EECE 310: Software Engineering

A Brief Introduction to the UML

adapted form Philippe Kruchten’s slides
Outline

- Purpose & genesis
- Reminder on objects and classes
- UML elements
- Key UML Diagrams
- From Notation to Code
- UML Tools
- UML References & resources
What is a model?

- A miniature representation of something.
- A semantically closed abstraction of a system under study.
- A representation of a system that allows for investigation of the properties of the system.
What is UML?

- Notation
  - Electronics analogy
  - Map analogy
- Syntax and semantics
  - Casual notation and formal notation
- Usage:
  - Illustration
  - Forward engineering: Model $\Rightarrow$ Code
  - Round-trip engineering: Model $\Leftrightarrow$ Code
What UML is not

- *Not a method* in itself
  - A notation designed to support various methods for requirement analysis and software design
    - E.g., (IBM) Rational Unified Process (RUP)
A Brief History of UML

- Language ‘wars’ (1985-95):
  - OOPSLA conferences as the main battlefield
- Contenders
  - OMT (Jim Rumbaugh)
  - Booch method and notation (Grady Booch)
  - OOSE (Ivar Jacobson)
  - OML (Brian Henderson-Sellers)
  - And many others.
- Rational Software and the “three amigos”
- Object Management Group (OMG)
Two types of UML diagrams

Static Diagrams
- Object Diagrams
- Component Diagrams
- Deployment Diagrams

Dynamic Diagrams (Behavior and Interaction)
- Sequence Diagrams
- Use-Case Diagrams
- Class Diagrams
- Collaboration Diagrams
- Statechart Diagrams
- Activity Diagrams
Key UML diagrams

- Class diagram
- Sequence diagram
- Object diagram
- State diagram or Statechart
- Activity diagram
- Deployment diagram
- Use-case diagram
- Collaboration diagram

In decreasing order of usefulness for the average developer.
Visual modeling of a software system

Use-Case Diagram

Class Diagram

State Diagram

Deployment Diagram

Collaboration Diagram

Sequence Diagram

Component Diagram

Forward and Reverse Engineering
Three views over the system

- **Functional requirements view**
  - Emphasizes the functional requirements of the system from the user's point of view.
  - Includes **use case diagrams**.

- **Static structural view**
  - Emphasizes the static structure of the system using objects, attributes, operations, and relationships.
  - Includes **class diagrams** and collaboration diagrams.

- **Dynamic behavior view**
  - Emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects.
  - Includes **sequence diagrams**, **activity diagrams** and **state machine diagrams**.
Elements of UML Diagrams

- Model elements
- Connectors
- Adornments
- Annotations
Reminder: Class

- A description of a set of objects that share the same attributes, operations, methods, relationships, and semantics.

- A class may use a set of interfaces to specify collections of operations it provides to its environment.
Reminder: Object

- An entity with a well-defined boundary and identity that encapsulates *state* and *behavior*.
  - State is represented by attributes and relationships;
  - Behavior is represented by operations, methods, and state machines.
- An object is an instance of a class.
Modeling elements: class, interface

Class

<table>
<thead>
<tr>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ origin : Point</td>
</tr>
<tr>
<td>+ move(p : Point)</td>
</tr>
<tr>
<td>+ resize(s : Scale)</td>
</tr>
<tr>
<td>+ display()</td>
</tr>
<tr>
<td># invalidateRegion()</td>
</tr>
</tbody>
</table>

Responsibilities:
-- manage shape state
-- handle basic shape transformations

Interface

objects are underlined
abstract elements are in italics.

name

attributes

signature

operations

extra compartment

IApplication
Types of relationship in class diagrams

- Class level:
  - Dependency:
    - $x \text{ depends on } y$ (for implementation, for example)
    - A dependency exists between two defined elements if a change to the definition of one would result in a change to the other.
  - Generalization (& specialization):
    - $x \text{ is a kind of } y$ (taxonomy, subclassing)

- Instance level
  - Association:
    - $x \text{ is a part of } y$
Class Level: Dependency and generalization

Dependency

name

source  →  Owner

target

Generalization

discriminator

child  →  power

parent
Instance-Level Relationships

Link

- The basic relationship among objects.
  - Represented as a line connecting *two or more* object boxes.
  - Shown on an object diagram or class diagram.
  - A link is an instance of an association.
Association

- A relationship that models a bi(or multi)-directional semantic connection among instances.
- An association represents a family of links
Multiplicity

- How many object can be associated
- 1 = exactly one
- 0 .. 1 = optional (zero or one)
- 1 .. N = at least one
- * = 0 .. N = any number
- N
  - For example 4, for 4 wheels in car
- m .. n
Association example

Person 0..* subscribes 0..* Magazine
+subscriber +subscribed magazine
Two Special Associations

- Aggregation = grouping (e.g., “by reference”)

- Composition = is made of (e.g., “by value”)

[Diagram of UBC logo]
Composition versus Aggregation

Course \(?\) Instructor

Car \(?\) Carburetor
Composition versus Aggregation

Course \[\text{Aggregation}\] Instructor

Car \[\text{Composition}\] Carburetor
Composition versus Aggregation

- Assume A associated with B

- If I destroy an object A, is the associated B also destroyed?
  - Yes? you probably have a composition

- If an object A1 is associated with object B1, can the same object B1 be also associated with another object A2
  - Yes? you probably have an aggregation
Composition vs. Aggregation

- The whole of a composition must have a multiplicity of 0..1 or 1, indicating that a part must be for only one whole.
  - The whole of an aggregation may have any multiplicity.

- Example:
  - represent real-world whole-part relationships,
    - e.g., an engine is part of a car,
    - → the composition relationship is most appropriate.
  - represent database relationship,
    - e.g., car model engine ENG01 is part of a car model CM01,
    - → an aggregation relationship is best, (as the engine ENG01 may be also part of a different car model)
packages

package

name

business rules

extra compartments may be used to show contents
Other major modeling elements

Active Class
- EventManager
  - q : EventQueue
  - suspend()
  - flush()

Component
- spelling.java

Node
- Server

extra compartments may be used to show contents
Collaborations and Use cases

Collaboration
- Chain of responsibility
- extra compartments may be used to show contents

Use Case
- Place order

name
Sequences
States
Consider the use of the broker design pattern here. *egb 12/11/97*
Class Diagram (3)

- Basic Elements
- Process Structure
- Process Lifecycle
- Names
- Guidance
- Process Components
More class diagrams

Airport
- name: String
  - origin
  - destination
  - departing Flights
  - arriving Flights

Flight
- departTime: Time
- arrivalTime: Time
- duration: Interval
- maxNrPass: Integer

Airline
- name: String
- flights
  - airline

Passenger
- $minAge: Integer
- age: Integer
- needsAssistance: Boolean
- book(f: Flight)
Class diagram with operations

**Switch**
- name : string
  - manages
  - controls
  - manages
  - controls

**Simulator**
- processCmds(in : istream)
  - controls

**Phone**
- state : PhoneState
  - name : string
  - selectLine(num : int) : void
  - pickup() : void
  - hangup() : void
  - ring() : void
  - dial(num : int) : void
  - addLine(l : Line) : void
  - onConnection

**Line**
- number : int
  - state : LineState
  - incomingCall(c : Connection) : void
  - hungUp(p : Phone) : void
  - pickedUp(p : Phone) : void
  - addPhone(p : Phone) : void
  - linkedLines

**Connection**
- id : int
  - state : ConnState
  - duration : int
  - disconnect(l : Line) : void
  - addLine(l : Line) : void

**Switch**
- manages
- controls

**Simulator**
- controls

**Phone**
- manages

**Line**
- selected

**Connection**
- linkedLines

**Simulator**
- manages

**Phone**
- manages
Sequence diagram

c : Client

create

actual parameter

call

setItinerary(i)

return

return value

route

«destroy»

p : PlanningAssistant

calculateRoute()

(local invocation)

call

send

notify()
Sequence diagram

```
p : StockQuotePublisher

attach(s1)

notify()

update()

getState()

s1 : StockQuoteSubscriber

s2 : StockQuoteSubscriber

attach(s2)

update()

getState()
```
Collaboration diagram

Sequence diagram
Sequence Diagram for a Phone Call

- caller
  - lift receiver
  - dial tone
  - dial
  - ringing tone
  - stop tone

- phoneNet
  - ring
  - answer

- receiver
  - stop ringing

- accounting
  - start
State diagram

Lamp On

Lamp Off

on

off

on

off
Instances of a Class (Objects)

<table>
<thead>
<tr>
<th>triangle : Polygon</th>
</tr>
</thead>
<tbody>
<tr>
<td>center : Point = (2,2)</td>
</tr>
<tr>
<td>vertices : Point* = ((0,0), (4, 0), (2,4))</td>
</tr>
<tr>
<td>borderColor : Color = black</td>
</tr>
<tr>
<td>fillColor : Color = white</td>
</tr>
</tbody>
</table>

Underlined name

Attribute links

triangle : Polygon

: Polygon
Deployment diagram

- Log Device
- Cash Dispenser
- Display
- Receipt Printer
- Card Reader
- Keypad
- Network Interface
- Processor: *200 Mhz Pentium
  Memory: *64 Mb
- ATM Main
  - Customer Interface
  - ATM Network Interface
  - Device Controller
- ATM Network Server
- Network Interface
- T-1 network connection

UBC
Use-case diagram
Component Diagram

- Page
- Find.html
- Executable
- Index.html
- Find.exe
- Dbacs.dll
- Nateng.dll
- Component
- Library

«hyperlink»
UML Extensions

```
 stereotype
<container>
  ActionQueue
  {version = 3.2}
  add(a : Action)
  remove(n : Integer)
  «query»
  length() : Integer
  «helper functions»
  reorder()
```

tagged value
{add runs in O(1) time}

constraint
Tools to “do UML”

- *Pen and pencil (and eraser) + scanner*
- *White board and digital camera*
- IBM Rational Rose & Rose XDE ($$)
- IBM Rational Software Architect ($$$$$$
- Microsoft Visio ($)
  - use the free Pavel Hruby stencil: http://www.softwarestencils.com/uml/index.html
- Eclipse UML Plug-in (free)
- Visual Paradigm (free)
- Together Designer (Borland) (free)
- ArgoUML (free)
- …. and many more, mostly not free
References

- **Books:**
  - Martin Fowler: *UML Distilled, 3rd ed.*, AWL
  - Grady Booch: *UML User’s Guide*, AWL

- **On-line Tutorial & Resources**
  - Tutorial David Braun *et al.* (Kennesaw St. U) at: http://pigseye.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/
  - Scott Ambler’s http://www.agilemodeling.com/
  - http://www.uml.org/