MSc programme (induction week) – Computer Science Department

INTRODUCTION TO UML

Some of this material is based on
Overview: modelling with UML

♦ What is modelling?
♦ What is UML?
♦ Use case diagrams
♦ Class diagrams
♦ Sequence diagrams
♦ Activity diagrams
What is modelling?

♦ Modelling consists of building an abstraction of reality.
♦ Abstractions are simplifications because:
  ♦ They ignore irrelevant details and
  ♦ They only represent the relevant details.
♦ What is relevant or irrelevant depends on the purpose of the model.
**Example: street map**
Why model software?

- Software is getting increasingly more complex:
  - Windows XP > 40 million lines of code.
  - A single programmer cannot manage this amount of code in its entirety.

- Code is not easily understandable by developers who did not write it.

- We need simpler representations for complex systems:
  - Modelling is a means for dealing with complexity.
Application and Solution Domain

- Application Domain (Requirements Analysis):
  - The environment in which the system is operating

- Solution Domain (System Design, Object Design):
  - The available technologies to build the system
Object-oriented Modelling

Application Domain (Phenomena)

Solution Domain (Phenomena)

System Model (Concepts) (Analysis)

System Model (Concepts) (Design)

TrafficControl

Aircraft

TrafficController

Airport

FlightPlan

MapDisplay

Summary Display

FlightPlanDatabase

TrafficControl

UML Package
What should be done first? Coding or Modelling?

♦ It all depends….
♦ Forward Engineering
  ♦ Creation of code from a model
  ♦ Start with modelling
  ♦ Greenfield projects
♦ Reverse Engineering
  ♦ Creation of a model from existing code
  ♦ Interface or reengineering projects
♦ Roundtrip Engineering
  ♦ Move constantly between forward and reverse engineering
  ♦ Reengineering projects
  ♦ Useful when requirements, technology and schedule are changing frequently.
What is UML? Unified Modelling Language

♦ Convergence of different notations used in object-oriented methods, mainly
  ♦ OMT (James Rumbaugh and colleagues), OOSE (Ivar Jacobson), Booch (Grady Booch)

♦ They also developed the Rational Unified Process, which became the Unified Process in 1999
Origins

- OO programming languages

- OO analysis and design techniques
  - business modelling
  - analysis of requirements
  - design of software systems

- UML: industry standard that merges the best features of different notations
What UML is not

♦ UML is not a programming language per se

♦ UML is not a software modelling tool

♦ UML is not a method, methodology or software development process
Why UML?

♦ De facto standard for OO modelling

♦ Unified modelling language

♦ UML provides extension mechanisms
Main diagram notations

- Use case diagrams
- Class diagrams and object diagrams
- Component diagrams
- Interaction diagrams
- Activity diagrams
- State machines
- Deployment diagrams
**UML overview**

- **Use case diagrams**
  - Describe the functional behaviour of the system as seen by the user.

- **Class diagrams**
  - Describe the static structure of the system: objects, attributes, associations.

- **Sequence diagrams**
  - Describe the dynamic behaviour between objects of the system.

- **Statechart diagrams**
  - Describe the dynamic behaviour of an individual object.

- **Activity diagrams**
  - Describe the dynamic behaviour of a system, in particular the workflow.
**UML Use Case Diagrams**

Used during requirements elicitation and analysis to represent external behaviour ("visible from the outside of the system")

An *Actor* represents a role, that is, a type of user of the system

A *use case* represents a class of functionality provided by the system

**Use case model:**
The set of all use cases that completely describe the functionality of the system.
An actor is a model for an external entity which interacts (communicates) with the system:

- User
- External system (Another system)
- Physical environment (e.g. Weather)

An actor has a unique name and an optional description.

Examples:

- Passenger: A person in the train
- GPS satellite: An external system that provides the system with GPS coordinates.
Use Case

- A use case represents a class of functionality provided by the system.
- Use cases can be described textually, with a focus on the event flow between actor and system.
- The textual use case description consists of 6 parts:
  1. Unique name
  2. Participating actors
  3. Entry conditions
  4. Exit conditions
  5. Flow of events
  6. Special requirements.

PurchaseTicket
**Textual Use Case Description Example**

1. **Name:** Purchase ticket

2. **Participating actor:** Passenger

3. **Entry condition:**
   - Passenger stands in front of ticket distributor
   - Passenger has sufficient money to purchase ticket

4. **Exit condition:**
   - Passenger has ticket

5. **Flow of events:**
   1. Passenger selects the number of zones to be traveled
   2. Ticket Distributor displays the amount due
   3. Passenger inserts money, at least the amount due
   4. Ticket Distributor returns change
   5. Ticket Distributor issues ticket

6. **Special requirements:** None.
Use Case Models should be packaged

Actor:

Use Case

Course
- GiveLecture
- HoldExercise
- DoHomework

System boundary

Instructor

Student

Teaching Assistant
Class Diagrams

♦ Class diagrams represent the structure of the system
♦ Used
  ♦ during requirements analysis to model application domain concepts
  ♦ during system design to model subsystems
  ♦ during object design to specify the detailed behaviour and attributes of classes.
A **class** represents a concept

- A class encapsulates state (**attributes**) and behaviour (**operations**)

Each attribute has a **type**

Each operation has a **signature**

The class name is the only mandatory information

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Attributes</th>
<th>Operations</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>TarifSchedule</td>
<td>Table zone2price</td>
<td>Enumeration getZones()</td>
<td>Price getPrice(Zone)</td>
<td>TarifSchedule</td>
</tr>
<tr>
<td>zone2price</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getPrice()</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getZones()</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Instances**

<table>
<thead>
<tr>
<th>tarif2006:TarifSchedule</th>
<th>:TarifSchedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>zone2price = {</td>
<td>zone2price = {</td>
</tr>
<tr>
<td>{‘1’, 0.20},</td>
<td>{‘1’, 0.20},</td>
</tr>
<tr>
<td>{‘2’, 0.40},</td>
<td>{‘2’, 0.40},</td>
</tr>
<tr>
<td>{‘3’, 0.60}}</td>
<td>{‘3’, 0.60}}</td>
</tr>
</tbody>
</table>

- An *instance* represents a phenomenon
- The attributes are represented with their *values*
- The name of an instance is *underlined*
- The name can contain only the class name of the instance (anonymous instance)
Actor vs Class vs Object

♦ Actor
  ♦ An entity outside the system to be modelled, interacting with the system (“Passenger”)

♦ Class
  ♦ An abstraction modelling an entity in the application or solution domain
  ♦ The class is part of the system model (“User”, “Ticket distributor”, “Server”)

♦ Object
  ♦ A specific instance of a class (“Joe, the passenger who is purchasing a ticket from the ticket distributor”).
Associations denote collaborations between classes by means of message exchange.

The multiplicity of an association end denotes how many objects the instance of a class can legitimately reference.
Association properties

♦ Name
♦ Multiplicity: number of object instances of the class at the far end of an association for one instance of the class at the near end of an association
♦ Role names
  ♦ role played by a class in an association
  ♦ useful to specify methods
Aggregation

- An **aggregation** is a special case of association denoting that one class may consist of, or include, instances of another class.
- A solid diamond denotes **composition**: the *life time of the component instances* is controlled by the aggregate.
Inheritance

Inheritance is another special case of an association denoting a “kind-of” hierarchy.

- Inheritance simplifies the analysis model by introducing a taxonomy.
- The children classes inherit the attributes and operations of the parent class.
Packages

- Packages help you to organize UML models to increase their readability.
- We can use the UML package mechanism to organize classes into subsystems.
- Any complex system can be decomposed into subsystems, where each subsystem is modelled as a package.
Sequence Diagrams

- Used during analysis
  - To refine use case descriptions
  - To find additional objects ("participating objects")
- Used during system design

- Instances are represented by rectangles.
- Actors by sticky figures.
- Lifelines are represented by dashed lines.
- Messages are represented by arrows.
- Activations are represented by narrow rectangles.

Messages -> Operations on participating Object

Select Zone
Pickup Change
Pick up Ticket

Focus on control flow
Sequence Diagrams can also model the Flow of Data

- The source of an arrow indicates the activation which sent the message
- Horizontal dashed arrows indicate data flow, for example return results from a message
Sequence Diagrams: Iteration & Condition

- Iteration is denoted by a * preceding the message name
- Condition is denoted by boolean expression in [ ] before the message name

...continued from previous slide...

...continued on next slide...

- Iteration is denoted by a * preceding the message name
- Condition is denoted by boolean expression in [ ] before the message name
Creation and destruction

- Creation is denoted by a message arrow pointing to the object
- Destruction is denoted by an X mark at the end of the destruction activation
  - In garbage collection environments, destruction can be used to denote the end of the useful life of an object.

...continued from previous slide...
Sequence Diagram Properties

- UML sequence diagram represent *behaviour in terms of interactions*
- Useful to identify or find missing objects
- Time consuming to build, but worth the investment
- Complement the class diagrams (which represent structure).
**Activity Diagrams**

- An activity diagram is a special case of a state chart diagram
- The states are activities ("functions")
- An activity diagram is useful to depict the workflow in a system
Activity Diagrams allow to model Decisions

Open Incident → Decision
- [lowPriority] → Allocate Resources
- [not fire & highPriority] → Notify Police Chief
- [fire & highPriority] → Notify Fire Chief
Activity Diagrams can model Concurrency

- Synchronization of multiple activities
- Splitting the flow of control into multiple threads
Activities may be grouped into **swimlanes** to denote the object or subsystem that implements the activities.
UML Summary

♦ UML provides a wide variety of notations for representing many aspects of software development
  ♦ Powerful, but complex
♦ UML is a programming language
  ♦ Can be misused to generate unreadable models
  ♦ Can be misunderstood when using too many exotic features
♦ We concentrated on a few notations:
  ♦ Functional model: Use case diagram
  ♦ Object model: class diagram
  ♦ Dynamic model: sequence diagrams, statechart and activity diagrams
Additional References

♦ Martin Fowler

♦ Grady Booch, James Rumbaugh, Ivar Jacobson

♦ Commercial UML tools
  ♦ Rational Rose XDE for Java
  ♦ Together (Eclipse, MS Visual Studio, JBuilder)

♦ Open Source UML tools
  ♦ [http://java-source.net/open-source/uml-modeling](http://java-source.net/open-source/uml-modeling)
  ♦ ArgoUML,UMLet,Violet, …