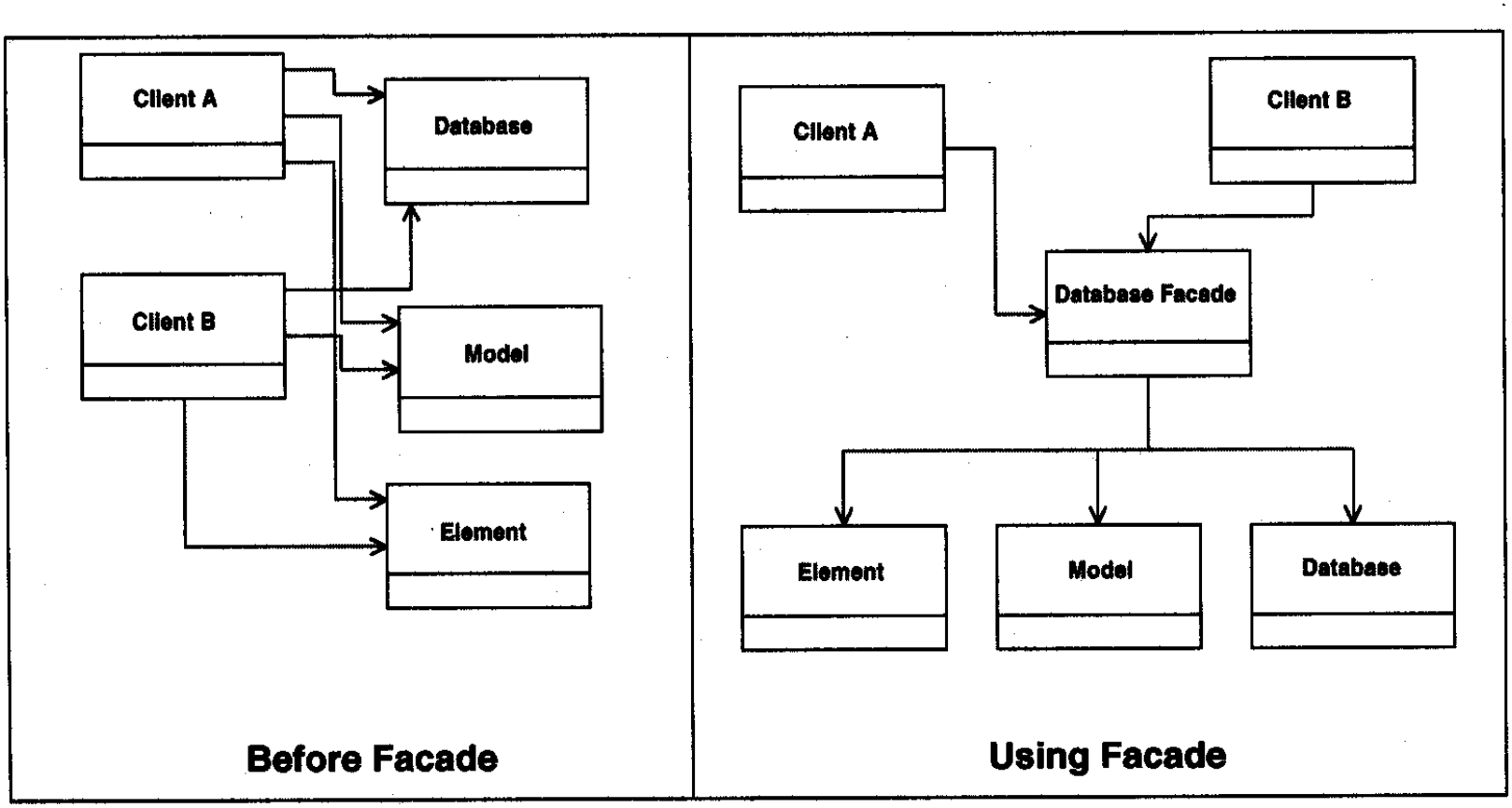


Figure 6-4 Facade reduces the number of objects for the client.

Outline

- The Requirements, Analysis, Design, and Design Refinement Models
- Class diagram refinement using design patterns
- Design patterns examples
  - The Facade pattern
  - The Strategy Pattern
  - The State Pattern
  - The Command Pattern
  - The Observer Pattern
  - The Proxy Pattern
- Design Patterns Tutorials

Another Example: The Strategy Pattern template

The Strategy Pattern: Key Features

Intent

Allows you to use different business rules or algorithms depending upon the context in which they occur.

Problem

The selection of an algorithm that needs to be applied depends upon the client making the request or the data being acted upon. If you simply have a rule in place that does not change, you do not need a Strategy pattern.

Solution

Separates the selection of algorithm from the implementation of the algorithm. Allows for the selection to be made based upon context.

Participants and Collaborators

- The `Strategy` specifies how the different algorithms are used.
- The `ConcreteStrategies` implement these different algorithms.
- The `Context` uses the specific `ConcreteStrategy` with a reference of type `Strategy`. The `Strategy` and `Context` interact to implement the chosen algorithm (sometimes the `Strategy` must query the `Context`). The `Context` forwards requests from its `Client` to the `Strategy`.

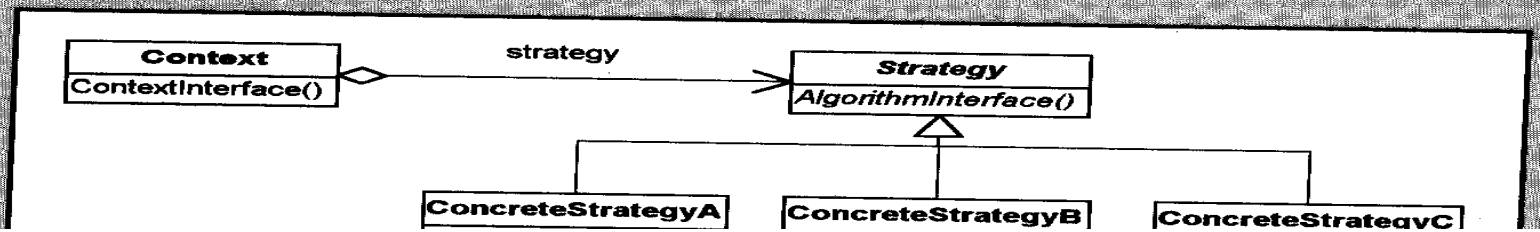
Consequences

- The Strategy pattern defines a family of algorithms.
- Switches and/or conditionals can be eliminated.
- You must invoke all algorithms in the same way (they must all have the same interface). The interaction between the `ConcreteStrategies` and the `Context` may require the addition of *`getState`* type methods to the `Context`.

Implementation

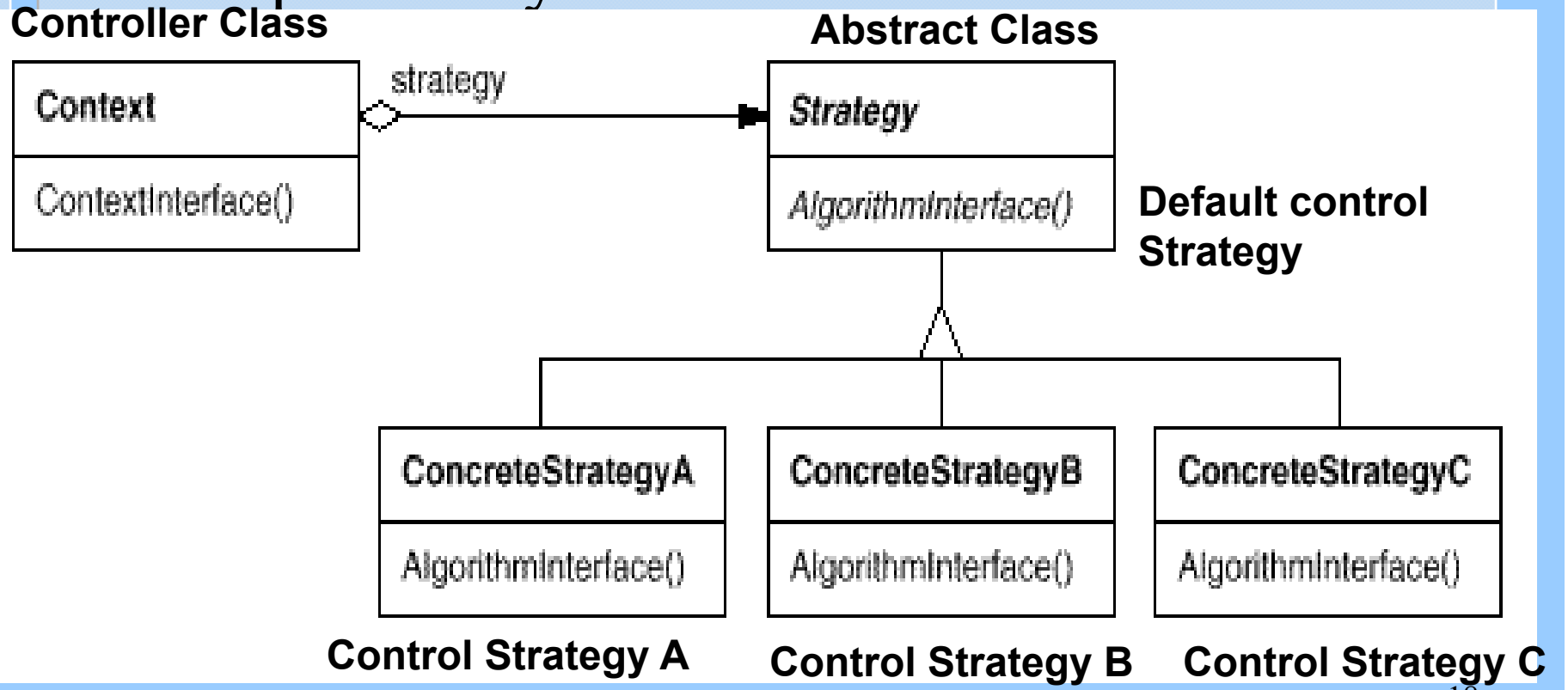
Have the class that uses the algorithm (the `Context`) contain an abstract class (the `Strategy`) that has an abstract method specifying how to call the algorithm. Each derived class implements the algorithm as needed. *Note:* this method wouldn't be abstract if you wanted to have some default behavior.

Note: In the prototypical Strategy pattern, the responsibility for selecting the particular implementation to use is done by the `Client` object and is given to the context of the `Strategy` pattern.



Another Example of Design Patterns

- The Strategy Pattern: lets the algorithm vary independently from clients that use it

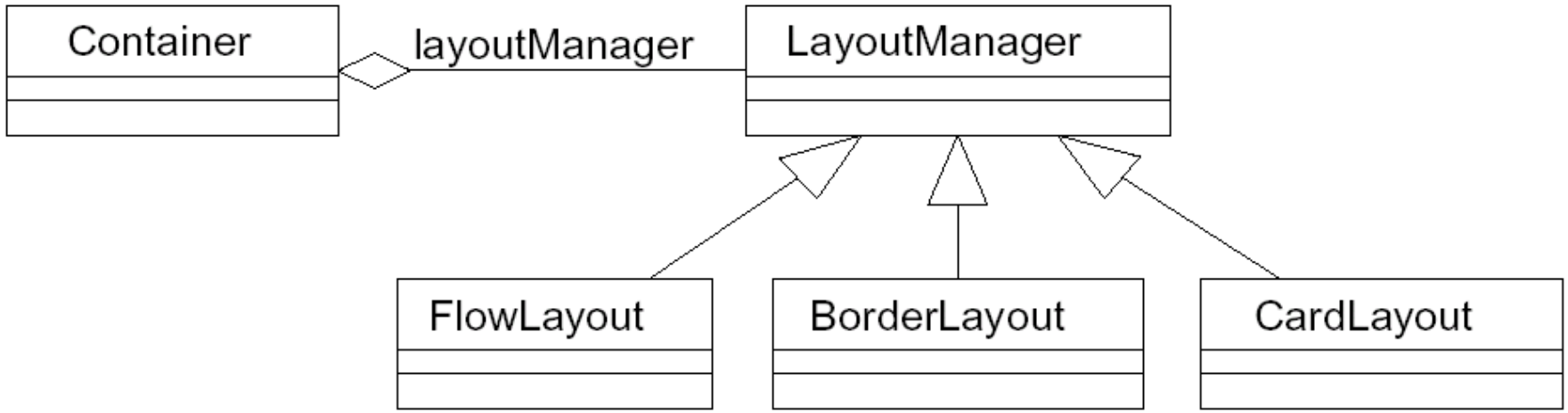


The Strategy Pattern

- The Strategy Pattern Context class has multiple control strategies provided by the concrete strategy classes, or by the abstract strategy (by default)
- The pattern lets us vary the algorithm that implements a certain function during run time depending on the conditions of the system
- The Pattern reduces coupling by having the client class be coupled only to the context class

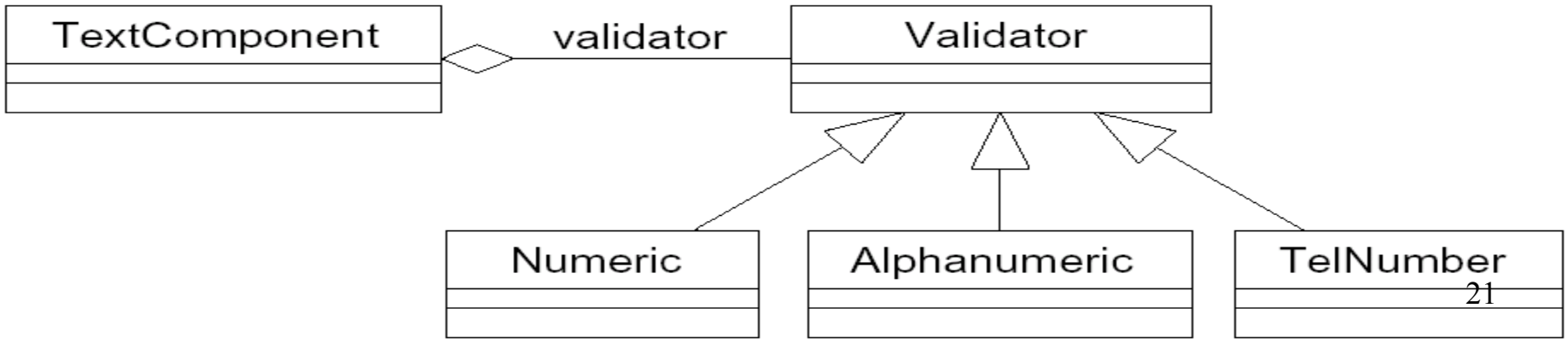
Examples of Design Patterns
The Strategy Pattern

- Example of using the pattern in JAVA AWT GUI components lay out managers



Examples of Design Patterns
The Strategy Pattern: another example

- Situation: A GUI text component object wants to decide at runtime what strategy it should use to validate user input. Many different validation strategies are possible: numeric fields, alphanumeric fields, telephone-number fields, etc.



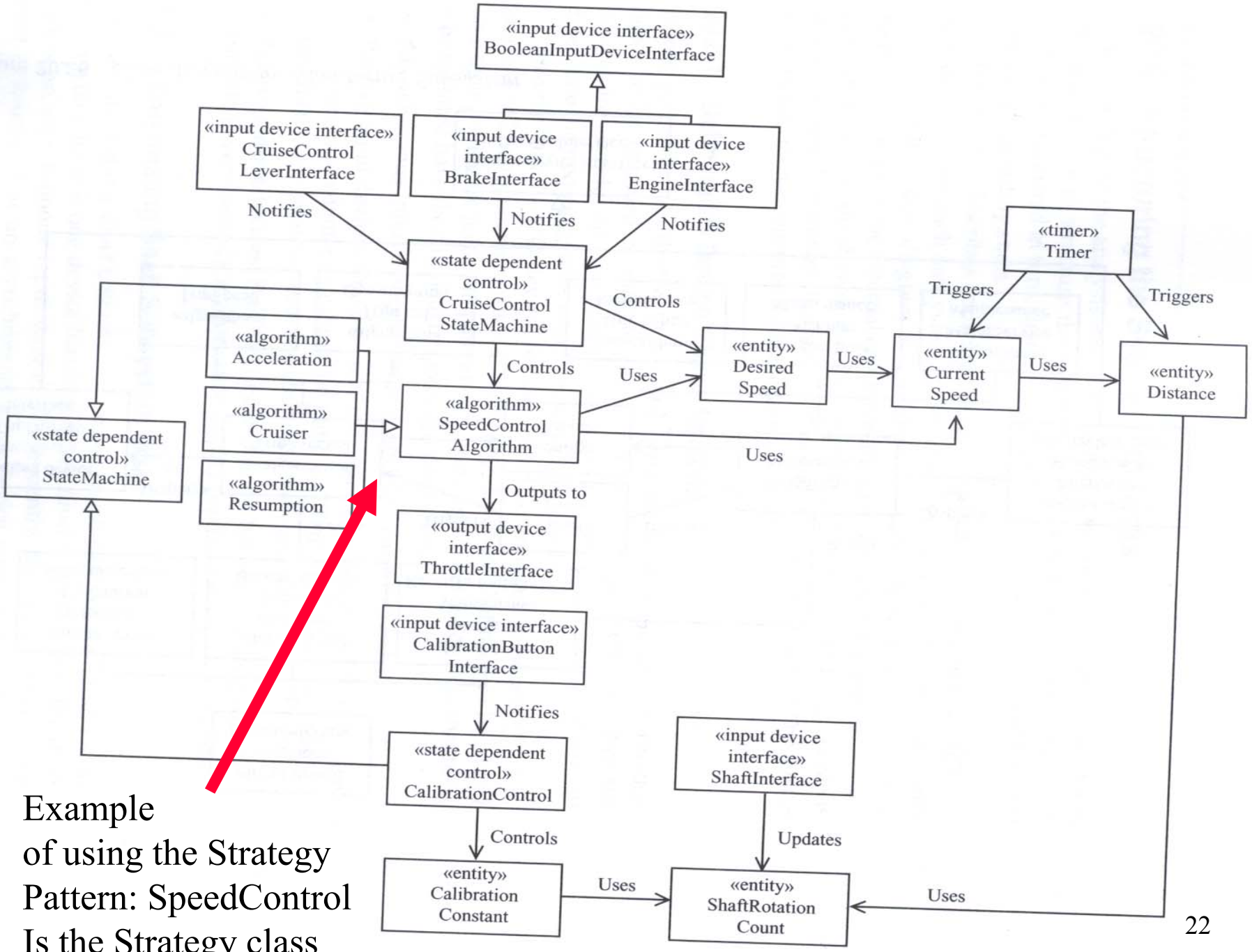
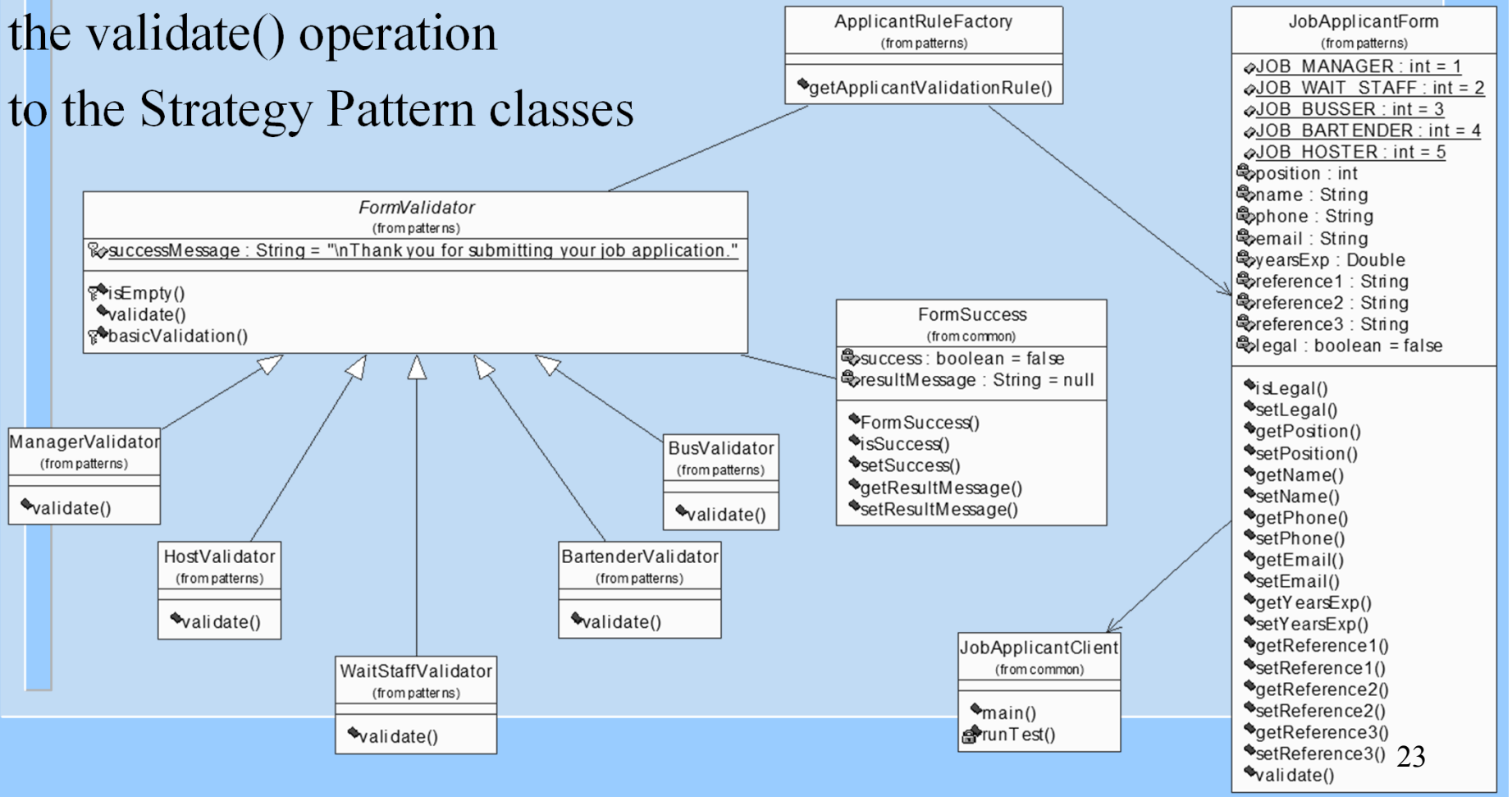


Figure 20.25 *Class diagram for Cruise Control Subsystem*

Example
of using the Strategy
Pattern: SpeedControl
Is the Strategy class

Another example of using the Strategy Pattern: A Job Application System

- The complexity of class JobApplication is reduced by moving the validate() operation to the Strategy Pattern classes

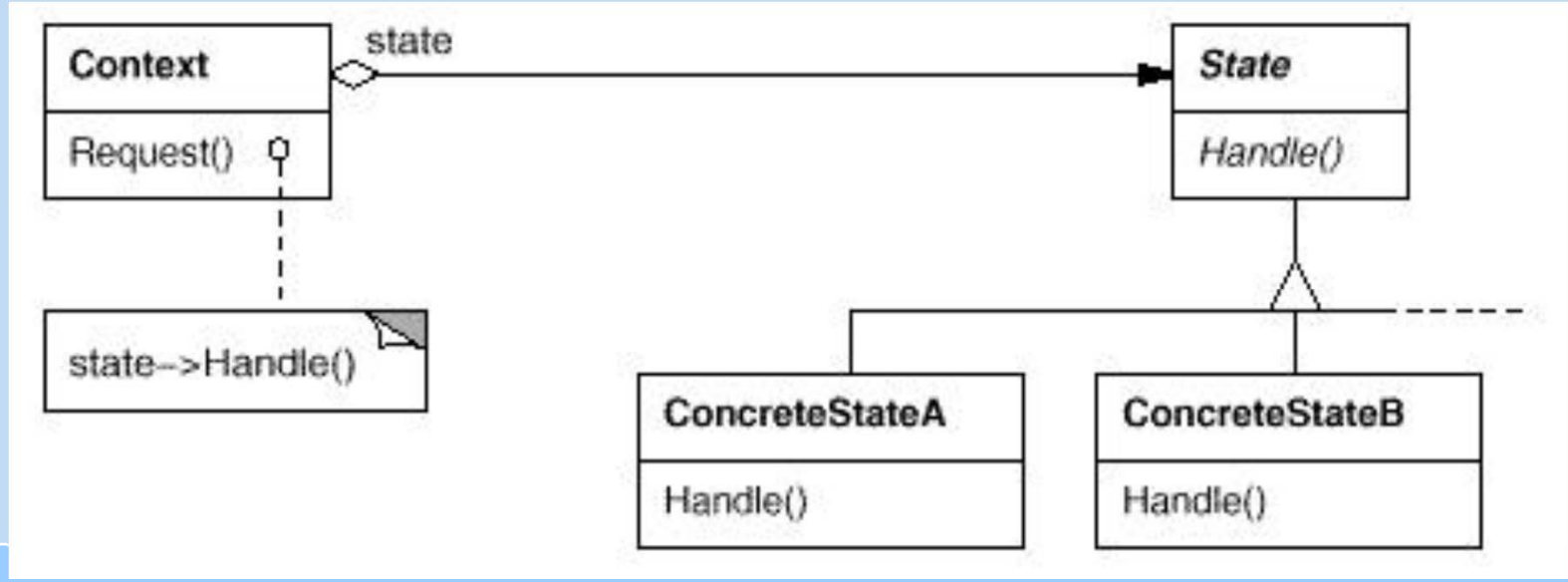


Outline

- The Requirements, Analysis, Design, and Design Refinement Models
- Class diagram refinement using design patterns
- Design patterns examples
  - The Facade pattern
  - The Strategy Pattern
  - The State Pattern
  - The Command Pattern
  - The Observer Pattern
  - The Proxy Pattern
- Design Patterns Tutorials

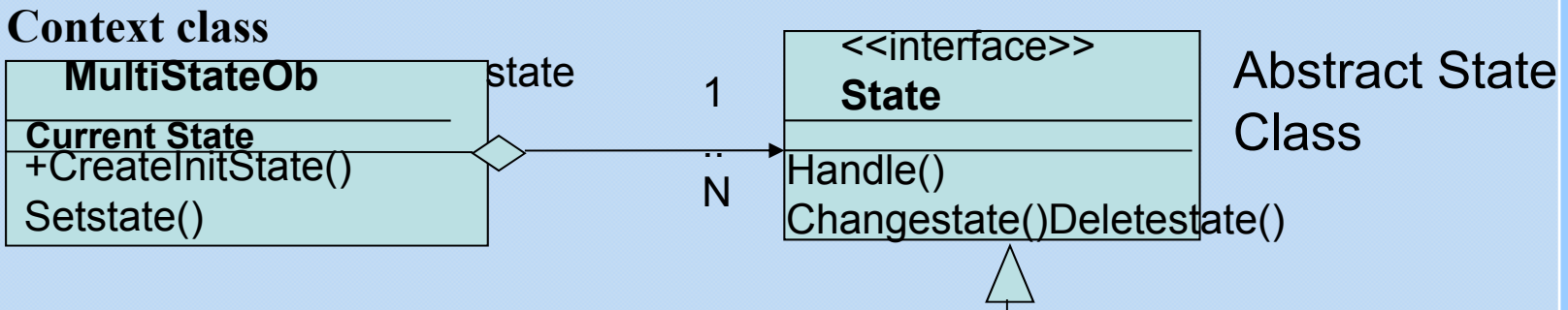
Examples of Design Patterns
The State Pattern

- Similar in structure (static) to the Strategy pattern but differs in dynamics
- Events are handled based on the curren state of the object



Examples of Design Patterns
The State Pattern

- The State Pattern: is a solution to the problem of how to make the behavior of an object depend on its state.



The Context Class

Abstract State Class

Lets a mutli state class divide its responsibilities (Opr1(),Opr2(), and Oprn() on multiple state classes.

N concrete state classes

For more Info, see http://home.earthlink.net/~huston2/dp/state.html

Examples of Design Patterns
The State Pattern

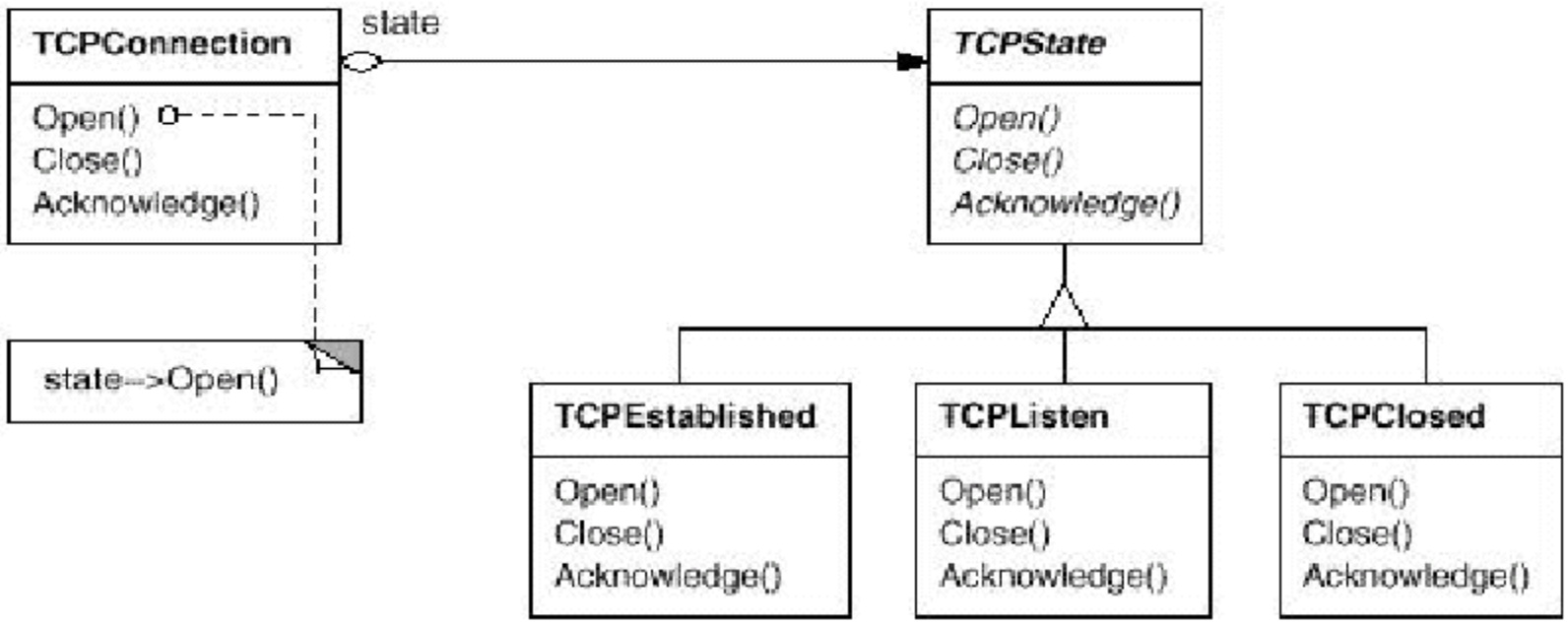
- The State pattern is a similar in structure to the Strategy Pattern but with different behavior or dynamics. the state objects are active one at a time depending on the actual state of the context object.
- The structure is defined as follows:
  - Define a "context" class to present a single interface to the outside world.
  - Define a State abstract base class.
  - Represent the different "states" of the state machine as derived classes of the State base class.
  - Define state-specific behavior in the appropriate State derived classes.
  - Maintain a pointer to the current "state" in the "context" class.
  - To change the state of the state machine, change the current "state" pointer
- State Transitions can be defined for each State class
- To be discussed later at length in slides 10 on

Examples of Design Patterns

- The context class Multistateob would create the initial state object to provide the services of the initial state (it will set its current state to its initial state)
- The initial state object would sense the condition for state transition to a new state, when this occurs it would then create an object of the new state and destroy itself
- Each state object implements the transition, actions, and activities in the state it represents

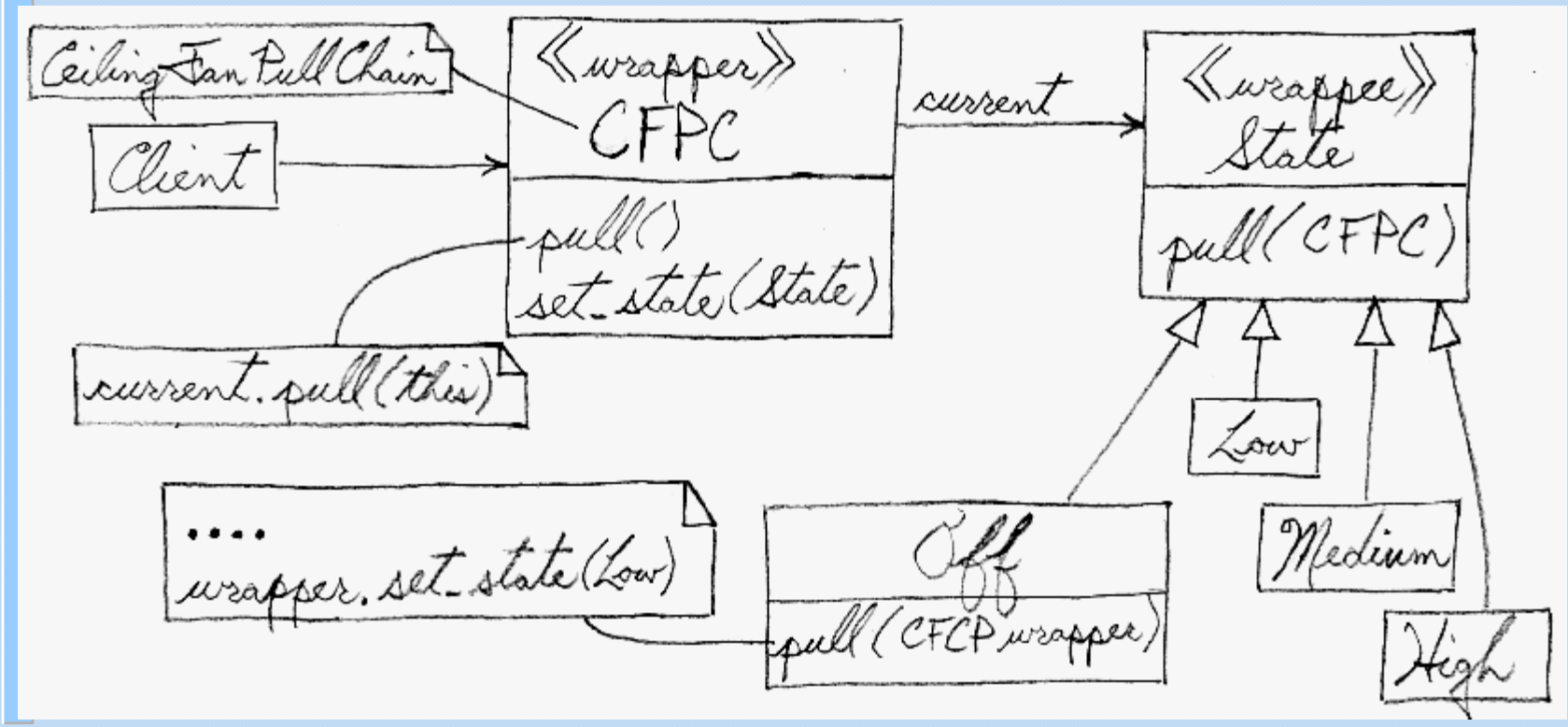
Examples of Design Patterns
The State Pattern

- TCP connection example



Examples of Design Patterns
The State Pattern

A Ceiling Fan Pull Chain Example :



Consolidated Collaboration Diagram of the ATM Client Subsystem

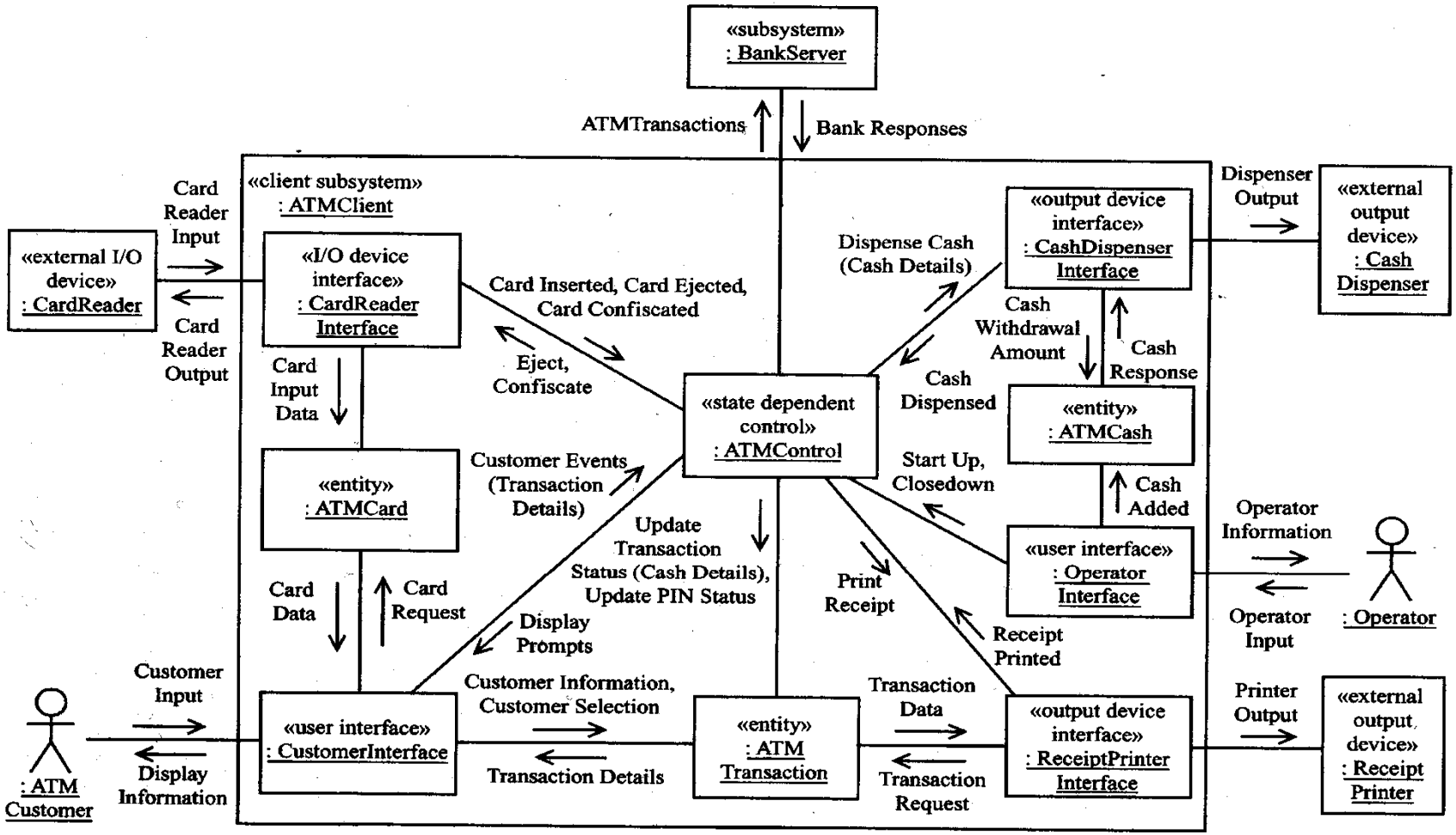


Figure 12.5 *Example of consolidated collaboration diagram: ATM Client subsystem*

Example: How can we apply The State Pattern to the ATM system using This ATM controller StateCharts ? See Next Slides On State Charts Pattern

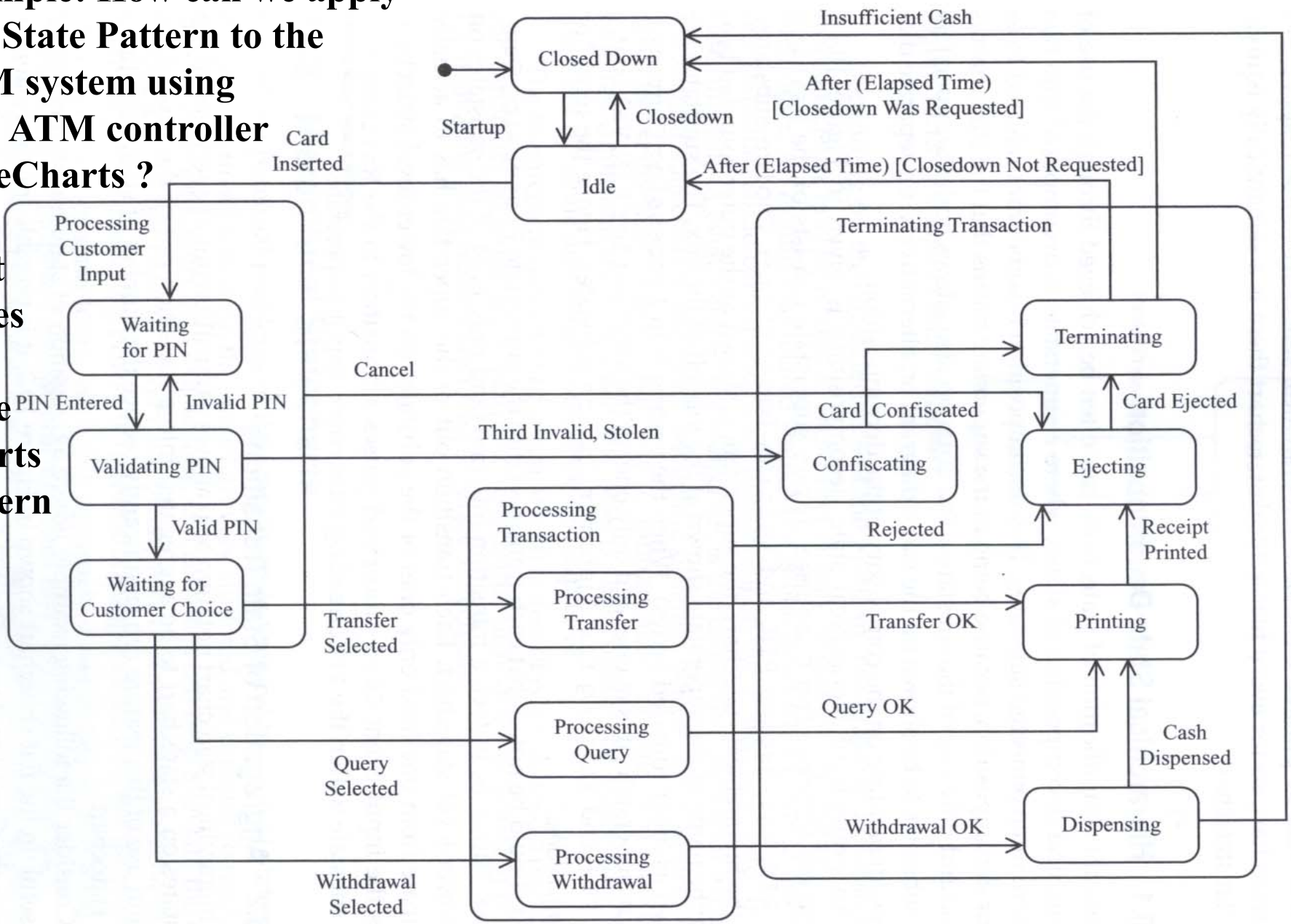


Figure 10.14 *Example of hierarchical statechart*

Outline

- The Requirements, Analysis, Design, and Design Refinement Models
- Class diagram refinement using design patterns
- Design patterns examples
  - The Facade pattern
  - The Strategy Pattern
  - The State Pattern
  - The Command Pattern
  - The Observer Pattern
  - The Proxy Pattern
- Design Patterns Tutorials

Examples of Behavioral Design Patterns

The Command Pattern: operator commands or user or customer requests are treated as a class of objects

Command object behaviora

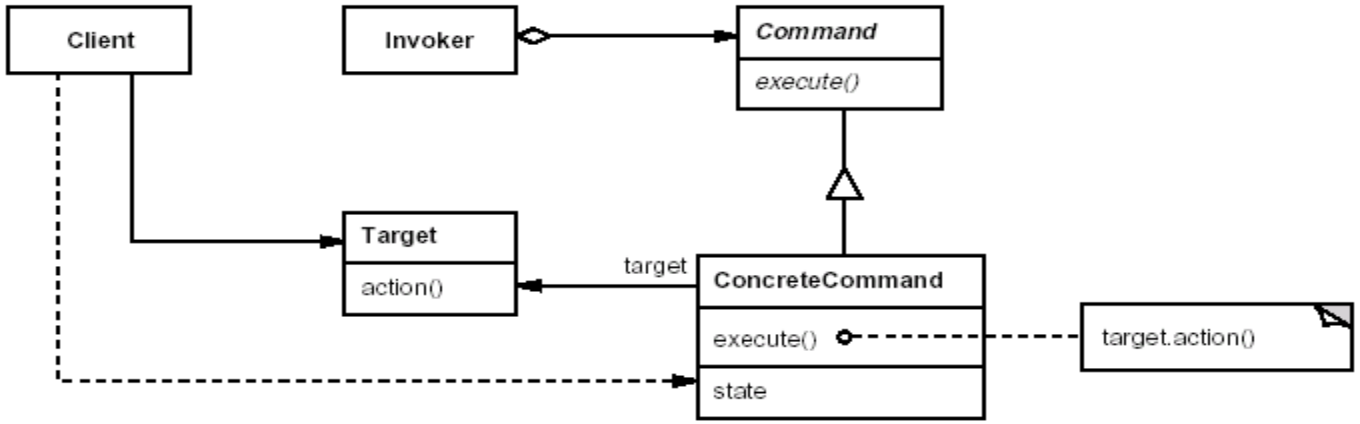
Intent

encapsulate the request for a service

Applicability

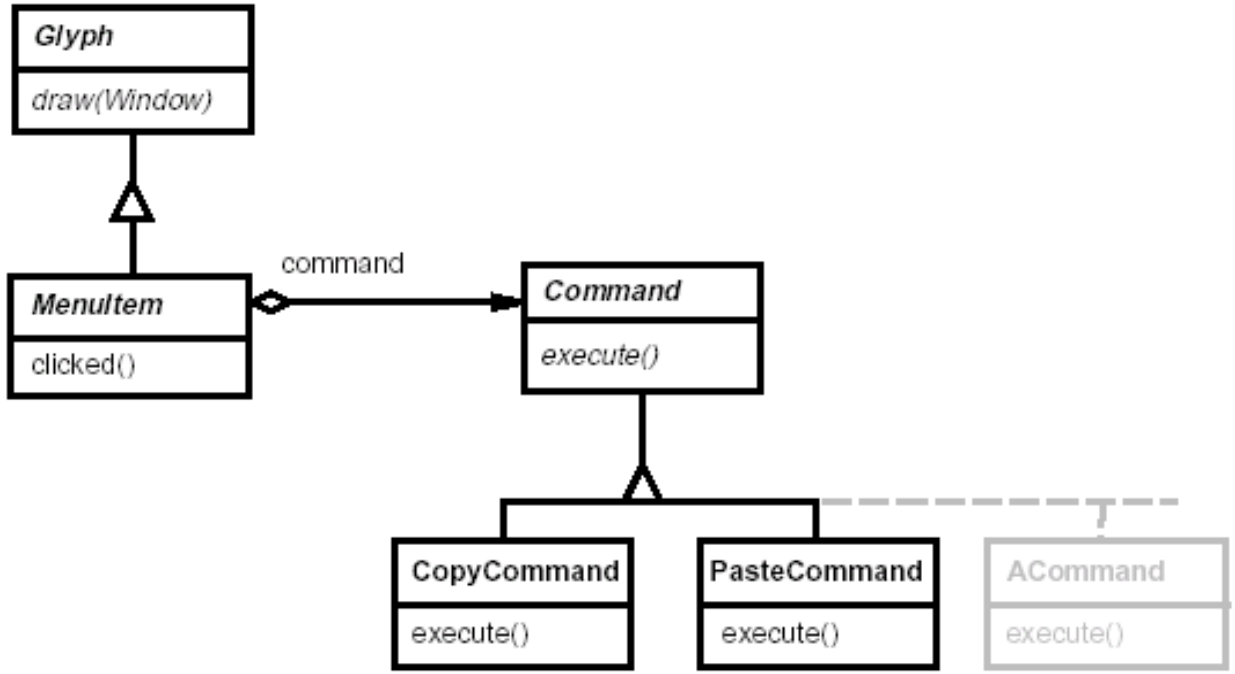
- to parameterize objects with an action to perform
- to specify, queue, and execute requests at different times
- for a history of requests
- for multilevel undo/redo

Structure



The Command Pattern

- Example of using the Command Pattern in a Menu driven graphics application



Outline

- The Requirements, Analysis, Design, and Design Refinement Models
- Class diagram refinement using design patterns
- Design patterns examples
  - The Facade pattern
  - The Strategy Pattern
  - The State Pattern
  - The Command Pattern
  - The Observer Pattern
  - The Proxy Pattern
- Design Patterns Tutorials

Examples of Behavioral Design Patterns

The Observer Pattern: Multiple observer objects are notified when changes of states subjects occur

Observer — object behavioral

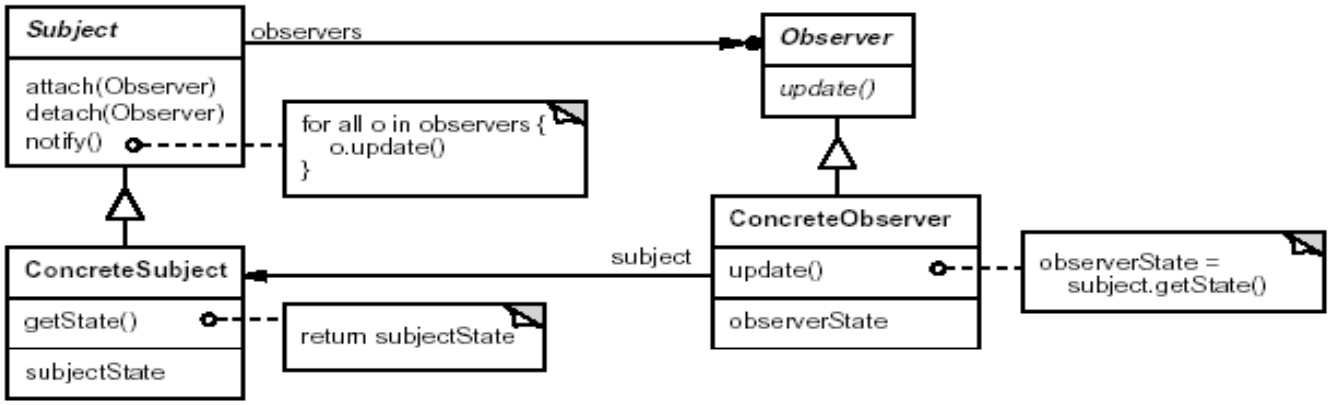
Intent

define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically

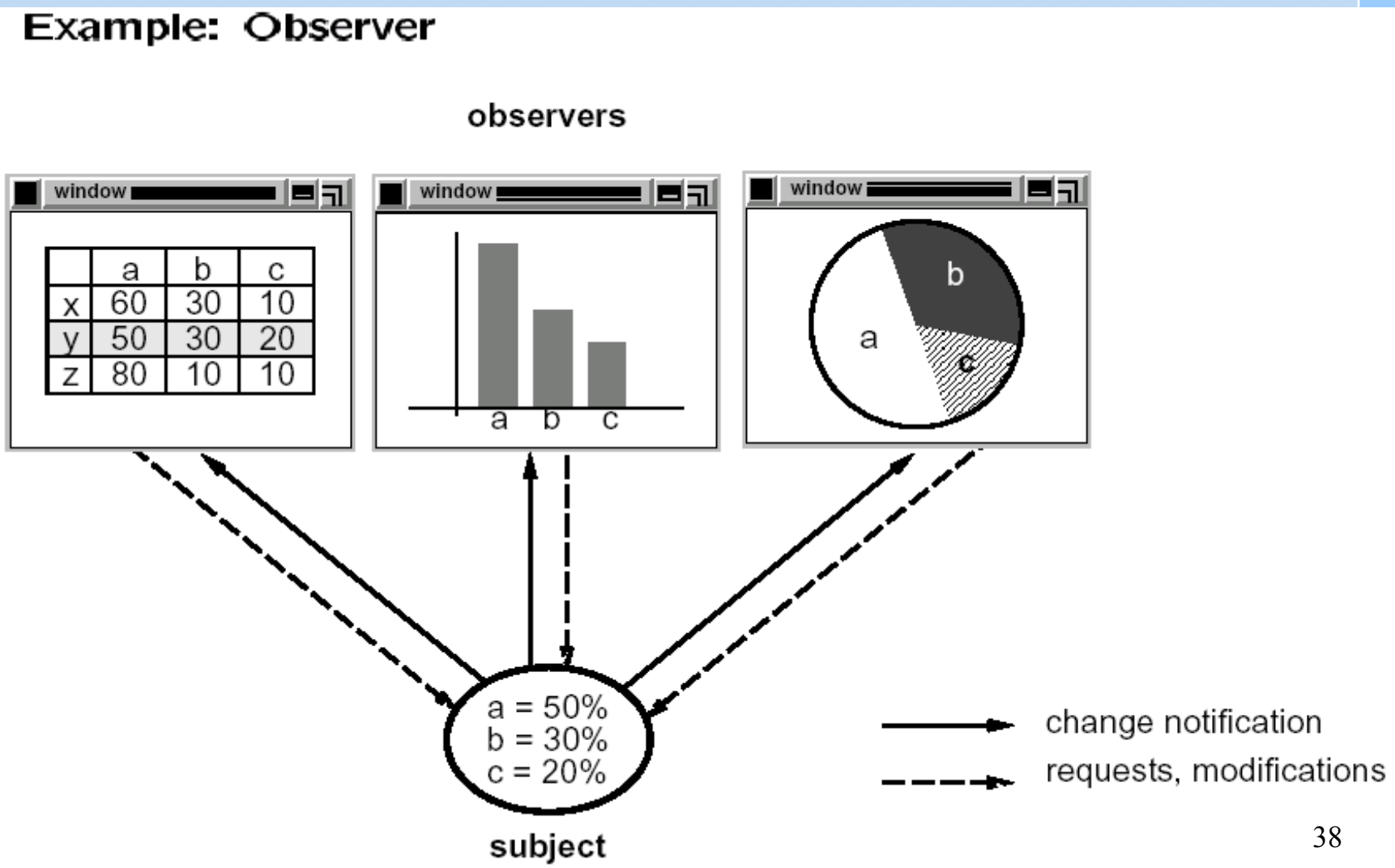
Applicability

- when an abstraction has two aspects, one dependent on the other
- when a change to one object requires changing others, and you don't know how many objects need to be changed
- when an object should notify other objects without making assumptions about who these objects are

Structure



The Observer Pattern



The Observer Pattern

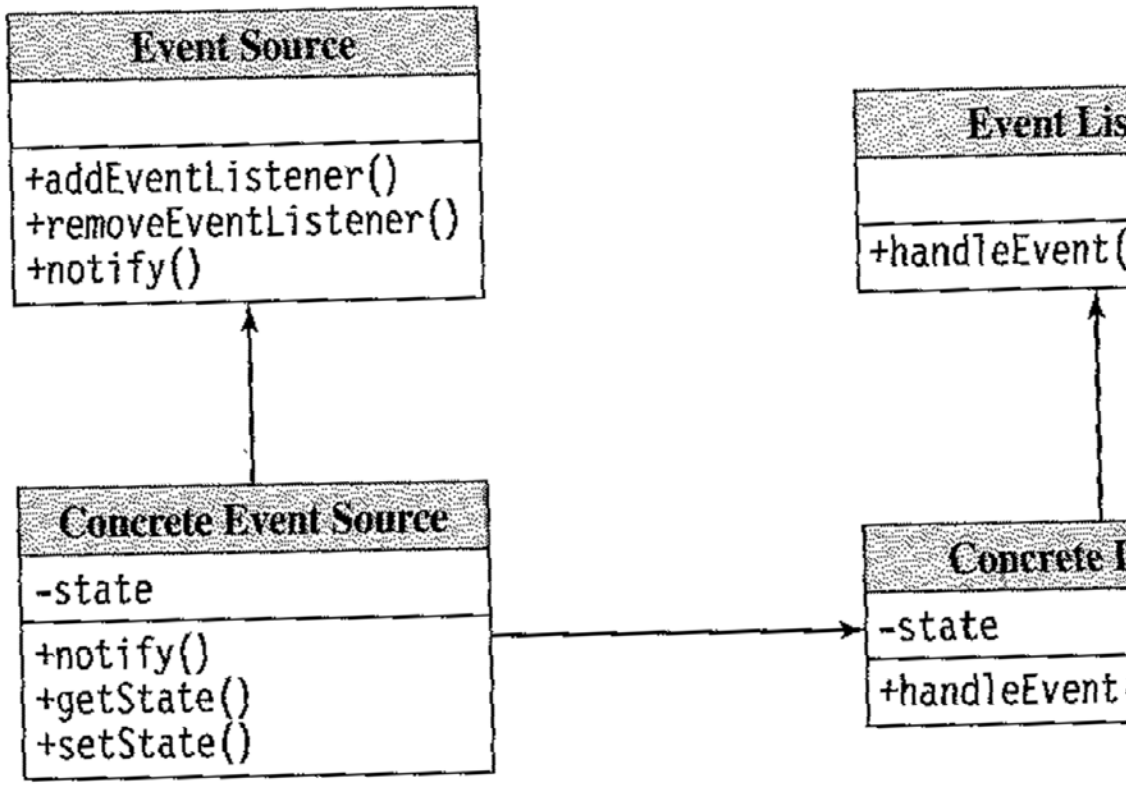


Figure 8.3
Class diagram for event-based implicit invocation architecture

Model-View-Controller Architecture Style

is based on the Observer pattern

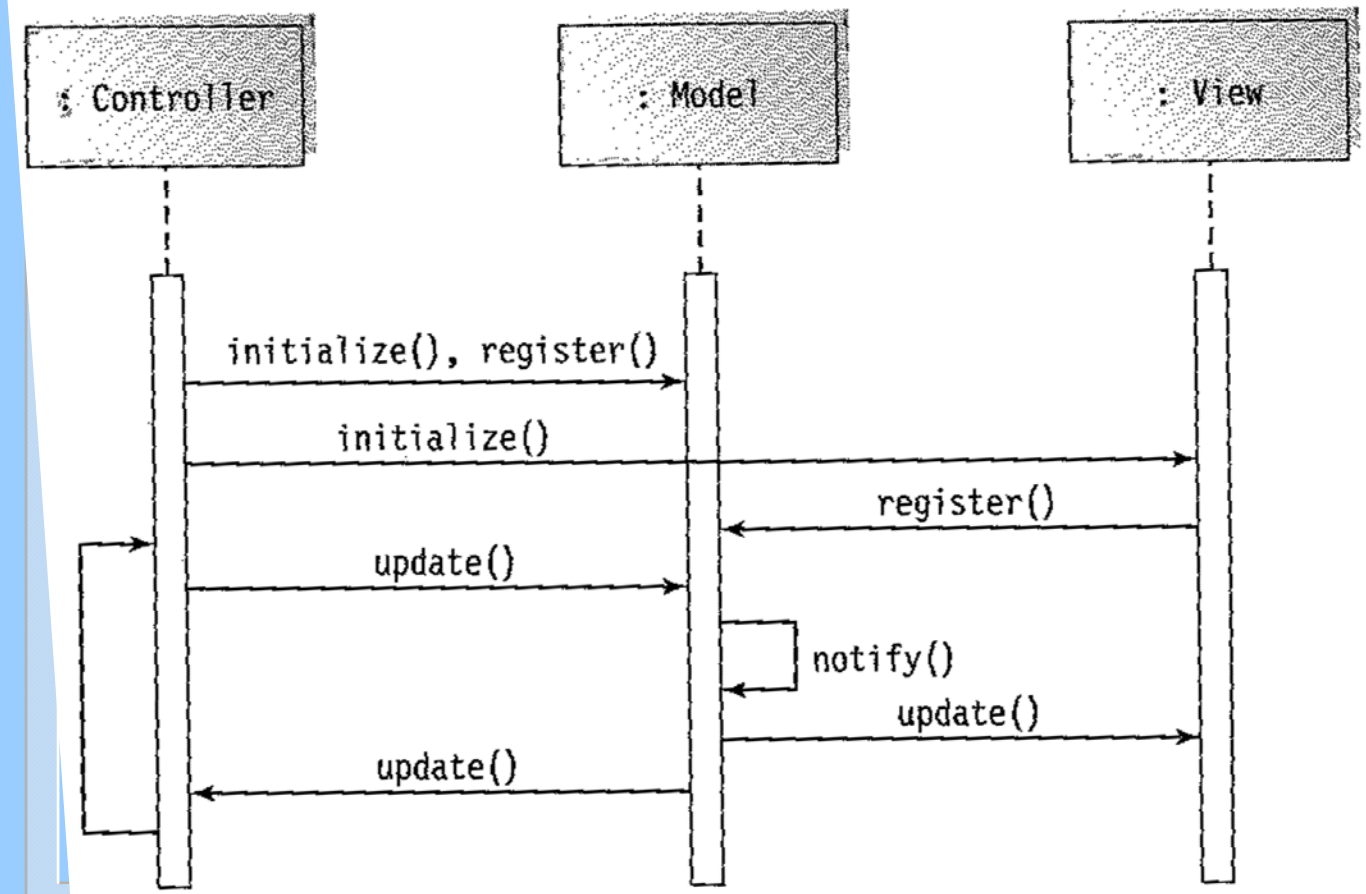


Figure 9.4
Sequence diagram for MVC architecture

Digital Sound Recorder: A Complete Example

- A Scheduler subsystem is added to provide interrupt Handling for timer interrupts to alert observers for synchronous tasks
- Uses the observer design pattern (to be discussed later)

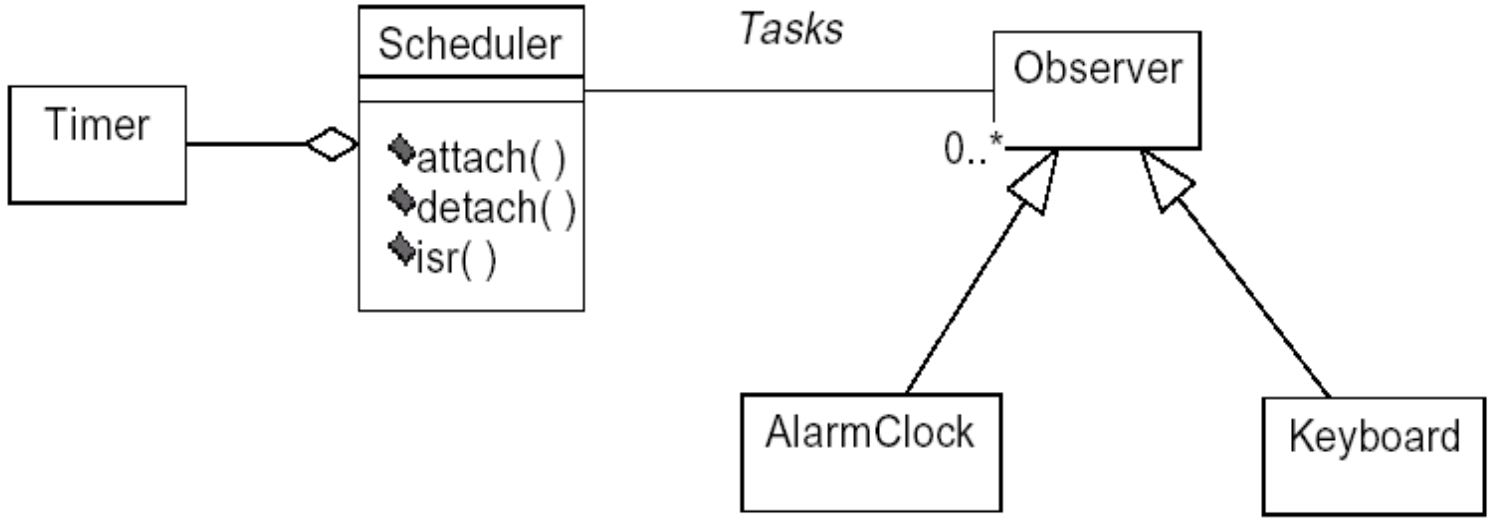
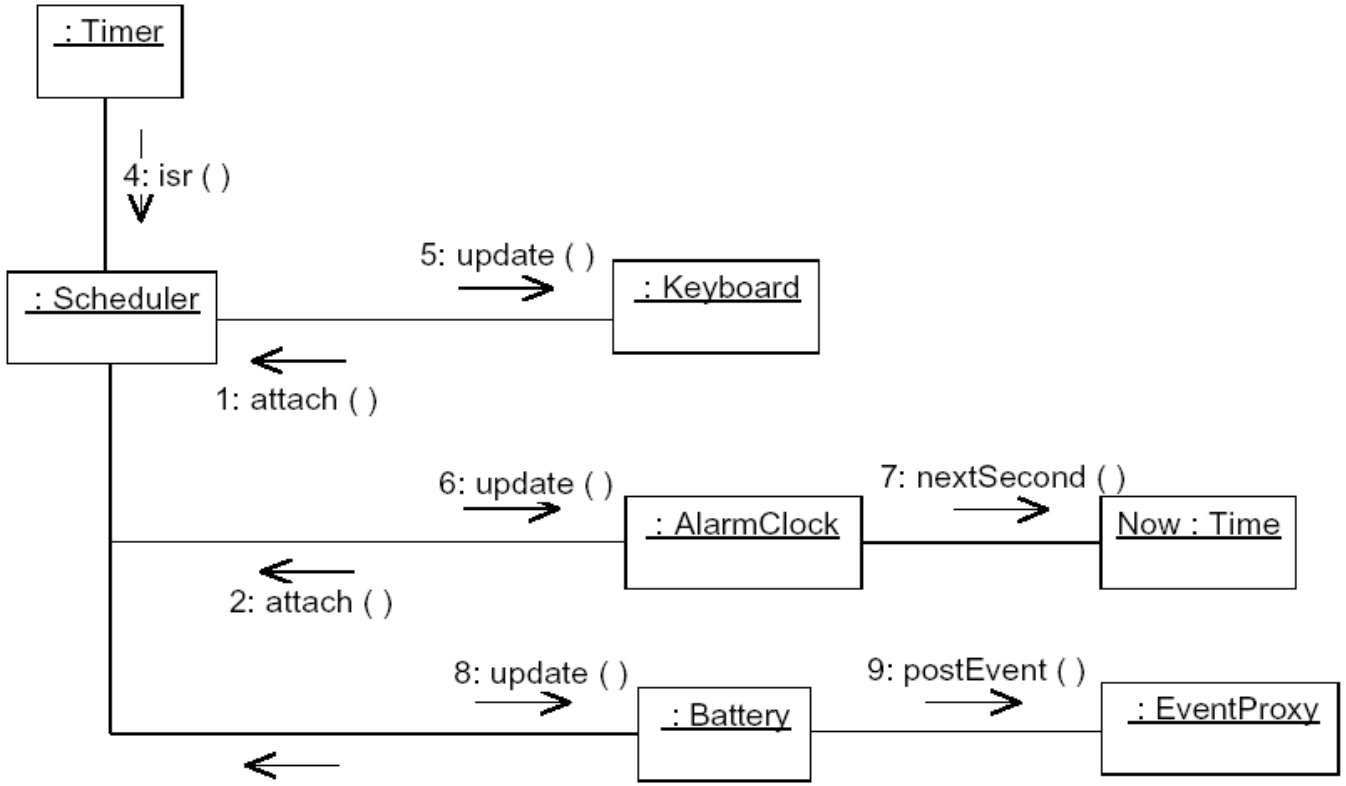


Figure 6.1: Scheduler class diagram

Digital Sound Recorder: A Complete Example: The Dynamic model

- Interactions are shown using a UML collaboration diagram. Timer interrupt update scenario

Notice an EventProxy Class is added For posting Events.
Uses the Proxy Desgin pattern



Outline

- The Requirements, Analysis, Design, and Design Refinement Models
- Class diagram refinement using design patterns
- Design patterns examples
  - The Facade pattern
  - The Strategy Pattern
  - The State Pattern
  - The Command Pattern
  - The Observer Pattern
  - The Composite Pattern
  - The Proxy Pattern
- Design Patterns Tutorials

Examples of Structural Design Patterns
The Composite Pattern

Composite object structural

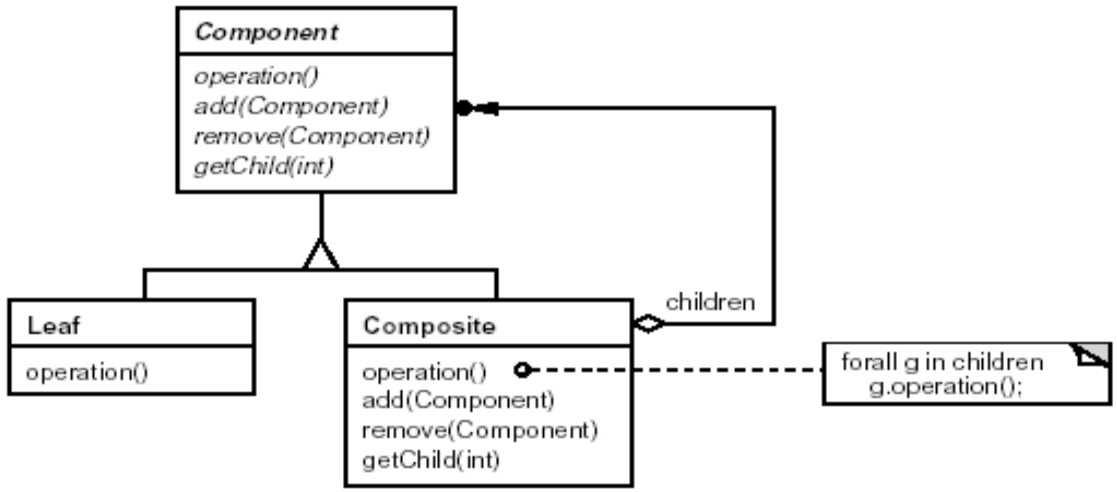
Intent

treat individual objects and multiple, recursively-composed objects uniformly

Applicability

objects must be composed recursively,
and there should be no distinction between individual and composed elements,
and objects in the structure can be treated uniformly

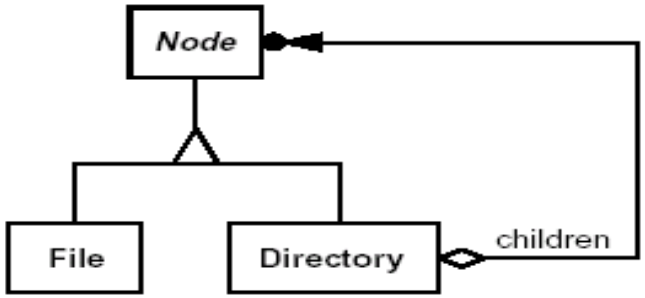
Structure



Examples of Design Patterns
The Composite Pattern :File System Structure

Mapping COMPOSITE participants to file system classes:

- Leaf, for objects that have no children
  → `File`, the file object
- Composite, for objects that have children
  → `Directory`, the directory object
- Component, the uniform interface
  → `Node`



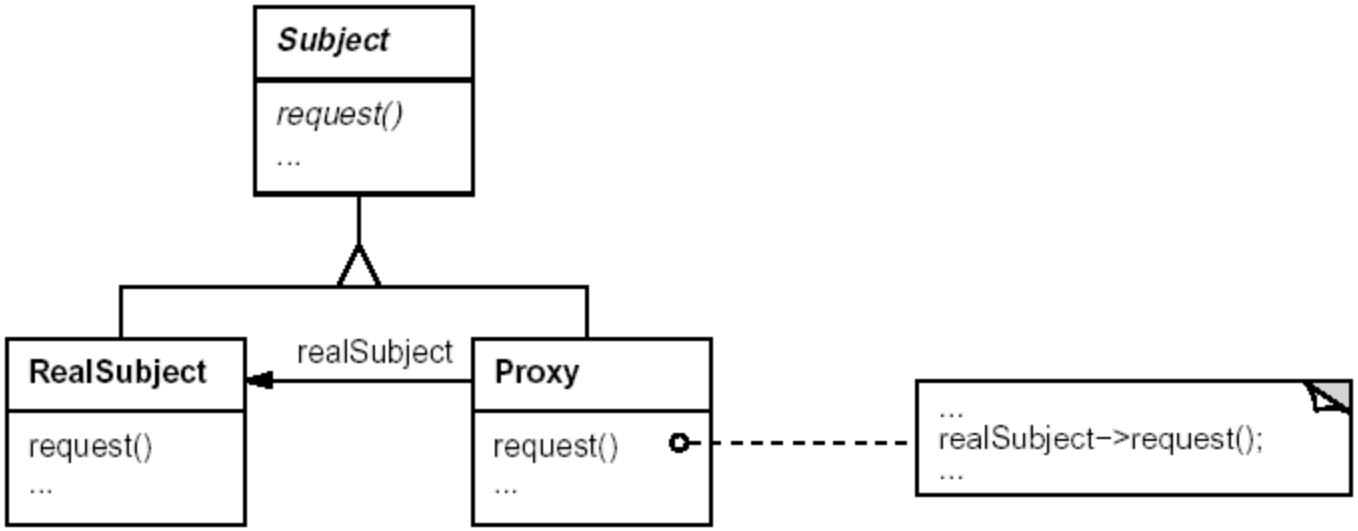
Outline

- The Requirements, Analysis, Design, and Design Refinement Models
- Class diagram refinement using design patterns
- Design patterns examples
  - The Facade pattern
  - The Strategy Pattern
  - The State Pattern
  - The Command Pattern
  - The Observer Pattern
  - The Proxy Pattern
- Design Patterns Tutorials

Examples of Structural Design Patterns

The Proxy Pattern (used heavily in communication software, CORBA, SOA)

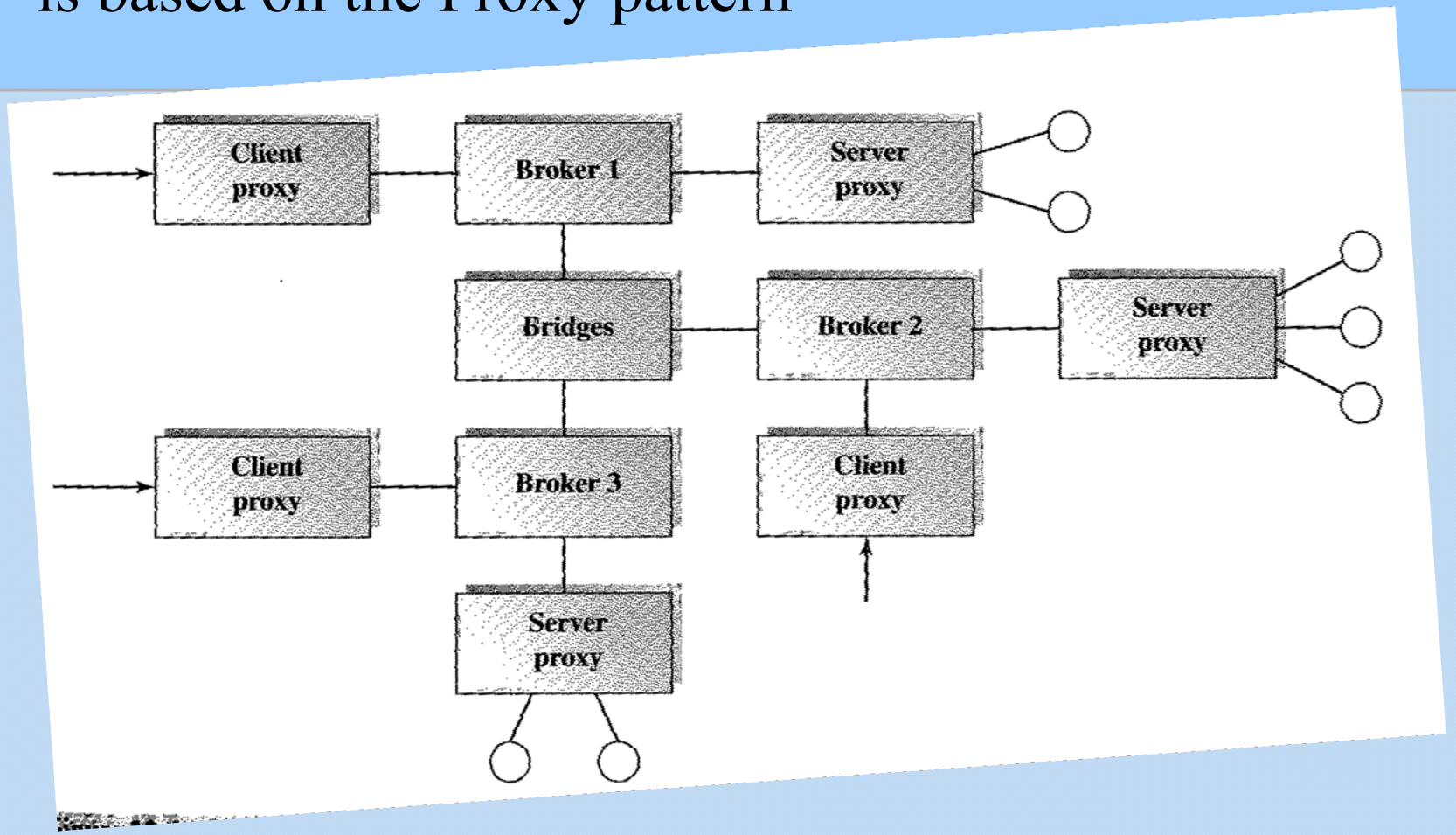
PROXY structure



- Proxy is a stand-in for RealSubject
- Proxy must match Subject interface

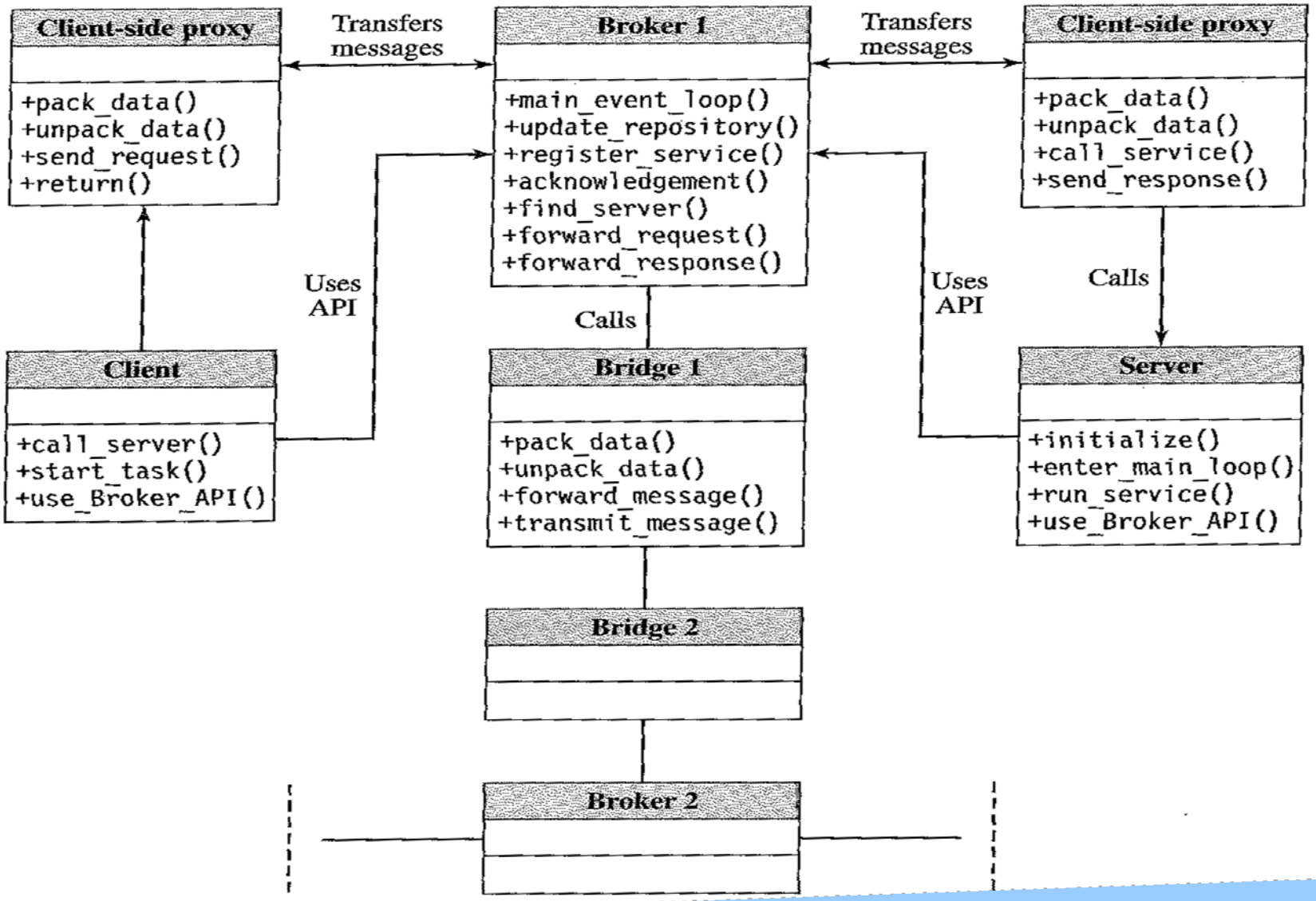
Broker Architecture Style

is based on the Proxy pattern

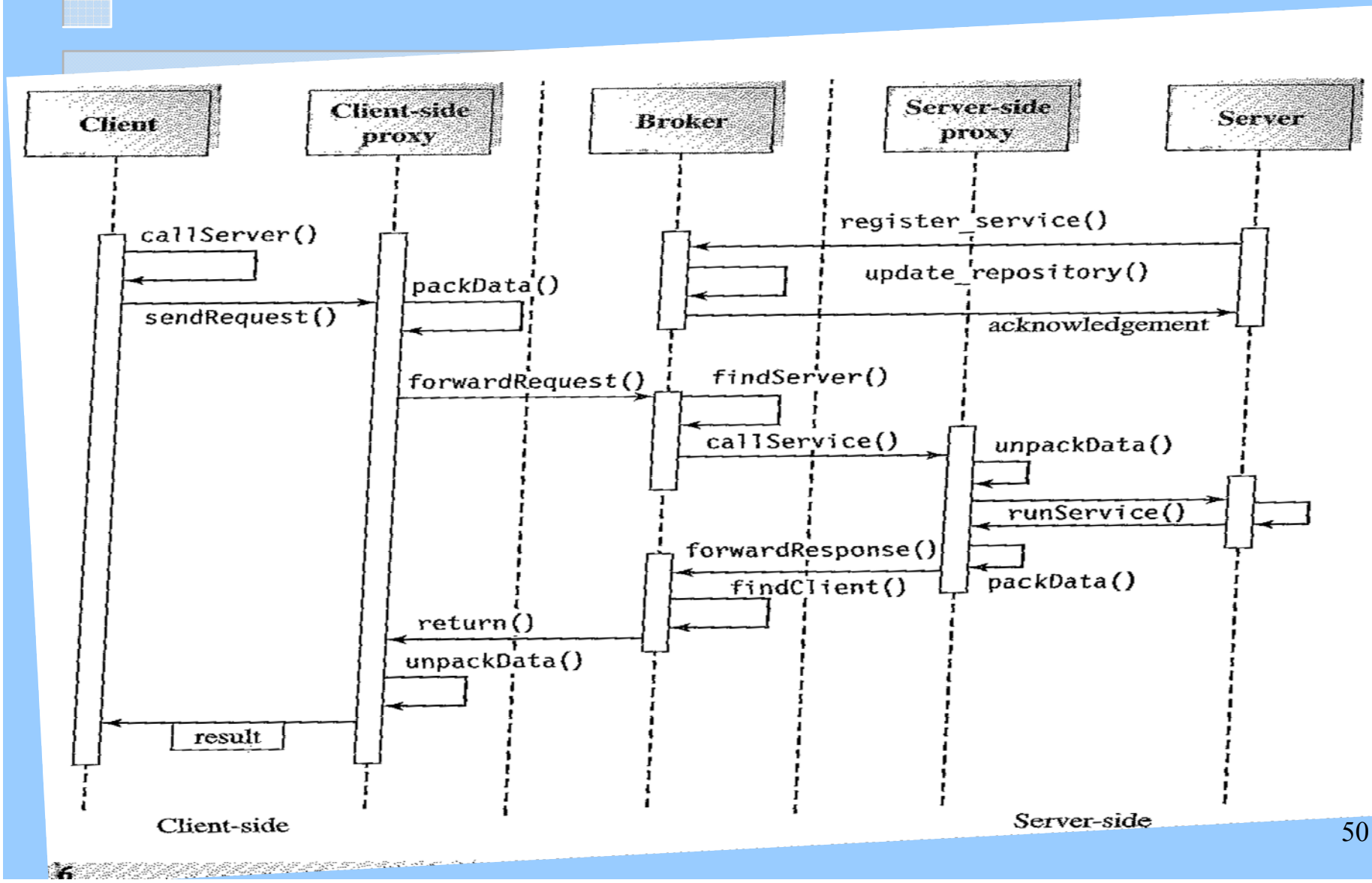


Brokers gets requests from client proxies and manages them by forwarding to server Proxies or dispatches them to other connected brokers

Broker Architecture Style



Broker Architecture Style



Design Patterns Examples and Tutorials

- **Strategy and State Patterns (important)**
- Two tutorials by John Vlissides
  - **An Introduction to Design Patterns**
    Also on the design patterns CD by Gama et al
  - **Designing with Patterns**