Activity Diagram Example

Object-Oriented Analysis and Design

CSCI 6448 - Fall 1998

Kenneth M. Anderson

Goals of this Lecture

- Present an example activity diagram
  - Relate to requirements, use cases, and class diagrams
- Also, respond to a question posed last week
  - A survey of OOA&D Methods

Survey of OOA&D Methods

- The Booch Method
- The Coad and Yourdon Method
- The Jacobson Method
- The Rambaugh Method
- The Wirfs-Brock Method
- Generalization
  - Taken from "SE: A Practitioner's approach, 4th ed." by Roger S. Pressman, McGraw-Hill, 1997

Detailed comparisons

- What follows is a barebones description of each method, detailed comparisons can be found in:
  - Graham, I. Object-Oriented Methods, Addison-Wesley, 1994
  - "A comparison of Object-Oriented Development Methodologies" by Edward Berard
    - (See http://www.toa.com/shnn?htmldocs)

The Booch Method

- Identify classes and objects
  - Propose candidate objects
  - Conduct behavior analysis
  - Identify relevant scenarios
  - Define attributes and operations for each class
- Identify the semantics of classes and objects
  - Select scenarios and analyze
  - Assign responsibility to achieve desired behavior
  - Partition responsibilities to balance behavior
  - Select an object and enumerate its roles and responsibilities
  - Define operations to satisfy the responsibilities

Booch, continued

- Identify relationships among classes and objects
  - Define dependencies that exist between objects
  - Describe the role of each participating object
  - Validate by walking through scenarios
- Conduct a series of refinements
  - Produce appropriate diagrams for the work conducted above
  - Define class hierarchies as appropriate
  - Perform clustering based on class commonality
- Implement classes and objects
  - In analysis and design, this means specify everything!

Coad and Yourdon Method

- Often viewed as the easiest method to learn
- Steps
  - Identify objects using “what to look for” criteria
  - Define a generalization-specification structure
  - Define a whole-part structure
  - Identify subjects (subsystem components)
  - Define attributes
  - Define services
- Coad, P. and E. Yourdon, Object-Oriented Analysis, 2nd ed., Prentice-Hall, 1991

The Jacobson Method

- Object-Oriented Software Engineering
  - Primarily distinguished by the use-case
  - Simplified model of Objectory
  - So, Objectory is Jacobson’s current method
  - For more information on this Objectory precursor, see
    - Jacobson, I., Object-Oriented Software Engineering, Addison-Wesley, 1992.

Jacobson, continued

- Identify the users of the system and their overall responsibilities
- Build a requirements model
  - Define the actors and their responsibilities
  - Identify use cases for each actor
  - Prepare initial view of system objects and relationships
  - Review model using use cases as scenarios to determine validity
- Continued on next slide

Jacobson, continued

- Build analysis model
  - Identify interface objects using actor-interaction information
  - Create structural views of interface objects
  - Represent object behavior
  - Isolate subsystems and models for each
  - Review the model using use cases as scenarios to determine validity

The Rambaugh Method

- Object Modeling Technique (OMT)
  - Rambaugh, J. et al., Object-Oriented Modeling and Design, Prentice-Hall, 1991
- Analysis activity creates three models
  - Object model
    - Objects, classes, hierarchies, and relationships
  - Dynamic model
    - object and system behavior
  - Functional model
    - High-level Data-Flow Diagram

Rambaugh, continued

- Develop a statement of scope for the problem
- Build an object model
  - Identify classes that are relevant for the problem
  - Define attributes and associations
  - Define object links
  - Organize object classes using inheritance
- Develop a dynamic model
  - Prepare scenarios
  - Define events and develop an event trace for each scenario
  - Construct an event flow diagram and a state diagram
  - Review behavior for consistency and completeness

Rambaugh, continued

- Construct a functional model for the system
  - Identify inputs and outputs
  - Use data flow diagrams to represent flow transformations
  - Develop a processing specification for each process in the DFD
  - Specify constraints and optimization criteria
- Iterate!

The Wirfs-Brock Method

- Wirfs-Brock, R., B. Wilkerson, and L. Weiner, Designing Object-0riented Software, Prentice-Hall, 1990
  - Evaluate the customer specification
  - Use a grammatical parse to extract candidate classes
  - Group classes in an attempt to identify superclasses
  - Define and assign responsibilities for each class
  - Identify relationships between classes
  - Define collaboration between classes
  - Build hierarchical representations of classes
  - Construct a collaboration graph for the system

In general...

- Obtain customer requirements for the OO System
  - Identify scenarios or use cases
  - Build a requirements model
- Select classes and objects using basic requirements
- Identify attributes and operations for each object
- Define structures and hierarchies that organize classes
- Build an object-relationship model
- Build an object-behavior model
- Review the OO analysis model against use cases

Now to the example

- Chimera will be our example domain
- Start with some requirements
- Identify use cases
- Construct a class diagram
- Construct an activity diagram

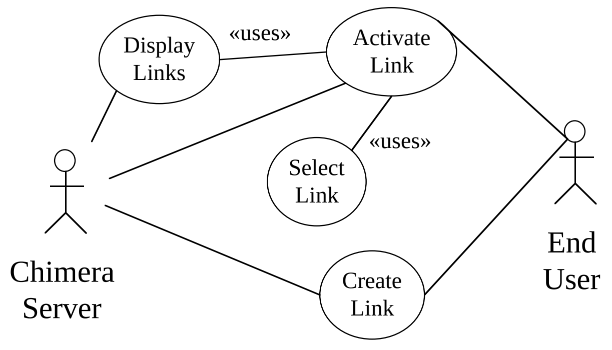
Requirements

- The Chimera Server will allow users to create links and activate them
- A Chimera viewer will allow a user to create anchors, add them to the active link, and traverse links associated with them

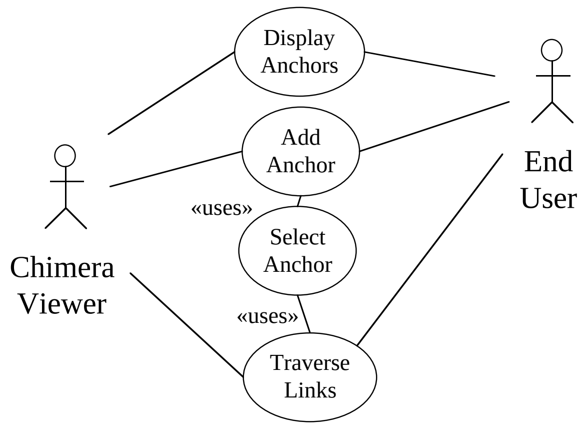
Actors

- Chimera Server
- End-User
- Chimera Viewer

Chimera Server-End User



Chimera Viewer-End User



Example Use Cases

- Activate Link

Preconditions:
    Hypermedia Context active,
    No link active
Postconditions:
    Active Link exists
Primary Actor: End-User
Secondary Actor: Chimera Server
Steps
    Display Links
    Select Link

- Select Link

Preconditions:
    Links displayed
    No selected link
Postconditions
    Link selected
Primary Actor: End-User
Steps
    Scroll through list of link names
    Click on link name

Use Cases continued

- Select Anchor

Preconditions:
    Anchors displayed
Postconditions
    Anchor selected
Primary Actor: End-User
Secondary Actor: Viewer
Steps
    Scroll through view
    Click on anchor

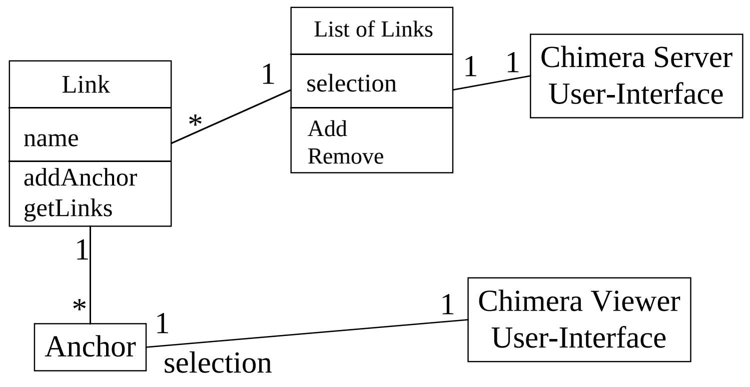
- Add Anchor

Preconditions:
    Active Link exists
    Selected Anchor exists
Postconditions
    Anchor added to active link
Primary Actor: Chimera Server
Secondary Actor: End-User
Steps
    Invoke Add Operation (GUI)
    Perform Add Operation (CS)

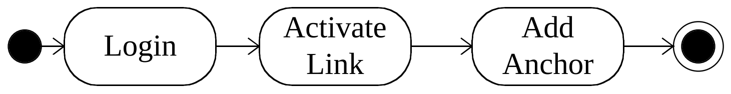
Domain Elements

- Anchor
- Link
- Active Link
- Selected Anchor
- User-Interface
- List of Links

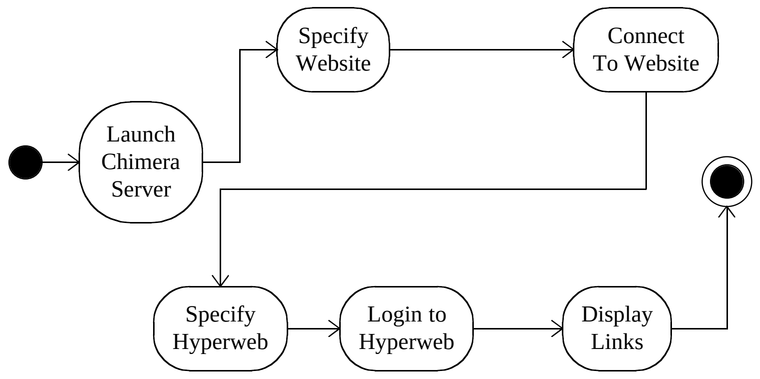
(One Possible) Class Diagram



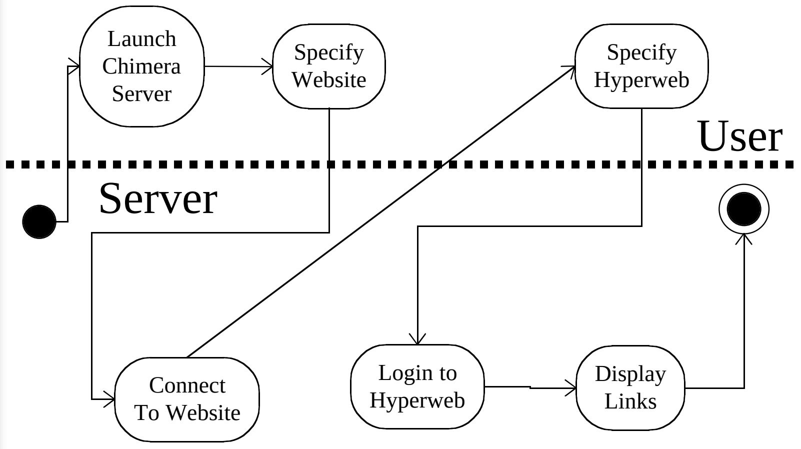
Initial Activity Diagram



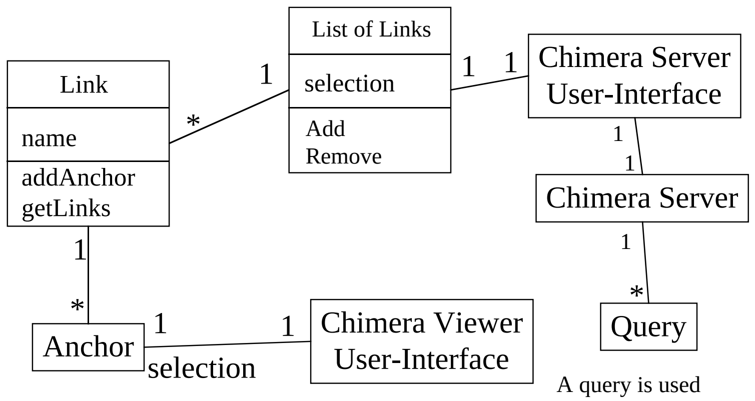
Activity Diagram Continued



Assign Responsibilities



An extension to the class diagram



A query is used to get the links that are displayed by the user-interface

Comments on Swimlanes

- Swimlanes are vertical or horizontal lines that partition the activity diagram
- Swimlanes associate activities with actors (and therefore use cases)
- Class diagrams can then be associated with activities and thus be related to use cases and actors as well

Comments on Example

- A fully decomposed activity diagram will result in complete coverage of the initially defined use cases
- In turn, this addresses the initial requirements
- Activity diagrams will bring up issues that previously could be ignored (such as logging into a hyperweb)