Object-Oriented Analysis and Design

Lecture 8: Identify Design Mechanisms
Objectives: Identify Design Mechanisms

- Define the purpose of the Identify Design Mechanisms activity and explain when in the lifecycle it is performed
- Explain what design and implementation mechanisms are and how they map from Analysis mechanisms
- Describe some key mechanisms that will be utilized in the case study
Identify Design Mechanisms in Context

- [Early Elaboration Iteration]
- Define a Candidate Architecture
- Perform Architectural Synthesis
- Analyze Behavior
- Refine the Architecture
  - Define Components
  - Design the Database
- [Inception Iteration (Optional)]
Identify Design Mechanisms Overview

Supplementary Specifications

Software Architecture Document

Analysis Class

Identify Design Mechanisms

Design Model
Identify Design Mechanisms: Steps

- Categorize clients of analysis mechanisms
- Document architectural mechanisms
Identify Design Mechanisms: Steps

⃣ Categorize clients of analysis mechanisms

• Documenting architectural mechanisms
Review: Patterns and Frameworks

- **Pattern**
  - Provides a common solution to a common problem in a context

- **Analysis/Design Pattern**
  - Provides a solution to a narrowly scoped technical problem
  - Provides a fragment of a solution, or a piece of the puzzle

- **Framework**
  - Defines the general approach to solving the problem
  - Provides a skeletal solution, whose details may be analysis/design patterns
What Is a Design Pattern?

- A design pattern provides a scheme for refining the subsystems or components of a software system, or the relationships between them. It describes a commonly-recurring structure of communicating components that solves a general design problem within a particular context.

  Erich Gamma et al. 1994. *Design Patterns—Elements of Reusable Object-Oriented Software*
## Examples of Pattern Usage

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Command</strong> (behavioral pattern)</td>
<td>Issue a request to an object without knowing anything about the operation requested or the receiver of the request: for example, the response to a menu item, an undo request, the processing of a time-out</td>
</tr>
<tr>
<td><strong>Abstract factory</strong> (creational pattern)</td>
<td>Create GUI objects (buttons, scrollbars, windows, etc.) independent of the underlying OS: the application can be easily ported to different environments</td>
</tr>
<tr>
<td><strong>Proxy</strong> (structural pattern)</td>
<td>Handle distributed objects in a way that is transparent to the client objects (<em>remote proxy</em>)&lt;br&gt;Load a large graphical object or any entity object “costly” to create/initialize only when needed (<em>on demand</em>) and in a transparent way (<em>virtual proxy</em>)</td>
</tr>
<tr>
<td><strong>Observer</strong> (behavioral pattern)</td>
<td>When the state of an object changes, the dependent objects are notified. The changed object is independent of the observers.&lt;br&gt;Note: The MVC architectural pattern is an extension of the Observer design pattern</td>
</tr>
</tbody>
</table>
Detailing the Command Pattern (cont.)

```java
Application

Menu
+menu 1
+items 0..*

MenuItem
- label : String
+ Clicked()

cmd.Process();

Command
+ Process()

menu
items

1
0..*
1
```
Detailing the Command Pattern (cont.)

Application

Menu

+menu 1

Command

+cmd 1

OpenCommand

+ Process()

AskUser();
DoOpen();

Menu

MenuItem

- label : String

+ Clicked()

+items 0..*

cmd.Process();
The user selects the Open… menu item
Detailing the Command Pattern (cont.)

Application
- menu
  - AddItem(s : String, c : Command)

Menu
+ menu
  + AddItem(s : String, c : Command)

MenuItem
- label : String
+ Clicked()
  + MenuItem(s : String, c : Command)

Clicked()
  cmd.Process();

Menultem()
  cmd = c;
  label = s;

OpenCommand
+ Process()
+ OpenCommand()
- AskUser()
- DoOpen()

AskUser();
DoOpen();

Command
+ Process()
Detailing the Command Pattern (cont.)

```
app
- Application
- CloseCommand
- OpenCommand

com
  + Command

gui
  + Menu
  - MenuItem
```
A design pattern is a parameterized collaboration:

Representing Design Patterns in UML
Describing Analysis Mechanisms

- Collect all analysis mechanisms in a list
- Draw a map of the client classes to the analysis mechanisms

<table>
<thead>
<tr>
<th>Analysis Class</th>
<th>Analysis Mechanism(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>Persistence, Security</td>
</tr>
<tr>
<td>Schedule</td>
<td>Persistence, Security</td>
</tr>
<tr>
<td>CourseOffering</td>
<td>Persistence, Legacy Interface</td>
</tr>
<tr>
<td>Course</td>
<td>Persistence, Legacy Interface</td>
</tr>
<tr>
<td>RegistrationController</td>
<td>Distribution</td>
</tr>
</tbody>
</table>
Categorize Analysis Mechanisms

♦ Purpose
  ▪ To refine the information gathered on the analysis mechanisms

♦ Steps
  ▪ Identify the clients of each analysis mechanism
  ▪ Identify characteristic profiles for each analysis mechanism
  ▪ Group clients according to their use of characteristic profiles
  ▪ Proceed bottom up and make an inventory of the implementation mechanisms that you have at your disposal
Identify Design Mechanisms: Steps

- Categorize clients of analysis mechanisms
- Documenting architectural mechanisms
Design and Implementation Mechanisms

- **Analysis Mechanism (Conceptual)**
  - Persistency
  - Distribution

- **Design Mechanism (Concrete)**
  - RDBMS
  - OODBMS
  - Remote Method Invocation (RMI)

- **Implementation Mechanism (Actual)**
  - JDBC
  - ObjectStore
  - Java 1.2 from Sun
  - Legacy Data
  - New Data

**Tools and Concepts**
- OODBMS
- RDBMS
- JDBC
- ObjectStore
- Java 1.2 from Sun
Architectural mechanisms can be treated as patterns (i.e., parameterized collaboration).

Documented in Design Guidelines
Review: Course Registration Analysis Mechanisms

- Persistence
- Distribution
- Security
- Legacy Interface
Design Mechanisms: Persistency: RDBMS: JDBC

- Persistence characteristics:
  - Granularity
  - Volume
  - Duration
  - Access mechanism
  - Access frequency (creation/deletion, update, read)
  - Reliability

Note: JDBC is the standard Java API for talking to a SQL database.
Example: Persistency: RDBMS: JDBC

Roles to be filled by the designer applying the mechanism

<<role>> PersistencyClient (from SamplePersistencyClient)

<<role>> DBClass

create() : PersistentClass
read(searchCriteria : String) : PersistentClassList
update(c : PersistentClass)
delete(c : PersistentClass)

<<role>> PersistentClassList (from SamplePersistentClass)
new()
add(c : PersistentClass)

<<role>> PersistentClass (from SamplePersistentClass)
getData()
setData()
command()
new()

DriverManager (from java.sql)
getConnection(url, user, pass) : Connection

ResultSet (from java.sql)
getString() : String
executeQuery(sql : String) : ResultSet
executeUpdate(sql : String) : int

Connection (from java.sql)
createStatement() : Statement

<<role>>

0..*

Object Oriented Analysis and Design
Example: Persistency: RDBMS: JDBC: Initialize

1. getConnection(url, user, pass)
Example: Persistency: RDBMS: JDBC: Create

1. create()
   1.1. New()
   1.2. getData()
   1.3. createStatement()
   1.4. executeUpdate(String)
Example: Persistency: RDBMS: JDBC: Read

**PersistencyClient**

1. read(string)
   - 1.1. createStatement()
   - 1.2. executeQuery(string)

**DBClass**

1.3. new()

**Connection**

The SQL statement built by the DBClass using the given criteria is passed to executeQuery().

**Statement**

Create a list to hold all retrieved data.

**ResultSet**

1.4. new()
1.5. getString()
1.6. setData()
1.7. add(PersistentClass)

**PersistentClassList**

Repeat these operations for each element returned from the executeQuery() command. The PersistentClassList is loaded with the data retrieved from the database.

**PersistentClass**

Add the retrieved course offering to the list to be returned.
Example: Persistency: RDBMS: JDBC: Update

1. update(PersistentClass)
   1.1. getData()
   1.2. createStatement()
   1.3. executeUpdate(string)

execute SQL statement
Example: Persistency: RDBMS: JDBC: Delete

1. delete(PersistentClass)
   1.1. createStatement()
   1.2. executeUpdate(string)

execute SQL statement
Incorporating JDBC: Steps

- Provide access to the class libraries needed to implement JDBC
  - *Provide java.sql package*
- Create the necessary DBClasses
  - Assign one DBClass per persistent class
- Incorporate DBClasses into the design
  - Allocate to package/layer
  - Add relationships from persistency clients
- Create/Update interaction diagrams that describe:
  - Database initialization
  - Persistent class access: Create, Read, Update, Delete

Deferred
Example: Incorporating JDBC

Sample Persistency Client Package

java.sql

DriverManager
(from java.sql)

Connection
(from java.sql)

Statement
(from java.sql)

ResultSet
(from java.sql)
Review: Identify Design Mechanisms

- What does an analysis mechanism do?
- What is a pattern? What makes a framework different from a pattern?
- Why should you categorize analysis mechanisms? Identify the steps.