Object-Oriented Analysis and Design

Lecture 12: Subsystem Design
Objectives: Subsystem Design

- Describe the purpose of Subsystem Design and where in the lifecycle it is performed
- Define the behaviors specified in the subsystem's interfaces in terms of collaborations of contained classes
- Document the internal structure of the subsystem
- Determine the dependencies upon elements external to the subsystem
Subsystem Design in Context

- Define a Candidate Architecture
- Perform Architectural Synthesis
- Analyze Behavior
- Refine the Architecture
- Define Components
- Design the Database
Subsystem Design Overview

Project Specific Guidelines

Subsystem Design

Design Subsystems and Interfaces

Design Classes
A Subsystem:
- Is a “cross between” a package and a class
- Realizes one or more interfaces that define its behavior
Subsystem Guidelines

❖ Goals
- Loose coupling
- Portability, plug-and-play compatibility
- Insulation from change
- Independent evolution

❖ Strong Suggestions
- Do not expose details, only interfaces
- Depend only on other interfaces

Key is abstraction and encapsulation
Review: Modeling Convention for Subsystems and Interfaces

Interfaces start with an “I”

Interfaces are EXTERNAL to the subsystem.
Subsystem Design Steps

- Distribute subsystem behavior to subsystem elements
- Document subsystem elements
- Describe subsystem dependencies
- Checkpoints
Subsystem Design Steps

- Distribute subsystem behavior to subsystem elements
  - Document subsystem elements
  - Describe subsystem dependencies
  - Checkpoints
Subsystem Responsibilities

- Subsystem responsibilities defined by interface operations
  - Model interface realizations
- Interface operations may be realized by
  - Internal class operations
  - Internal subsystem operations
Distributing Subsystem Responsibilities

- Identify new, or reuse existing, design elements (for example, classes and/or subsystems)
- Allocate subsystem responsibilities to design elements
- Incorporate applicable mechanisms (for example, persistence, distribution)
- Document design element collaborations in “interface realizations”
  - One or more interaction diagrams per interface operation
  - Class diagram(s) containing the required design element relationships
- Revisit “Identify Design Elements”
  - Adjust subsystem boundaries and dependencies, as needed
Modeling Convention: Subsystem Interaction Diagrams

Subsystem Client

Subsystem Proxy

Design Element 1

Design Element 2

performResponsibility() → Op1()

Op2()

Op3()

Op4() → Subsystem interface not shown

Subsystem responsibility

Internal subsystem interactions

Subsystem interface not shown
Example: CourseCatalogSystem Subsystem in Context

1. Registrar

: RegisterForCourses
Form

2. Registration
Controller

: ICourseCatalog
System

3. Schedule

: Student

A blank schedule displayed for the students to select offerings

Student wishes to create a new schedule

A list of the available course offerings for this semester...

1://create schedule( )
2://get course offerings( )
4://display course offerings( )
5://display blank schedule( )
6://select 4 primary and 2 alternate offerings( )
7://create schedule with offerings( )
8://create with offerings( )
9://add schedule(Schedule)

Legacy RDBMS Database Access

Subsystem responsibility

Subsystem interface
Incorporating the Architectural Mechanisms: Persistency

• Analysis-Class-to-Architectural-Mechanism Map from Use-Case Analysis

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

OODBMS Persistency

RDBMS Persistency

OODBMS Persistency was discussed in Use-Case Design
Review: Incorporating JDBC: Steps

- Provide access to the class libraries needed to implement JDBC
  - Provide java.sql package

- Create the necessary DBClasses
  - One DBClass per persistent class
  - Course Offering persistent class => DBCourseOffering

√ - Done
Incorporate DBClasses into the design

- Allocate to package/layer
  - \textit{DBCourseOffering placed in CourseCatalogSystem subsystem}

- Add relationships from persistency clients
  - \textit{Persistency clients are the CourseCatalogSystem subsystem clients}

Create/Update interaction diagrams that describe:

- Database initialization
- Persistent class access: Create, Read, Update, Delete
Example: Local CourseCatalogSystem Subsystem Interaction

1. getCourseOfferings(Semester)
   1.1. read(string)
      1.1.1. createStatement()
      1.1.2. executeQuery(String)
         sql statement is passed in specifying the search criteria course offerings in the current semester
      1.1.2.1. // executeQuery()
         Retrieve all available course offerings for the current semester
      1.1.2.1. // executeQuery()
         Create a list to hold all retrieved course offerings
      1.1.3. new()
      1.1.4. new()
   1.1. read(string)
   2. getString()
   3. setData()
   4. add(CourseOffering)

Repeat these operations for each element returned from the executeQuery() command.

The CourseOfferingList is loaded with the data retrieved from the database.
The getData and setData operations are called for each attribute in the each retrieved class instance.
**Example: Billing System Subsystem In Context**

**Subsystem Interface**
- Registrar
- CloseRegistration Form
- CloseRegistration Controller
- ICourseCatalog System
- Course Offering
- Schedule
- Student
- Ibilling System

**Subsystem Responsibility**
- Close registration for each course offering
- If the maximum number of selected primary courses have not been committed, select alternate course offerings.
- Currently assuming tuition based on number of offerings taken and certain attributes of students. If different offerings get different prices this will change slightly.

**Current Flow:**
1. // close registration()
   1.1. // is registration open?()
2. // close registration()
   2.1. getCourseOfferings(Semester)
   2.2. // close registration()
   2.3. // level()
   2.4. // close()
   2.5. getTuition()
   2.6. submitBill(Student, double)

**Notes:**
- Repeat twice this is for simplicity; realistically, an indefinite number of iterations could occur.
- Finally commit or cancel the course offering once all leveling has occurred.
- Send student and tuition to the Billing System, which will do the actual billing to the student for the schedule.
Example: Local BillingSystem Subsystem Interaction

**Subsystem Proxy**

1. submitBill(Student, double)
   - 1.1. create(Student, double)
     - 1.1.1. // get contact info()
   - 1.2. submit(StudentBillingTransaction)
     - 1.2.1. // open connection()
     - 1.2.2. // process transaction()
     - 1.2.3. // close connection()
Subsystem Design Steps

- Distribute subsystem behavior to subsystem elements
- Document subsystem elements
- Describe subsystem dependencies
- Checkpoints
Example: CourseCatalogSystem Subsystem Elements

**Subsystem Interface**

- **ICourseCatalogSystem**
  - getCourseOfferings(forSemester : Semester) : CourseOfferingList

**Entity**

- **CourseOffering**
  - new()
  - add()

**Subsystem Proxy**

- **CourseCatalogSystem**
  - getCourseOfferings(forSemester : Semester) : CourseOfferingList

**Java Class Diagram**

- **CourseOfferingList**
  - new()
  - add()

- **CourseOffering**
  - new()
  - setData()

- **Connection**
  - createStatement()

- **Statement**
  - executeQuery()
  - executeUpdate()

- **ResultSet**
  - getString()
Example: Billing System Subsystem Elements

**Subsystem Interface**

```
<<Interface>>
IBillingSystem
(from External System Interfaces)
```

- `submitBill()`

**Subsystem Proxy**

```
<<subsystem proxy>>
BillingSystem
```

- `submitBill(forStudent : Student, forTuition : double)`

**Entity**

```
<<Entity>>
Student
(from University Artifacts)
```

- `// get contact info()`

**Interface**

```
BillingSystemInterface
```

- `submit(theTransaction : StudentBillingTransaction)`

**Transaction**

```
StudentBillingTransaction
```

- `create(forStudent : Student, forAmount : double)`
Subsystem Design Steps

- Distribute subsystem behavior to subsystem elements
- Document subsystem elements
- ★ Describe subsystem dependencies
- Checkpoints
Subsystem Dependencies: Guidelines

- Subsystem dependency on a subsystem

  ![Diagram showing dependency between Client Support and Server Support subsystems]

  - Flexible, Preferred

- Subsystem dependency on a package

  ![Diagram showing dependency between Client Support subsystem and Supporting Types]

  - Use with care
Example: CourseCatalogSystem Subsystem Dependencies

<<subsystem>>
CourseCatalogSystem
(from Business Services)

External System Interfaces
(from Business Services)

University Artifacts
(from Business Services)

java.sql
(from Middleware)
Example: BillingSystem Subsystem Dependencies

<<subsystem>>
BillingSystem
(from Business Services)

External System Interfaces
(from Business Services)

University Artifacts
(from Business Services)
Subsystem Design Steps

- Distribute subsystem behavior to subsystem elements
- Document subsystem elements
- Describe subsystem dependencies

★ Checkpoints
Checkpoints: Design Subsystems

- Is a realization association defined for each interface offered by the subsystem?
- Is a dependency association defined for each interface used by the subsystem?
- Are you sure that none of the elements within the subsystem have public visibility?
- Is each operation on an interface realized by the subsystem documented in an interaction diagram? If not, is the operation realized by a single class, so that it is easy to see that there is a simple 1:1 mapping between the class operation and the interface operation?
Review: Subsystem Design

- What is the purpose of Subsystem Design?
- How many interaction diagrams should be produced during Subsystem Design?
- Why should dependencies on a subsystem be on the subsystem interface?
Exercise: Subsystem Design

Given the following:

- The defined subsystems, their interfaces and their relationships with other design elements (the subsystem context diagrams)
- Patterns of use for the architectural mechanisms

(continued)
Exercise: Subsystem Design (cont.)

- Identify the following for a particular subsystem(s):
  - The design elements contained within the subsystem and their relationships
  - The applicable architectural mechanisms
  - The interactions needed to implement the subsystem interface operations

(continued)
Exercise: Subsystem Design (cont.)

- Produce the following diagrams for a particular subsystem(s):
  - “Interface realizations”
    - Interaction diagram for each interface operation
    - Class diagram containing the subsystem design elements that realize the interface responsibilities and their relationships
  - Class diagram that shows the subsystem and any dependencies on external package(s) and/or subsystem(s) (subsystem dependencies class diagram)
Exercise: Review

- Compare your Subsystem Interface Realizations
  - Have all the main and/or subflows for the interface operations been handled?
  - Has all behavior been distributed among the participating design elements?
  - Has behavior been distributed to the right design elements?
  - Are there any messages coming from the interfaces?