The Rational Unified Process (RUP) and Unified Modeling Language (UML)

Todd Bacastow
IST 210: Organization of Data
RUP is a Visual Modeling Tool

"Modeling captures essential parts of the system."
Dr. James Rumbaugh

Visual Modeling is modeling using standard graphical notations

Copyright © 1997 by Rational Software Corporation
Capture Business Process

*Use Case Analysis is a technique to capture business process from user’s perspective*
Capture business objects and logic

Analyze and design your application
Manage Complexity

The human mind can only handle 7 plus or minus 2 things at once
Define Software Architecture

Model your system independent of implementation language
Promote Reuse

Reusable Components

Multiple Systems
What is the UML?

- UML stands for Unified Modeling Language
- The UML combines the best of the best from
  - Data Modeling concepts (Entity Relationship Diagrams)
  - Business Modeling (work flow)
  - Object Modeling
  - Component Modeling
- The UML is the standard language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system
- It can be used with all processes, throughout the development life cycle, and across different implementation technologies
History of the UML

- Oct '95: Dr. Ivar Jacobson joins Rational (Fall of 1995)
- Oct '95: Dr. James Rumbaugh joins Rational (Oct, 1994)
- UML 0.9
- Unified Method 0.8
- OMT
- Booch
- Use Case
- Jan '97
- UML 1.0
- Sept '97
- UML 1.1
- Nov ‘97

UML approved by the OMG
UML Supports Application Development

Objects

Business Objects

Relationships

large scale system

application partitioning

Scenarios

CORBA

OMG

Components

Microsoft

Use Cases

ActiveX/COM

Microsoft

Business Process

ORMDBMS

Oracle

Classes
The UML may be used to:

- Display the boundary of a system & its major functions using use cases and actors
- Illustrate use case realizations with interaction diagrams
- Represent a static structure of a system using class diagrams
- Model the behavior of objects with state transition diagrams
- Reveal the physical implementation architecture with component & deployment diagrams
- Extend your functionality with stereotypes
Putting the UML to Work

- A University wants to computerize their registration system
  - The Registrar sets up the curriculum for a semester
    - One course may have multiple course offerings
  - Students select 4 primary courses and 2 alternate courses
  - Once a student registers for a semester, the billing system is notified so the student may be billed for the semester
  - Students may use the system to add/drop courses for a period of time after registration
  - Professors use the system to receive their course offering rosters
  - Users of the registration system are assigned passwords which are used at logon validation
An actor is someone or some thing that must interact with the system under development.

- Registrar
- Professor
- Student
- Billing System
A use case is a pattern of behavior the system exhibits
- Each use case is a sequence of related transactions performed by an actor and the system in a dialogue

Actors are examined to determine their needs
- Registrar -- maintain the curriculum
- Professor -- request roster
- Student -- maintain schedule
- Billing System -- receive billing information from registration
Documenting Use Cases

- A flow of events document is created for each use case
  - Written from an actor point of view
- Details what the system must provide to the actor when the use case is executed
- Typical contents
  - How the use case starts and ends
  - Normal flow of events
  - Alternate flow of events
  - Exceptional flow of events
Maintain Curriculum Flow of Events

- This use case begins when the Registrar logs onto the Registration System and enters his/her password. The system verifies that the password is valid (E-1) and prompts the Registrar to select the current semester or a future semester (E-2). The Registrar enters the desired semester. The system prompts the professor to select the desired activity: ADD, DELETE, REVIEW, or QUIT.

- If the activity selected is ADD, the S-1: Add a Course subflow is performed.

- If the activity selected is DELETE, the S-2: Delete a Course subflow is performed.

- If the activity selected is REVIEW, the S-3: Review Curriculum subflow is performed.

- If the activity selected is QUIT, the use case ends.

- …
Use case diagrams are created to visualize the relationships between actors and use cases.
Uses and Extends Use Case Relationships

- As the use cases are documented, other use case relationships may be discovered
  - A uses relationship shows behavior that is common to one or more use cases
  - An extends relationship shows optional behavior
Use Case Realizations

- The use case diagram presents an outside view of the system
- Interaction diagrams describe how use cases are realized as interactions among societies of objects
- Two types of interaction diagrams
  - Sequence diagrams
  - Collaboration diagrams
A sequence diagram displays object interactions arranged in a time sequence.

- Student
- registration form
- registration manager
- math 101
- math 101 section 1

1: fill in info
2: submit
3: add course(joe, math 01)
4: are you open?
5: are you open?
6: add (joe)
7: add (joe)
A collaboration diagram displays object interactions organized around objects and their links to one another.
Class Diagrams

- A class diagram shows:
  - Classes
    - Attributes
    - Methods
  - Interfaces
  - Collaborations
  - Dependency, Generalization, Relationships
- A class diagram is a STATIC view of system
Basic Class Diagrams

Class Name

Class Attributes

Class Methods
Basic Class Diagrams

- **Superclass**
- **Subclass**
- **Class with parts**
- **Assembly Class**
- **Inheritance** *(Generalization)* *(is-a, kind-of)*
- **Composition** *(Part-Of)*
- **Association** *(relationship)*

**Note**
- **name**
Basic Class Diagram (Example)

- Brain
- Person
- Money
- Student
- Class

The diagram shows the relationships between the classes:
- A Person class has a relationship with a Brain class and a Money class.
- A Student class has a relationship with a Person class and a Class class.
- The Student class also has a relationship with the takes relationship to the Class class.
Class Diagrams

Cardinality (Multiplicity)
1
0..1
0..n
1..n
*

Student \( \text{takes} \) 0..n Class
Classes

- A class is a collection of objects with common structure, common behavior, common relationships and common semantics
- Classes are found by examining the objects in sequence and collaboration diagram
- A class is drawn as a rectangle with three compartments
- Classes should be named using the vocabulary of the domain
  - Naming standards should be created
  - e.g., all classes are singular nouns starting with a capital letter
Operations

- The behavior of a class is represented by its operations
- Operations may be found by examining interaction diagrams

3: add course(joe, math 01)

<table>
<thead>
<tr>
<th>registration form</th>
<th>registration manager</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>RegistrationManager</th>
</tr>
</thead>
<tbody>
<tr>
<td>addCourse(Student,Course)</td>
</tr>
</tbody>
</table>
Attributes

- The structure of a class is represented by its attributes.
- Attributes may be found by examining class definitions, the problem requirements, and by applying domain knowledge.

Each course offering has a number, location and time.
Classes

RegistrationForm

ScheduleAlgorithm

RegistrationManager

addStudent(Course, StudentInfo)

Course

name
numberCredits

open()
addStudent(StudentInfo)

Student

name
major

Professor

name
tenureStatus

CourseOffering

location

open()
addStudent(StudentInfo)
Relationships

- Relationships provide a pathway for communication between objects.
- Sequence and/or collaboration diagrams are examined to determine what links between objects need to exist to accomplish the behavior -- if two objects need to “talk” there must be a link between them.
- Three types of relationships are:
  - Association
  - Aggregation
  - Dependency
Relationships

- An association is a bi-directional connection between classes
  - An association is shown as a line connecting the related classes

- An aggregation is a stronger form of relationship where the relationship is between a whole and its parts
  - An aggregation is shown as a line connecting the related classes with a diamond next to the class representing the whole

- A dependency relationship is a weaker form of relationship showing a relationship between a client and a supplier where the client does not have semantic knowledge of the supplier
  - A dependency is shown as a dashed line pointing from the client to the supplier
Finding Relationships

- Relationships are discovered by examining interaction diagrams
  - If two objects must “talk” there must be a pathway for communication

3: add student(joe)
Relationships

- RegistrationForm
- RegistrationManager
  - addStudent(Course, StudentInfo)
- ScheduleAlgorithm
- Course
  - name
  - numberCredits
  - open()
  - addStudent(StudentInfo)
- Student
  - name
  - major
  - open()
  - addStudent(StudentInfo)
- Professor
  - name
  - tenureStatus
- CourseOffering
  - location
  - open()
  - addStudent(StudentInfo)
Multiplicity and Navigation

- Multiplicity defines how many objects participate in a relationship
  - Multiplicity is the number of instances of one class related to ONE instance of the other class
  - For each association and aggregation, there are two multiplicity decisions to make: one for each end of the relationship
- Although associations and aggregations are bi-directional by default, it is often desirable to restrict navigation to one direction
- If navigation is restricted, an arrowhead is added to indicate the direction of the navigation
Inheritance

- Inheritance is a relationships between a superclass and its subclasses
- There are two ways to find inheritance:
  - Generalization
  - Specialization
- Common attributes, operations, and/or relationships are shown at the highest applicable level in the hierarchy
Inheritance
The State of an Object

- A state transition diagram shows
  - The life history of a given class
  - The events that cause a transition from one state to another
  - The actions that result from a state change

- State transition diagrams are created for objects with significant dynamic behavior
**State Transition Diagram**

- **Initialization**
  - do: Initialize course
  - Add Student / Set count = 0

- **Open**
  - entry: Register student
  - exit: Increment count
  - Add student[ count < 10 ]

- **Closed**
  - do: Finalize course

- **Canceled**
  - do: Notify registered students
  - [ count = 10 ]
The Physical World

- Component diagrams illustrate the organizations and dependencies among software components
- A component may be
  - A source code component
  - A run time components or
  - An executable component
Component Diagram
Deploying the System

- The deployment diagram shows the configuration of run-time processing elements and the software processes living on them.
- The deployment diagram visualizes the distribution of components across the enterprise.
Deployment Diagram

- Registration
- Database
- Library
- Main Building
- Dorm
Quiz Review

- 100 points
- 7 questions
  - 1 x Data organization process essay (20%)
  - 1 x Draw an ERD (20%)
  - 1 x Normalize a table (20%)
  - 1 x Relational Algebra (10%)
  - 3 x SQL (30%)
- Almost 100% from
  - Quizzes
  - Labs
  - PowerPoints
  - In-class exercises
- Closed book, closed note --- you are on your own!
- Bring a pencil
- 75 minutes