UML: Unified Modeling Language

V22.0474-001 Software Engineering
Lecture 5

Adapted from CS 169, George Nevela, Berkeley

Modeling

- Describing a system at a high level of abstraction
  - A model of the system
  - Used for requirements and specification
- Many notations over time
  - State machines
  - Entity-relationship diagrams
  - Dataflow diagrams

Recent History: 1980's

- The rise of object-oriented programming
- New class of OO modeling languages
- By early '90's, over 50 OO modeling languages

Recent History: 1990's

- Three leading OO notations decide to combine
  - Grady Booch (BOOCH)
  - Jim Rumbaugh (OML: Object Modeling Technique)
  - Ivar Jacobson (OOSE: OO Soft. Eng)
- Why?
  - Natural evolution towards each other
  - Effort to set an industry standard
UML

- UML stands for Unified Modeling Language
- Design by committee
  - Many interest groups participating
  - Everyone wants their favorite approach to be "in"

UML (Cont.)

- Resulting design is huge
  - Many features
  - Many loosely unrelated styles under one roof
- Could also be called Union of all Modeling Languages

This Lecture

- We discuss
  - Use Case Diagrams for functional models
  - Class Diagrams for structural models
  - Sequence Diagrams for dynamic models
  - State Diagrams
- This is a subset of UML
  - But probably the most used subset

Sources and more information

- Practical UML: A Hands-On Introduction for Developers by Randy Miller
  - http://booharland.com/article/0141.0.31863.00.html
- UML 2 for Dummies - by Chonoles and Schardt
  - Available on books24x7 through home.nyu.edu
- CS 169 web page (George Necula, Berkeley)
  - http://www-inst.eecs.berkeley.edu/~cs169/lectures.shtml
- Free UML tool
  - ArgoUML: http://argouml.tigris.org
Running Example: Automatic Train

- Consider an unmanned people-mover
  - as in many airports

- Train
  - Moves on a circular track
  - Visits each of two stations (A and B) in turn
  - Each station has a "request" button
    - To stop at this station
  - Each train has two "request" buttons
    - To stop at a particular station

Use-Cases

- Describe functionality from the user’s perspective
- One (or more) use-cases per kind of user
  - May be many kinds in a complex system
- Use-cases capture requirements

An Example Use-Case in UML

- Name
  - Normal Train Ride
- Actors
  - Passenger
- Entry Condition
  - Passenger at station
- Exit Condition
  - Passenger leaves station
An Example Use-Case in UML

- Event-flow
  - Passenger presses request button
  - Train arrives and stops at platform
  - Doors open
  - Passenger steps into train
  - Doors close
  - Passenger presses request button for final stop
  - ... 
  - Doors open at final stop
  - Passenger exits train
- Nonfunctional requirements

Exceptional Situations

- Use cases have relationships
  - Inclusion (E.g., push button included in ride)
  - Variations
- UML has a special notation
  - The "extends" relationship to express a exceptional variation of a use case
  - Normally used to express errors

Use Case Diagram

- Graph showing
  - Actors
  - Use cases
  - Edges actor-case if that actor is involved in that case
- Actors
  - Stick figures
- Use cases
  - Ovals

Extension

Dotted arrow pointing to "normal" case
Summary of Use Cases

- Use Case Diagram
  - Shows all actors, use cases, relationships
  - Actors are agents external to the system
    - E.g., users

- Supplemental information
  - Entry/Exit Conditions, Story, Main and Alternative flows, Nonfunctional requirements
  - Specified in a separate document
    - In English

Class Diagrams

- Describe classes
  - In the OO sense

- Each box is a class
  - List fields
  - List methods

- The more detail, the more like a design it becomes

Class Diagrams: Relationships

- Many different kinds of edges to show different relationships between classes

- Mention just a couple

 Associations

- Capture n-m relationships
  - Subsumes ER diagrams

- Label endpoints of edge with cardinalities
  - Use * for arbitrary

- Typically realized with embedded references
  - Can be directional (use arrows in that case)

  One request button per station; each train has two request buttons
Aggregation

- Show contains a relationships
- Station and Train classes can contain their respective buttons
- Denoted by open diamond on the "contains" side

More about Class Diagrams

- Classes vs Objects
  - Some diagrams can be used to specify relationships between instances of classes
- Roles and Association Classes
  - More detail on relationships between classes
- Hierarchical Diagrams

Generalization

- Inheritance between classes
- Denoted by open triangle

Sequence Diagrams

- A table
  - Columns are classes or actors
  - Rows are time steps
  - Entries show control/data flow
    - Method invocations
    - Important changes in state
Example Sequence Diagram

Methods and Actors

Method invocation
Note: These are all synchronous method calls. There are other kinds of invocations.

Invocation lifetime spans lifetimes of all nested invocations

"Lifelines" fill in time between invocations
**Sequence Diagrams Notes**

- Sequence diagrams
  - Refine use cases
  - Gives view of dynamic behavior of classes
    - Class diagrams give the static class structure

- Not orthogonal to other diagrams
  - Overlapping functionality
  - True of all UML diagrams

**Activity Diagrams**

- Reincarnation of flow charts
  - Uses flowchart symbols

- Emphasis on control-flow

- Two useful flowchart extensions
  - Hierarchy
    - A node may be an activity diagram
  - Swim lanes

**Example Activity Diagram**

- Activities in rounded rectangles
- May itself be a nested activity diagram

**Example Activity Diagram**

- Concurrency, fork & join

- lightButton
- addStop
**Example Activity Diagram**

Swim lanes show which classes/actors are responsible for which part of the diagram.

- Station
- Train
- pushButton
- lightButton
- addStop

---

**Another Example Activity Diagram**

Classic flow-chart if-then-else

- StopRequested?
  - yes
  - stopTrain
  - no
  - announceNoStop

---

**StateCharts**

- Hierarchical finite automata
  - Invented by David Harel, 1983
- Specify automata with many states compactly
- Complications in meaning of transitions
  - What it means to enter/exit a compound state

---

**Example Simple StateChart**

- Button
  - on
  - push
  - off
  - depart
StateChart for the Train

- A train can be
  - At a station (atA, atB)
  - Between stations (AtoB, BtoA)

- Pending requests are subset of \{A,B\}

- 16 possible states
  - Transitions: pushA, pushB, departA, departB, ...

StateChart for Buttons + Train

- Transition causes control to leave any possible state of the component automaton

Opinions about UML: What's Good

- A common language
  - Makes it easier to share requirements, specs, designs

- Visual syntax is useful, to a point
  - A picture is worth 1000 words
  - For the non-technical, easier to grasp simple diagrams than simple pseudo-code

- To the extent UML is precise, forces clarity
  - Much better than natural language

- Commercial tool support
  - Something natural language could never have
<table>
<thead>
<tr>
<th>Opinions On UML: What's Bad</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Hodge-podge of ideas</td>
</tr>
<tr>
<td>- Union of most popular modeling languages</td>
</tr>
<tr>
<td>- Sublanguages remain largely unintegrated</td>
</tr>
<tr>
<td>- Visual syntax does not scale well</td>
</tr>
<tr>
<td>- Many details are hard to depict visually</td>
</tr>
<tr>
<td>- Ad hoc text attached to diagrams</td>
</tr>
<tr>
<td>- No visualization advantage for large diagrams</td>
</tr>
<tr>
<td>- 1000 pages are very hard to understand</td>
</tr>
<tr>
<td>- Semantics is not completely clear</td>
</tr>
<tr>
<td>- Some parts of UML underspecified, inconsistent</td>
</tr>
<tr>
<td>- Plans to fix</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UML is Happening</th>
</tr>
</thead>
<tbody>
<tr>
<td>- UML is being widely adopted</td>
</tr>
<tr>
<td>- By users</td>
</tr>
<tr>
<td>- By tool vendors</td>
</tr>
<tr>
<td>- By programmers</td>
</tr>
<tr>
<td>- A step forward</td>
</tr>
<tr>
<td>- Seems useful</td>
</tr>
<tr>
<td>- First standard for high-levels of software process</td>
</tr>
<tr>
<td>- Expect further evolution, development of UML</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tutorials and Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>- C++ Tutorials</td>
</tr>
<tr>
<td>- Start next Monday</td>
</tr>
<tr>
<td>- Two sessions: 1:00-2:00 and 3:15-4:15</td>
</tr>
<tr>
<td>- Location to be announced</td>
</tr>
<tr>
<td>- Make sure you’re signed up for the email list!</td>
</tr>
<tr>
<td>- Project</td>
</tr>
<tr>
<td>- Selected project proposals are online at</td>
</tr>
<tr>
<td>- <a href="http://c.cnyu.edu/courses/spring05/CS/2474201/pr-requests.txt">http://c.cnyu.edu/courses/spring05/CS/2474201/pr-requests.txt</a></td>
</tr>
<tr>
<td>- Bidding form will be posted to web page shortly</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>XP Lecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Thursday, Feb. 10, 3:30pm</td>
</tr>
<tr>
<td>- Pace University, E319</td>
</tr>
<tr>
<td>- Email me if interested</td>
</tr>
</tbody>
</table>