Unified Modeling Language

Modeling Language
- A way of expressing the various models produced during the development process
- A language whose vocabulary and rules focus on the conceptual and physical representations of a system

UML
- Developed by Booch, Rumbaugh, and Jacobson at Rational Software Corporation
  - Standard Language for writing software blueprints
  - Compendium of diverse design diagrams
- Some references:
  - James Rumbaugh, Ivar Jacobson, and Grady Booch. The Unified Modeling Language User Guide. Addison-Wesley
  - Perdita Stevens with Rob Pooley. Using UML. Addison-Wesley, 2000
  - http://www.agilemodeling.com/artifacts/
  - Wikipedia

Why use UML?
- Diagrams facilitate communication
- Widely used
- Does not have well defined semantics, so sometimes the meaning is ambiguous or inconsistent

Most Commonly used UML Diagrams
- Use Case Diagrams
- State Machine Diagrams
  - State Transition Diagrams
- Activity Diagrams
- Sequence Diagrams
- Class Diagrams
Use Case Diagrams

- Models typical interactions between a user (or users) and a system
  - Captures one or more scenarios
  - Can be small or large
  - Describes how to achieve a goal for the user of the system

Constructs

- Actor: A role that a user plays with respect to the system
- Use Cases: A scenario or a task; what a system does, not how it does it
- Communications: Interactions between user and tasks
- Includes: Factors out common behavior
  - (pre-existing and/or reusable component)
- Extends: Separates variant behavior
- Generalizations: Inheritance of behavior

A Simple Use Case Diagram

Use Case: Basic Notation

ATM Example

ATM Example
ATM Example

Benefits of Use Case Diagrams
- Models how the functions of a system are applied
- Validates that the requirements are covered
- Provides a high level description of the test cases
  - One use case may result in many test cases

State Transition Diagram

State Machine Diagram (State Transition Diagrams)
- Shows the components of the system and the flow between those components
  - But does not show the ordering of this flow
- Often used to show the high-level architecture of a system
- Can use hierarchical decomposition to further decompose any of the components

Sequence Charts
- Describe the flow between components
- Provides some information about the ordering of the flow
  - Vertical axis represents time
- Often used to describe interaction among distributed components
Sequence Diagrams (Message Sequence Charts)

Benefits of Sequence Diagrams
- Shows the flow between components
  - Useful for showing the flow among distributed components
- State Transition Diagrams do not show the order in which the flow occurs or any constraints on that flow
- Sequence Diagrams provide some ordering information
- Activity Diagrams provide ordering information and information about the constraints on that ordering

Activity Diagrams
- Shows the flow through the system
  - Similar to flowcharts and control-flow graphs
  - Lower level of detail than provided by State Transition Diagrams or Sequence Diagrams

ATM Example
- Deposit Check
- Validate Check
- Is Valid?
- Reject Deposit
- Record Transaction
- Print Receipt
- Welcome Next visitor

Construction Example
- An activity diagram showing the sequence of activities in a construction process

Constructs
- Activity: A task that needs to be done
- Transition: Shows the path from one activity to the next activity
- Decision boxes/guards: Shows decisions, asks questions, booleans
- Synchronization bars (forks/joins): Shows parallel behavior; concurrent threads
- Swimlanes: Groups the activities according to responsibilities or the actors
- Start and stop markers
Upload Files

Activity Diagram with Swim Lanes

UML Class Diagrams

- Represent static structure of a system’s classes (types)
- Show attributes (fields, parameters) of classes and relationships with other classes
- Useful for both architecture and low-level design

Class Hierarchy Example

Class Example

<table>
<thead>
<tr>
<th>name</th>
<th>attributes</th>
<th>responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>Name, Phone Number, Email Address, Student Number, Average Mark</td>
<td>Eligible to Enroll, Provide Seminars Taken</td>
</tr>
</tbody>
</table>

Class and Method Example

<table>
<thead>
<tr>
<th>Course</th>
<th>public methods (can specify private)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>getName()</td>
</tr>
<tr>
<td></td>
<td>getCourseNumber()</td>
</tr>
<tr>
<td></td>
<td>setCourseNumber(number)</td>
</tr>
<tr>
<td></td>
<td>getFee()</td>
</tr>
<tr>
<td></td>
<td>setFees(amount)</td>
</tr>
<tr>
<td></td>
<td>getName(amount)</td>
</tr>
</tbody>
</table>
Class Associations

- **Association between A and B**
- **Label should be clear**

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..1</td>
<td>Zero or one</td>
</tr>
<tr>
<td>1</td>
<td>One only</td>
</tr>
<tr>
<td>0..*</td>
<td>Zero or more</td>
</tr>
<tr>
<td>1..*</td>
<td>One or more</td>
</tr>
<tr>
<td>0..n</td>
<td>Zero to n (where n (\neq 1))</td>
</tr>
<tr>
<td>1..n</td>
<td>One to n (where n (\geq 1))</td>
</tr>
</tbody>
</table>

Class and Relationship Example

- **Student**
  - Name
  - Phone Number
  - Email Address
  - Student Number
  - Average Mark
  - Is Eligible to Enroll
  - Enroll
  - Provide Seminars Taken

- **Address**
  - Street
  - City
  - State
  - Postal Code
  - Country
  - Validate
  - Output As Label

Class Inheritance Example

- **Person**
  - Name
  - Phone Number
  - Email Address
  - Purchase Parking Pass

- **Address**
  - Street City
  - State Postal Code
  - Country
  - Validate
  - Output As Label

- **Student**
  - Student Number
  - Average Mark
  - Is Eligible To Enroll
  - Get Seminars Taken

- **Professor**
  - Salary

Possible end result

UML tools

- Together Control Center 6.0
  - [http://www.togethersoft.com/](http://www.togethersoft.com/)
- Poseidon
- VP–UML
  - [http://www.visual-paradigm.com/](http://www.visual-paradigm.com/)