Chapter 11

Class and Interaction Design

© Pearson Education Limited 2005

Topics

- Finding classes from use case requirements
- Architectural elaboration of class design
- Class instantiation
- Interactions
  - Sequence diagrams
  - Communication (collaboration) diagrams
- Examples of interactions for EM
About class and interaction design

- **Class design** is the process of ensuring that the classes deliver the behavior specified in the use case model while conforming to the architectural framework chosen for the system.
  - is inseparable from **interaction design**
  - is concerned with the **design classes**
  - includes the design of **interfaces**

- **Interaction design**
  - serves the purpose of verifying the existing class design and augmenting it with further details
  - **signatures** (the argument list) of class operations (methods) can be specified
  - uses
    - **sequence diagrams** and
    - **communication diagrams** (known as **collaboration diagrams** prior to the UML 2.0)

Finding classes from use case requirements

<table>
<thead>
<tr>
<th>Req. No.</th>
<th>Requirement Definition</th>
<th>Responsible Package and Class Name</th>
<th>Responsible Operation Name</th>
<th>Collaborating Package and Class or Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>The system displays an informational message and requests that the Employee provides a username and password.</td>
<td>presentation PConsole</td>
<td>displayLogin</td>
<td>presentation PConsole</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>getUserInput</td>
<td>control CActioner</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>control CActioner</td>
<td>login</td>
</tr>
<tr>
<td>R2</td>
<td>The system attempts to connect the Employee to the EM database.</td>
<td>foundation FConnection</td>
<td>getConnection</td>
<td>acquaintance IAConstants, foundation FReader</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>foundation FReader</td>
</tr>
</tbody>
</table>
Presenting classes and dependencies

Class Diagram:

- **PConsole**
  - displayLogin()
  - getUserInput()

- **IConstants**

- **DATABASE_CONNECTION**

- **CActioner**
  - login()

- **FConnection**
  - getConnection()

- **FReader**
  - readEmployee()

Elaborating classes from architectural requirements

Architectural Elaboration of Class Design

<table>
<thead>
<tr>
<th>Req. No.</th>
<th>Responsible Class and Operation</th>
<th>Collaborating Package and Class or Interface</th>
<th>Architectural Principle and/or Pattern and/or Other Reason for Change</th>
<th>New/Updated Responsible Class and Operation</th>
<th>New/Updated Collaborating Package and Class or Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>PConsole displayLogin</td>
<td>presentation PConsole</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PConsole getUserInput</td>
<td>control CActioner</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CActioner login</td>
<td>mediator MBroker</td>
<td>DDP and Chain of Responsibility</td>
<td>MBroker login</td>
<td>foundation FConnection</td>
</tr>
<tr>
<td>R2</td>
<td>FConnection getConnection</td>
<td>acquaintance IConstants, foundation FReader</td>
<td></td>
<td>employee retrieved from the database</td>
<td>MBroker createEmployee, entity EEmployee</td>
</tr>
</tbody>
</table>
**Elaborated classes and dependencies**

- **PConsole**
  - `displayLogin()`
  - `getUserInput()`

- **IAConstants**
  - `DATABASE_CONNECTION`

- **CActioner**
  - `login()`

- **FConnection**
  - `getConnection()`

- **FReader**
  - `readEmployee()`

- **MBroker**
  - `login()`
  - `createEmployee()`

- **EEmployee**

---

**Class instantiation**

- **Who should send a message to the constructor of another class so that a new object of that class is instantiated?**
  - Some classes may be instantiated on the program’s startup by an initialization method
  - Other classes are created dynamically at runtime but their creators are known statically (at compile time)
  - In more complex cases, the exact creator of an object is determined at runtime

- **PMain**
  - `main(args : String[]) : void`
  - `<<instantiate>>`

- **PConsole**
**Instantiation diagram for EM**

- PMain
- PConsole
- EEmployee
- CActioner
- EContact
- MBroker
- EOutMessage
- FConnection
- FReader
- FWriter

---

**Interactions**

- **Interaction** – “a unit of behavior that focuses on the observable exchange of information between parts”
- “A part represents a set of instances that are owned by a containing classifier instance”
- The existence of an object (a part) at a particular time is called the **lifeline**
- An interaction is realized as a sequence of **messages** between lifelines
  - synchronous or
  - asynchronous
**Sequence diagram - notation**

- **InteractionName**
  - Client
  - Supplier

- **Lifeline representing a part.**
  - Part notation is: partName:ClassName[multiplicity]

- **Method activations with hierarchical numbering**

- **Method activations**
  - Client
    - method1(arg1 : String)
    - 1. method2( )
  - Supplier
    - 1. method1(arg1 : String)
    - 1.1. method2( )

- **Messages in sequence diagram**

  - **Class1**
    - 1. m1()
  - **Class2**
    - 1. m2()
    - 2. m3()
    - 2.1. m4()
    - 2.1.1. m5()
  - **Class3**
    - asynchronous message
    - synchronous message to self

- **Message types**
  - Synchronous message
  - Asynchronous message
  - Callback message
Messages in communication (collaboration) diagram

1. m1() → : Class1
2. m2() → : Class2
2.1. m3() → : Class3
2.1.1. m4()
2.1.1.1. m5()
2.1. m3()

EM – “Login” interaction

1. getLoginResult prompts String[]
1.1. loginUsername String, password String
1.1.1. loginUsername String, password String
1.1.1.1. connectUsername String, password String
1.1.1.1.1. query sql String
1.1.1.1.3. mapEmployees java.sql.ResultSet
1.1.1.3.1. Employee java.sql.ResultSet
1.1.1.4. closeResultSet java.sql.ResultSet

: Employee
EM – “Exit” interaction

1. getUserInput(prompts : String[])
   1.1. exit()
   1.1. logout()
   1.1. 1.1. logout()
   1.1. 1.1. close()

Summary

- **Class design** and interaction design are two sides of the same coin.
- Finding classes from **use case requirements** involves extracting requirements from the use case document and conceiving of classes and collaborations between classes.
- **Architectural constraints** introduce a need for the elaboration of the initial class design.
- **Interactions** are modeled in sequence diagrams and communication (collaboration) diagrams.
- Interactions focus on sequences of messages, not on the data that the messages pass around.