UML: Unified Modeling Language

V22.0474-001 Software Engineering
Lecture 5, Spring 2008

Clark Barrett, New York University
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
History: 1980’s

• The rise of object-oriented programming

• New class of OO modeling languages

• By early ’90’s, over 50 OO modeling languages
History: 1990's

- Three leading OO notations decide to combine
  - Grady Booch (BOOCH)
  - Jim Rumbaugh (OML: Object Modeling Technique)
  - Ivar Jacobsen (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
  - Activity 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://dn.codegear.com/article/31863

- UML 2 for Dummies - by Chonoles and Schardt
  - Available on books24x7 through home.nyu.edu

- 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
Picture
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**
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

Diagram:
- Actors: passenger, technician
- Use cases: Ride, Repair
- Edges: passenger to Ride, technician to Repair
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 an exceptional variation of a use case
  - Normally used to express errors
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

<table>
<thead>
<tr>
<th>Train</th>
</tr>
</thead>
<tbody>
<tr>
<td>lastStop</td>
</tr>
<tr>
<td>nextStop</td>
</tr>
<tr>
<td>velocity</td>
</tr>
<tr>
<td>doorsOpen?</td>
</tr>
<tr>
<td>addStop(stop);</td>
</tr>
<tr>
<td>startTrain(velocity);</td>
</tr>
<tr>
<td>stopTrain();</td>
</tr>
<tr>
<td>openDoors();</td>
</tr>
</tbody>
</table>
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
Generalization

- Inheritance between classes
- Denoted by open triangle

```
Button
  / \  
|   | 
| \  |
|  \|
|  RequestButton
  \-
|   |
| \  |
|  \-
|  EmergencyButton
```

22
More about Class Diagrams

- **Classes vs Objects**
  - Same diagrams can be used to specify relationships between instances of classes

- **Roles and Association Classes**
  - More detail on relationships between classes

- **Hierarchical Diagrams**
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

Passenger

PushButton()

Station

addStop()

PushButton(S)

Train

openDoors()

closeDoors()
Example Sequence Diagram

Method invocation

Note: These are all synchronous method calls. There are other kinds of invocations.
Example Sequence Diagram

Invocation lifetime spans lifetimes of all nested invocations
Example Sequence Diagram

Passenger → Station

pushButton()

Station → Train

addStop()

openDoors()

pushButton(S)

Train

closeDoors()

“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

pushButton

lightButton

addStop

Station

Train
Example Activity Diagram

Concurrency, fork & join

Station

pushButton

lightButton

Train

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

off

depart

push

on
```
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

Dotted lines separate concurrent automata

ButtonA

off

on

departA

pushA

Train

atA, A

departA

ButtonB

AtoB, none

derail
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
Opinions On UML: What’s Bad

• Hodge-podge of ideas
  - Union of most popular modeling languages
  - Sublanguages remain largely unintegrated

• Visual syntax does not scale well
  - Many details are hard to depict visually
    • Ad hoc text attached to diagrams
  - No visualization advantage for large diagrams
    • 1000 pictures are very hard to understand

• Semantics is not completely clear
  - Some parts of UML underspecified, inconsistent
  - Plans to fix
UML is Happening

• UML is being widely adopted
  - By users
  - By tool vendors
  - By programmers

• A step forward
  - Seems useful
  - First standard for high-levels of software process
  - Expect further evolution, development of UML
Suggestions on using UML

• Requirements
  - Use Case Diagrams to illustrate use cases
  - Activity or Sequence Diagrams to illustrate typical flow within a use case (scenarios)

• Design
  - Class Diagram for system architecture
Presentations (Requirements)

• 20 minutes/presentation
  - Enough time to give some details

• Format
  - 15 minute presentation
  - 5 minutes Q&A

• Try to make your presentation useful
  - It is a plus to share negative experiences, perhaps with solutions
Presentation 1: Requirements: ~10 slides

1. Project name and name of speaker
2. What does it do?
   - Brief description of what project will do
3. Who are the customers?
   - List of customers you have contacted
   - Comments on each
4. What are the requirements?
   - Bulleted list, use cases
5. What are the problems?
   - What don’t you know how to solve yet?

- HTML, PDF, or PowerPoint
- Email to barrett@cs by 10am on the day of presentations.
Presentation 2: Design: ~10 slides, 20 min.

1. Project name and name of speaker
   • Different speaker than last time
2. How has the spec. changed
   • If nothing, say “none”
3. Design
4. Plan
   • Implementation and testing plan
5. What are the problems?
   - What don’t you know how to solve yet?

• HTML, PDF, or Powerpoint
• Email to barrett@cs by 10 a.m. on the day of the presentation
Presentations 3&4: Testing, Final Report

1. Different speakers (so everyone gets a chance)
2. More information on these coming later