Internet and Intranet Protocols and Applications

Lecture 5a:
HTTP Client-Server Design and Implementation

February 15, 2005
Arthur Goldberg
Computer Science Department
New York University
artg@cs.nyu.edu

Topics

- OO Design
- Reading the RFC
- Testing

Key Steps In OO Design
(Given a specification and an architecture)

- Within a module identify real or abstract entities
- Map entities to objects (i.e., classes)
- Determine
  - Object state and operations
    - i.e., instance variables and methods
  - Object relationships
    - i.e., inheritance and composition
  - Tests

OO Design Axioms

1. The independence axiom. Maintain the independence of components.

2. The information axiom. Minimize the complexity (information content) of the design.

  - Design rules (antecedent axioms):
    - Design highly cohesive objects that require low coupling (1, 2).
    - Each class should have a single purpose (1, 2).
    - A large number of simpler classes enhances reusability (1, 2).
    - Map strongly from objects in the analysis to objects in the design (1).
    - Promote standardization by reusing classes and building to standard interfaces (2).
    - Design for inheritance—move common behavior to superclasses (2).
Some Approaches for Identifying Classes

- Noun phrase
- Classes, Responsibilities and Collaborators
- Unified / Use-case driven
- Common Class Patterns

Noun Phrase Technique

1. Enumerate Noun phrases
2. Eliminate irrelevant
3. Eliminate redundant
4. Identify attributes
5. Describe each class’s purpose

See my example

Classes, Responsibilities and Collaborators (CRC)

- An object’s collaborators are objects that cooperate with the object
- Describe each class on an index card like this

<table>
<thead>
<tr>
<th>Class name</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Using classes</td>
</tr>
<tr>
<td>Responsibilities</td>
<td>Uses classes</td>
</tr>
<tr>
<td>• Constructor(s)</td>
<td>Superclass</td>
</tr>
<tr>
<td>• Method(s)</td>
<td>Interface(s)</td>
</tr>
<tr>
<td>• Instance variable(s)</td>
<td>Subclasses</td>
</tr>
</tbody>
</table>

- Iterate!
  - Keep collaborators physically nearby.
  - Combine similar classes in hierarchies

Let’s do this for the HTTP Client/Server system

Common Class Patterns

- Concept class; e.g., (performance)
- Events class;
- Organization class
- People class
- Places class
- Things and devices class
Unified approach

- Booch, Rumbaugh and Jacobson
- Use-case driven development
- OO analysis = specification
- **OO design**
  - Identify classes through sequence and collaboration modeling
  - Develop sequence diagrams
- Incremental development and prototyping
- Continuous testing

---

OO Design References

- Beck, Cunningham, *A Laboratory For Teaching OO Thinking*, OOPSLA, 1989 (CRC technique)

---

Reading the RFC

- ABNF
- **Issues**
  - 4.4 Message Length
  - Persistent connections and pipelining
    - Does pipelining make sense for a single-threaded client?
- Footnotes
  - Discuss 501 (Not Implemented)

---

Testing

- Hand test
- Unit tests
  - Could use networking ...
- Use real systems
  - Client tests server
  - Server tests client
  - Monitor messages with Ethereal
- Partner testing
  - How to test
  - What to report
- To be continued ...

---
Internet and Intranet Protocols and Applications

Lecture 5a:
HTTP Client-Server
Design and Implementation

February 15, 2005
Arthur Goldberg
Computer Science Department
New York University
artg@cs.nyu.edu
Topics

- OO Design
- Reading the RFC
- Testing
Key Steps In OO Design
(Given a specification and an architecture)

- Within a module identify real or abstract entities
- Map entities to objects (i.e., classes)
- Determine
  - Object state and operations
    - I.e., instance variables and methods
  - Object relationships
    - I.e., inheritance and composition
  - Tests
1. *The independence axiom*. Maintain the independence of components.

2. *The information axiom*. Minimize the complexity (information content) of the design.

- **Design rules (antecedent axioms):**
  - Design highly cohesive objects that require low coupling (1, 2).
  - Each class should have a single purpose (1, 2).
  - A large number of simpler classes enhances reusability (1, 2).
  - Map strongly from objects in the analysis to objects in the design (1).
  - Promote standardization by reusing classes and building to standard interfaces (2).
  - Design for inheritance—move common behavior to superclasses (2).
Some Approaches for Identifying Classes

- Noun phrase
- Classes, Responsibilities and Collaborators
- Unified / Use-case driven
- Common Class Patterns
Noun Phrase Technique

1. Enumerate Noun phrases
2. Eliminate irrelevant
3. Eliminate redundant
4. Identify attributes
5. Describe each class’s purpose

See my example
Classes, Responsibilities and Collaborators (CRC)

- An object’s collaborators are objects that cooperate with the object
- Describe each class on an index card like this

<table>
<thead>
<tr>
<th>Class name</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Using classes</td>
</tr>
<tr>
<td>Responsibilities</td>
<td>Uses classes</td>
</tr>
<tr>
<td>Constructor(s)</td>
<td>Superclass</td>
</tr>
<tr>
<td>Method(s)</td>
<td>Interface(s)</td>
</tr>
<tr>
<td>Instance variable(s)</td>
<td>Subclasses</td>
</tr>
</tbody>
</table>

- Iterate!
  - Keep collaborators physically nearby.
  - Combine similar classes in hierarchies

*Let's do this for the HTTP Client/Server system*
Common Class Patterns

- Concept class; e.g., (performance)
- Events class;
- Organization class
- People class
- Places class
- Things and devices class
Unified approach

- Booch, Rumbaugh and Jacobson
- Use-case driven development
- OO analysis = specification

- OO design
  - Identify classes through sequence and collaboration modeling
  - Develop sequence diagrams
- Incremental development and prototyping
- Continuous testing
OO Design References

  http://www.rational.com/uml/html/notation (9/97)
• Coad and Yourdon. *Object-Oriented Analysis*, 1991.
• Beck, Cunningham, *A Laboratory For Teaching OO Thinking*, OOPSLA, 1989 (CRC technique)
Reading the RFC

• ABNF

• Issues
  – 4.4 Message Length
  – Persistent connections and pipelining
    • Does pipelining make sense for a single-threaded client?
  – Footnotes
    • Discuss 501 (Not Implemented)
Testing

- Hand test
- Unit tests
  - Could use networking ...
- Use real systems
  - Client tests server
  - Server tests client
  - Monitor messages with Ethereal
- Partner testing
  - How to test
  - What to report
- To be continued ...