Introduction to Unified Modelling Language (UML)
(part 1- Building Blocks & Use Case Modelling)

CS 446/646 ECE452
May 4th, 2010

Material covered in this lecture is based on various chapters from UML 2 and the Unified Process- 2nd Edition Practical Object Oriented Analysis & Design

IMPORTANT NOTICE TO STUDENTS
These slides are NOT to be used as a replacement for student notes. These slides are sometimes vague and incomplete on purpose to spark a class discussion.
Outline

Introduction

Use case Modelling
Introduction

What is UML

- visual modelling language
- UML is a language not a methodology?
  - Q: why is this distinction important?

UML Model

- repository of all things and relationships
Building Blocks

Things

• Things are the nouns of a UML model

• Three types
  – structural things:
    • *nouns* (class, interface, use case etc.)
  – behaviour things:
    • *verbs* (interactions, activities etc.)
  – grouping things:
    • *package*
  – annotational things:
    • *note*
Building Blocks

Relationships

- *representation* of how things relate to each other
- adds *semantics* to connections between entities
Building Blocks

Diagrams

- views of a UML model
- not a model itself
  - things/relationships can be omitted from diagrams but still be part of the model
Building Blocks

Diagram

Structure Diagram
- Package Diagram
- Class Diagram
- Component Diagram

Behavior Diagram
- Use case Diagram
- Interaction Diagram
- Activity Diagram
  - Communication Diagram
  - Sequence Diagram
Building Blocks

Types of UML Diagrams that

- represent static model
  - things
  - the structural relationship between things

- represent dynamic behaviour
  - how things interact to generate the required functionality
Building Blocks

UML Model

- graphical for visualization
- specifications
  - provide meaning to the visual components

<table>
<thead>
<tr>
<th>«account» BankAccount</th>
<th>Deposit</th>
<th>— — — — &gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountNumber</td>
<td></td>
<td></td>
</tr>
<tr>
<td>owner</td>
<td></td>
<td></td>
</tr>
<tr>
<td>balance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>withdraw()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>calculateInterest()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>deposit()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class Specification</th>
<th>Use case Specification</th>
<th>Dependency Specification</th>
</tr>
</thead>
</table>
Building Blocks

Extensibility Mechanisms

- stereotypes
  - allow defining new modelling elements based on existing
  - must define the semantics of the new elements
    - or else the new model is just a picture
    - usually attach a note to the new element
Use Case Modelling
Use Case Modelling

Definition

● formalism to capture system requirements

Purpose

● to capture discrete unit of interactions between
  – system components & actors
  – not between actors

● Q: is use case diagram a behavioural or structural diagram?
Use Case Modelling

Components

• system/boundary
  – defines internal (system) & external parts

• actors
  – a role that an external entity adopts when interacting with the systems

• flows
  – main & alternative flows
Closer Look at Actors

Actors are External

- actors are **always external** to a system

```
«actor»
Customer
```

```
«entity»
Customer

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>address</th>
</tr>
</thead>
</table>
```

- *your system may contain internal representation of the external actors*
Closer Look at Actors

Identifying Actors

- consider who or what \textit{uses/interacts} with your system

Use Roles not Individuals

- assuming a set of customers \{Jim, Mike, Helen\}
  - Q: who/what should be the actor here?

Other things as actors

- Q: what about time as an actor?
  - e.g. scheduled tasks
What are Use case

Definition

- “A specification of sequence of actions, including variant sequences and error sequences, that a system, subsystem or class can perform by interacting with the outside actors” - UML Reference Manual, 2nd Edition-2004

- A use case defines system behaviour during interactions with the actors
An Example

The diagram represents a mail order system with the following components:

- **Actor**: Customer
  - **Communication Relationship**: Customer
  - **Use Case**: Place Order, Cancel Order, Validate Order, Request Catalog
- **System Boundary**: Mail Order System
  - **Use Case**: Ship Product
  - **Actors**: Shipping Company, Dispatcher
Actor Generalization

Actor Generalization

Sales System
- ListProducts
- OrderProducts
- AcceptPayment
- CalculateCommission

Sales System
- ListProducts
- OrderProducts
- AcceptPayment
- CalculateCommission

SalesAgent
Customer

Purchaser
SalesAgent
Customer
Use Case Generalization

Child use case may

- inherit features from the parent use case
- add new features
- override inherited features
Use Case Specification

**ID1: use case: PaySalesTax**

**id & name**

**description**

Pay sales tax to to the tax authority

**actors**

Primary actors: Secondary actors

**pre conditions**

Pre-Conditions:

1. it is the end of business quarter

**main flow**

Main Flow:

1. determine the amount of tax owed
2. prepare payment & other related documentation
3. send an electronic payment

**post conditions**

Post-Conditions:

1. tax authority receives the amount

**alternate flows**

Alternate flows:

InvalidPaymentAmount
Cancel

one main flow

0 or more alt flows may not merge with the main flow
## Use Case Generalization-Revisited

<table>
<thead>
<tr>
<th>Use case feature</th>
<th>Inherit</th>
<th>Add</th>
<th>Override</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relationship</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Extension point</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Precondition</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Postcondition</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Step in main flow</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Alternative flow</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
Use Case «include»

Purpose

- collect common use case steps into a single use case to be reused by other relevant (base) use cases
Use Case «extend»

Purpose

• provides a mechanism to insert new behaviour into existing (base) use case
  – base use case provides extension points (hooks)
  – extension use case provides a set of insertion segments
Use Case «extend»

Observation

- base use case does not know about the extension use case
- extension use cases are not complete on their own
- base use cases are complete

The first segment in `IssueFine` is inserted at `overdueBook` and the second segment at `payFine`
Use Case «extend»

Rules

- «extend» relationship must identify one or more extension points in the base use case
  - otherwise the relationship refers to all extension points
- extension use case must have the same number of insertion segments as there are extension points in the «extend» relationship
- it is legal for two extension use cases to «extend» the same base use case at the same extension point
Conditional Extensions

- **ReturnBook**
  - extension points: overdueBook, payFine

- **IssueWarning**
  - condition: {first offense}
  - extension point: overdueBook

- **IssueFine**
  - condition: {!first offense}
  - extension point: overdueBook, payFine
Use Case Modelling

Functional Decomposition

Good or Bad?

- what is functional decomposition?

![Diagram of functional decomposition]

System

- MaintainBooks
  - AddBook
  - DeleteBook

- MaintainTickets
  - AddTicket
  - DeleteTicket

- RunLibrary
  - MaintainTickets
  - MaintainLoans

- MaintainLoans
  - AddLoan
  - DeleteLoan
Functional Decomposition

Observations

• focus on capturing the requirements
• it is not object design
• the higher level use cases might not be of interest
• model is complicated
• usually indicates that the analyst is viewing the system in a procedural way rather than the OO paradigm

Verdict

• bad
Use Case & Requirements Tracing

Purpose

- link requirements to use cases
- many-to-many relationship

Requirement Tracing Matrix

- validate consistency
  - missing use cases
  - missing requirements

<table>
<thead>
<tr>
<th>requirements</th>
<th>UC1</th>
<th>UC2</th>
<th>UC3</th>
<th>UC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>R4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>