UML basics
Corso di Laurea Magistrale in Ingegneria Informatica

Lecturer: Valentina Presutti
Academic Year: 2009/2010
Courtesy of Paolo Ciancarini
The soul never thinks without an image

Aristotle, *De Anima*
Agenda

- Modeling notations
- UML 1.* and UML 2.*
- Models and Views
- Basic diagrams
- Structures, behaviors, interactions
- Case study
Modeling - a foundation of all engineering disciplines

Models improve communication and understanding about:
- **why** the system is needed;
- **what** its functionality should be;
- **how** it should be implemented.
A modeling engineering spiral

Domain

has needs

Model

Domain models

Develop

System

Manufacture

System models

Deploy

quality = satisfaction of needs

quality = satisfaction of needs
Visual documentation

• Software models are artifacts like...

Mechanical designs  Electrical schemata  Blueprints
Visual modeling a business process

1. Customer places order
2. Sales Representative checks order
3. Item is fulfilled
4. Item is shipped via delivery
5. Business process is completed
UML: a modeling language

- A **modeling language** allows the specification, the visualization and the documentation of a software development process
- The **models** are **artifacts** which clients and developers use to communicate
- The most used **modeling languages** are standard (eg. UML is a standard by OMG)
- UML 1.* is a modeling language
- UML 2.0 is also a programming language
Unified Modeling Language

An industrial standard (OMG) notation to:

• Model a business, its roles and processes
• Write the requirements of a software system
• Describe its software architecture
• State the structure and behavior of a software artifact
• Document a software application
• Generate automatically an implementation
OMG

- Object Management Group
- Consortium of industries and interested universities
- Produces specifications of reference architectures, eg. CORBA
- UML has been adopted by OMG as standard *de facto*
Roots of UML

At the beginning of the ’90 there was a convergence:

- **Booch method** (Grady Booch)  
- **OMT** (Jim Rumbaugh)  
- **Fusion/OOSE** (Ivar Jacobson)

'94 – join *Rational Software Corporation*

'95 – joins Rational
The “tre amigos”

- Booch: analysis by objects
- Rumbaugh: Object Modeling Technique (OMT)
- Jacobson: process Objectory
- In 1994-95 they define for Rational both UML and UP

The three amigos:
Grady Booch, Jim Rumbaugh, Ivar Jacobson
UML evolution

Approved OMG 2005
Standard ISO/IEC 19501
IBM buys Rational, 2003
OMG, 2001; ISO, 2005
OMG, jan ’97
Beta version OOPSLA ’96
WWW - June ’96
OOPSLA ’95
Unified Method 0.8
Draft
UML 0.9
UML 1.1
UML 1.4
UML 1.5
UML 2.0
3 Amigos books:
-- User Guide
-- Reference Manual
-- Process Book
on Web

other methods
Booch
OMT
OOSE/Objectory

public comments
UML History

• OO languages appear, since mid 70’s to late 80’s
• Between ’89 and ’94, OO methods increased from 10 to 50
• Unification of ideas began in mid 90’s.
  • 1994 Rumbaugh joins Booch at Rational
  • 1995 v0.8 draft Unified Method
  • 1995 Jacobson joins Rational (Three Amigos)
  • 1996 June: UML v0.9 published
  • 1997 Jan: UML 1.0 offered to OMG
  • 1997 Jul: UML 1.1 OMG standard
    • Maintenance through OMG RTF
  • 1998 June: UML 1.2
  • 1999 Sept: UML 1.3
  • 2001 Sept: UML 1.4

• 2003 Feb: IBM buys Rational
  • 2003 March: UML 1.5
  • 2004: UML 1.4.2 becomes the standard ISO/IEC 19501
  • 2005: UML 2.0
  • 2007 Nov: UML 2.1.2
Main UML specification documents

- **Superstructure**: defines the UML elements (diagrams, etc)
- **Infrastructure**: defines the UML metamodel
- **OCL** (Object Constraint Language): formal language for writing constraints and formulas
- **XMI** (XML Metadata Interchange): DTD for UML models
- **UML Diagram Interchange**: XMI + graphic info
Canonical diagrams (vers 1.5)

- Use case
- **Class** (Object diagrams are class diagrams without classes 😊)
- Behavior
  - Statecharts
  - Activity
  - Interaction
    - Sequence
    - Collaboration
- Implementation
  - Components
  - Deployment
Version 2.0 includes 13 canonical diagrams

- **Structure**
  1. Class
  2. Composite structure
  3. Component
  4. Deployment
  5. Object
  6. Package

- **Behavior**
  1. Activity
  2. Statecharts
  3. Usecase

- **Interaction**
  1. Communication
  2. Interaction Overview
  3. Sequence
  4. Timing
Structure and behavior

- UML focuses on two aspects of object oriented design: structure and behavior
- It aims at visualizing both

Tour Eiffel (1889)  
G. Balla: Dinamismo di cane al guinzaglio (1912)
Discuss

• Which ways do you know to pictorially describe “behaviors” - or actions?
Lullaby

Souvenir coudible

PARTIES
TOURNOI INTERCLUBS DE PARIS
183. — Défense des deux cavaliers.
Le 24-2-1929.

O. S. BERNSTEIN—J. Cukierman

1. ♙ ♦ 2. ♦ ♦ 3. △ ♦ 4. ♦ ♦ 5. 0—0

La suite 5., △× ♦; 6. ♦ ♦, ♦ ♦; 7. ♦ ♦, ♦ ♦; 8. ♦ ♦; etc., conduit en fin de compte à l'égalité. Le coup du texte assure par contre aux Blancs une supériorité en espace dans la première phase de la partie.

6. ♦ ♦ 7. ♦ ♦ 8. ♦ ♦
Discuss

• Are structures and behaviors all we need for software design?
Example

• The structure of a chess program could be stand-alone, client-server, agent based etc.
• Its behavior should be coherent with the rules of chess
• What is its goal? To play and win a chess game against an opponent
• This is its function
Example

• The very same chess program, with identical structure and behavior, could be used with a different function?
• For instance, could it be used to learn to play chess?
• Or to write a chess book, like a chess game editor?
• Or to play a game of loser’s chess (where the winner is the one who is checkmated)?
Function

• A software system is designed with some *functional requirements* in mind
• UML has a specific diagram for this: Use Cases

<table>
<thead>
<tr>
<th>Use Cases</th>
<th>Interaction Diagrams</th>
<th>Class Diagrams</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Description</td>
<td>State Diagrams</td>
<td>Deployment Diagrams</td>
</tr>
<tr>
<td>Overview of process goals, participants and collaboration.</td>
<td>Life cycle of core system elements in process.</td>
<td>Actual structure of the final system</td>
</tr>
</tbody>
</table>

**Function**  
**Behavior**  
**Structure**
Modeling Requirements in UML

• Use case diagram
  – Describes the main user stakeholders
  – Describes the externally observable behavior of system functions, usually to define system requirements
  – Describes interactions between the system and external entities, including users and other systems
Use Case: elements

- **Student**
- **Register**
- **Check Grades**
- **Validate User**

**Actor:** Student

**Association:**

**System Boundary:**

**Use Case:**

<<include>>
Elements of a Use Case Diagram

• Actor:
  – Represents a role played by external entities (humans, systems) that interact with the system

• Use case:
  – Describes what the system does (i.e., functionality)
  – Scenario: sequence of interactions between the actors and the system

• Relationships:
  – Association between actors and use cases
  – Extension (or generalization) among actors
  – Dependency among use cases: include and extend
Example
Use Case Scenarios

**Use Case:** Check Grades

**Description:** View the grades of a specific year and semester

**Actors:** Student

**Precondition:** The student is already registered

**Main scenario:**

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
</tr>
</thead>
</table>
| 3. The user enters the year and semester, e.g., Fall 2007. | 1. The system carries out “Validate User”, e.g., for user “miner” with password “allAs”.
2. The system prompts for the year and semester. |
| | 4. The system displays the grades of the courses taken in the given semester, i.e., Fall 2007. |

**Alternative:**
The student enters “All” for the year and semester, and the system displays grades of all courses taken so far.

**Exceptional:**
The “Validate User” use case fails; the system repeats the validation use case.
Exercise

• Draw a use case diagram and a related scenario for the following situation:

• A user can borrow a book from a library; a user can give back a book to the library
Structure diagrams
Object-Oriented Modeling

- Use **object-orientation** as a basis of modeling
- Models a system as a **set of objects** that interact with each other
- No semantic gap, seamless development process
Key Ideas of OO Modeling

• Abstraction
  – Mechanisms to hide minor details so to focus on major details

• Encapsulation
  – Modularity: principle of separation of functional concerns
  – Information-hiding: principle of separation of design decisions

• Relationships
  – Association: relationship between objects or classes
  – Inheritance: relationship between classes, useful to represent
generalizations or specializations of objects

• Object-oriented language model
  = object (class) + inheritance + message send
The basic **building blocks** of UML

- **Elements**: domain modeling concepts
- **Relationships**: connection between model elements that adds semantic information to a model
- **Diagrams**: collections of entities and relationships representing some “perspective” on a model
Basic building blocks - Things

- **UML 1.x**
  - **Structural** — nouns/static of UML models (irrespective of time)
  - **Behavioral** — verbs/dynamic parts of UML models.
  - **Grouping** — organizational parts of UML models.
  - **Annotational** — explanatory parts of UML models.
Structural elements - 7 Kinds (Classifiers)

- Nouns in the requirements
- Conceptual or physical elements

Class

- Student
  - std_id
  - grade
  - changeLevel()
  - setGrade()
  - getGrade()

Active Class (processes/threads)

- Event Mngr
  - thread
  - time
  - Start
  - suspend()
  - stop()

Component (replaceable part, realizes interfaces)

- Interface (collection of externally visible ops)
  - IGrade
    - setGrade()
    - getGrade()

Collaboration (chain of responsibility shared by a web of interacting objects, structural and behavioral)

Use Case (a system service -sequence of Interactions w. actor)

Node (computational resource at run-time, processing power w. memory)
1. Associations

*Structural* relationship that describes a set of links, a link being a connection between objects. Variants: aggregation and composition

2. Generalization

A specialized element (the child) is more specific than the generalized element

3. Realization

One element guarantees to carry out what is expected by the other element. (e.g., interfaces and classes/components; use cases and collaborations)

4. Dependency

A change to one thing (independent) may affect the semantics of the other thing (dependent) (direction, label are optional)
Class

- Defines the structure of the states and the behaviors shared by all the instances
- Defines a template for creating instances
  - Names and types of all fields
  - Names, signatures, and implementations of all methods
Class diagram

• Most common diagram in OO modeling
• Describes the static structure of a system
• Consists of:
  – Nodes representing classes
  – Links representing of relationships among classes
    • Inheritance
    • Association, including aggregation and composition
    • Dependency
Notation for classes

- The UML notation for classes is a rectangular box with as many as three compartments.

<table>
<thead>
<tr>
<th>ClassName</th>
</tr>
</thead>
<tbody>
<tr>
<td>field₁</td>
</tr>
<tr>
<td>....</td>
</tr>
<tr>
<td>fieldₙ</td>
</tr>
<tr>
<td>method₁</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>methodₙ</td>
</tr>
</tbody>
</table>

The top compartment shows the class name.
The middle compartment contains the declarations of the fields, or attributes, of the class.
The bottom compartment contains the declarations of the methods of the class.
Example

A point defined by classes at three different abstraction levels

<table>
<thead>
<tr>
<th>Point</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
</tr>
</tbody>
</table>

```
+ move(dx: int, dy: int): void
```
Example

Document

Docuemnt

| Pages[] |
| nPages |
| display |

Point

| - Pages: array of Page |
| - nPages: int |
| + display(k:int, p:Page): void |
Field and Method Declarations

- Field declarations
  - birthday: Date
  - +duration: int = 100
  - -students[1..MAX_SIZE]: Student
- Method declarations
  - +move(dx: int, dy: int): void
  - +getSize(): int

<table>
<thead>
<tr>
<th>Visibility</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>+</td>
</tr>
<tr>
<td>protected</td>
<td>#</td>
</tr>
<tr>
<td>package</td>
<td>~</td>
</tr>
<tr>
<td>private</td>
<td>-</td>
</tr>
</tbody>
</table>
Visibility
Exercise

Draw a class diagram for the following Java code

class Person {
    private String name;
    private Date birthday;
    public String getName() {
        // ...
    }
    public Date getBirthday() {
        // ...
    }
}
# Objects vs. Classes

<table>
<thead>
<tr>
<th></th>
<th>Interpretation in the Real World</th>
<th>Representation in the Model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object</strong></td>
<td>An <em>object</em> represents anything in the real world that can be distinctly identified.</td>
<td>An <em>object</em> has an identity, a state, and a behavior.</td>
</tr>
<tr>
<td><strong>Class</strong></td>
<td>A <em>class</em> represents a set of objects with similar characteristics and behavior. These objects are called <em>instances</em> of the class.</td>
<td>A <em>class</em> defines the structure of states and behaviors that are shared by all of its instances.</td>
</tr>
</tbody>
</table>
Object = Identity + State + Behavior

• Identity
  – Distinguishes an object from all other objects.

• State
  – Consists of a set of attributes (or fields), which have names, types, and values

• Behavior
  – Defined by the set of operations (or methods) that may operate on the object
  – Each method has a name, a type, and a value, where
    • The type consists of the return type and the list of parameter types of the method, often called signature.
    • The value is the implementation of the method often expressed as a sequence of statements, in languages like Java and C++
Notation for Objects

- Object: Rectangular box with one or two compartments
- Object name: object name/role name:class name.

<table>
<thead>
<tr>
<th>objectName: Classname</th>
</tr>
</thead>
<tbody>
<tr>
<td>field₁ = value₁</td>
</tr>
<tr>
<td>......</td>
</tr>
<tr>
<td>fieldₙ = valueₙ</td>
</tr>
</tbody>
</table>

The top compartment shows the name of the object and its class.

The bottom compartment contains a list of the fields and their values.

<table>
<thead>
<tr>
<th>p₁:Point</th>
<th>:Point</th>
<th>P2/origin:Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 10</td>
<td>x = 30</td>
<td>x = 20</td>
</tr>
<tr>
<td>y = 20</td>
<td>y = 30</td>
<td>y = 30</td>
</tr>
</tbody>
</table>
Association

• General binary relationship between classes or objects
• Represented as a line between boxes

Diagram:

- Class A (role A) connected to Class B (role B) with a line labeled "multiplicity A - label - multiplicity B".
- Boxes labeled "Student" and "Course" connected by the line.
- Boxes labeled ":John" and ":SwEng" connected by the line.
Association

• An association line may have an optional label consisting of a name and a direction
• The direction arrow indicates the direction of association with respect to the name

An arrow may be attached to the end of path to indicate that navigation is supported in that direction
Association

- An association line may have an optional role name and an optional multiplicity specification.
- The multiplicity specifies an integer interval, e.g.,
  - $l..u$ closed (inclusive) range of integers
  - $i$ singleton range
  - $0..*$ entire nonnegative integer, i.e., 0, 1, 2, …
Association example

- A **Student** can take up to **five Courses**
- Every Student has to be enrolled in at least **one course**
- Up to **300** students can enroll in a course
- A class should have **at least 10 students**
Example

- A teacher teaches 1 to 3 courses
- Each course is taught by only one teacher
- A student can take between 1 to 5 courses
- A course can have 10 to 300 students
Net data structures

• How we can represent a net of objects from the same class?

• A class with an association to itself with both ends of the association marked with 0..*
Hierarchic data structures

• How we can represent a hierarchy of objects from the same class?
• A class with an association to itself with one end of the association marked with 0..* (children) and the other as 0..1 (parent)
Exercise

Explain the meaning of this diagram
Aggregation

• An *aggregation* is a special form of association representing *has-a* or *part-whole* relationship.
• It distinguishes the whole (aggregate class) from its parts (component class).
• WARNING: an aggregation does not bind the parts’ lifetime to the whole (they can exist separately).

```
Whole                                    Part
                               
Course                                    Students
```
Hierarchic file system

- A directory can contain any number of elements (either a file or a directory)
- Any element is part of exactly one directory
Non-hierarchic file system

- A directory can contain any number of elements (either a file or a directory)
- An element can be part of many directories
Composition

- A *composition* is a stronger form of aggregation
- It implies exclusive ownership of the component class by the aggregate class
- The lifetime of the parts is entirely included in the lifetime of the whole (a part can not exist without its whole)
Example
Exercise

Imagine some possible aggregation or composition relationships among the following classes and draw a corresponding class diagram

– Employee
– Manager
– Office
– Department
Dependency

• Relationship between the entities such that the proper operation of one entity depends on the presence of the other entity, and changes in one entity would affect the other entity

• The common form of dependency is the use relation among classes
Example

Dependencies are often omitted from the diagram unless they convey some significant information.
Inheritance

• Important relationship in OO modeling
• Defines a relationship among classes or interfaces
• Three kinds of inheritance
  – extension relation between two classes (subclass and superclass)
  – extension relation between two interfaces (subinterface and superinterface)
  – implementation relation between a class and an interface
Inheritance in UML

- An extension relation is called *specialization* and *generalization*
- An implementation relation is called *realization*
Class and superclass

Point

Colored Point

Point

x
y
Move

Colored Point

... color
...
SetColor
Notation for Interfaces

interface Drawable {
    void draw(Graphics g);
}

interface Drawable {
    void draw(Graphics g);
}
Interfaces

• A class and an interface differ: a class can have an actual instance of its type (but can also have zero instances), whereas an interface must have at least one class to implement it

• **Example**: both the Professor and Student classes implement the Person interface
Example

- Student
  - No-degree
  - Undergraduate
  - Graduate
    - Master
    - PhD
Exercise

• Draw a class diagram showing possible inheritance relationships among classes Person, Employee, and Manager

• Draw a class diagram showing possible inheritance relationships among classes Person, Student, Professor, and Assistant
Real example: DOM

Invoke "getName" to read instance variable name when using XML DOM or XML4J.
Real example: Facebook
Strange examples
Behavior diagrams

Duchamp: Nude descending a staircase
Modeling Behavior

• Statechart diagram
  – Depicts the flow of control inside an object using states and transitions (finite state machines)

• Activity diagram
  – Describes the control flow among objects by actions organized in workflows (Petri Nets)

• Sequence diagram
  – Depicts objects’ interaction by highlighting the time ordering of method invocations

• Communication (collaboration) diagram
  – Depicts the message flows among objects
Behavioral elements

- Verbs in the requirements
- Dynamic parts of UML models: “behavior over time”
- Usually connected to structural elements

Two primary kinds of behavioral elements:

- **Interaction**
  a set of objects exchanging messages, to accomplish a specific purpose.

- **State Machine**
  specifies the sequence of states an object goes through during its lifetime in response to events
State diagram

- Graph: net of states (nodes) and transitions (arrows)
- Graph representing a finite state machine
- Useful for modeling a reactive (event-driven) system
- Animation by “token game”
State diagram: elements

Initial State

Idle

Transition

Running

Final State

State
Example: Unix process

- fork
- fork
- Pre-empted
- scheduled
- Sys call
- Sys return
- Waiting
- Running
- end
State

• Situation in the life of an object (or system) during which it:
  – Satisfies some condition,
  – Performs some activity, or
  – Waits for some events

• Set of values of properties that affect the behavior of the object (or system)
  – Determines the response to an event,
  – Thus, different states may produce different responses to the same event
Elements of a state diagram

- **StateName**: do/action
- **superstate**: State1/do/action1 -> State2/do/action2 -> State3/do/action3
- **event**: [condition]
- **activity**: entry/start/exit/stop
- **transition**: digit(n) / [number.isValid()]
Transition

• Relationship between two states indicating that a system (or object) in the first state will:
  – Perform certain actions and
  – Enter the second state when a specified event occurs or a specified condition is satisfied

• A transition consists of:
  – Source and target states
  – Optional event, guard condition, and action
Definition

• Event
  – An occurrence of a stimulus that can trigger a state transition
  – Instantaneous and no duration
• Action
  – An executable atomic computation that results in a change in state of the model or the return of a value
Example

- Normal
  - Anomaly
    - Pressure
      - Recovery Success
    - Temperature
      - Recovery Failure
  - Recovery Identification
    - Recovery Failure
    - Recovery Success
  - Pressure Recovery
    - Recovery Failure
    - Recovery Success
  - Temperature Recovery
    - Recovery Failure
    - Recovery Success
Composite states

- Normal
- Pressure Recovery
- Temperature Recovery
- Recovery Identification

- Anomaly
- Recovery success
- Recovery failure

- Pressure
- Temperature
Composite state

- Used to simplify diagrams
- Inside, looks like a statechart
- May have composite transitions
- May have transitions from substates
- Sequential and parallel
Composites and transitions

Transition to/from composite state

Transition from substate

Idle

Maintenance

Active

Validating

Selecting

Processing

Printing
Including composite states

- **Dial Number**
  - Include / Dialing

- **Dialing**
  - **Start**
    - entry / start dial tone
    - exit / end dial tone

- **Partial Dialing**
  - entry / number.append(n)

  - [number.isValid()]
Parallel composition

• Concurrency (multiple threads of control)
• Synchronization

Diagram:
- Superstate
- Substate 1
- Substate 2
- Substate 3
- Substate 4
Example

Diagram showing a workflow with the following stages:

- **Incomplete**
  - Mod1
  - Mod2

- **Project**
  - Midterm
  - Final

- **Passed**
- **Failed**

The diagram indicates that failing any stage can lead to failure at the end.
History pseudo state

- A **history** pseudostate represents the most recent active substate of its containing state.
- There are two kinds of this pseudostate: **shallow** or **deep** (See the Superstructure sect 15.3.8 pag 541 for their definition).
Consistency among diagrams

Student

DegreeProgram

Sport

FirstName
LastName
Age
Role

enrolls *

* practices

1

Enrolls a different program

Finishes education program

Changes career

Changes role

Studies

Graduated

Sporty
Exercise: Cellular Phone

• Draw a statechart describing the operation of a cellular phone. Assume that the phone has keys for:
  – power on and off
  – keypad locking and unlocking
  – 0-9, #, and *
  – talk (or send) and end

Model the following operations:
  – power on/off
  – keypad locking/unlocking
  – making calls (e.g., dialing, connecting, talking),
  – receiving calls (e.g., ringing, talking)
A special kind of state diagram that shows the flow from activity to activity

- **Initialize course**
  - **Add student**
    - **Notify Registrar**
    - **Notify Billing**
      - **[count < 10]**
        - **[else]**
          - **Close course**

- **fork/spawn**
- **synchronization**
- **guard**
- **initial**
- **activity**
- **final**
Example
Example
Example

- **[baggage]**
  - Receive baggage and print receipt
  - Print boarding card

- **[no baggage]**
  - Conditional thread
    - [frequent flyer member]
      - Award frequent flyer miles

- Conditional thread

Example: business plan
Activity partition

- Partitions divide the nodes and edges to constrain and show a view of the contained nodes.
- Partitions can share contents; they often correspond to organizational units in a business model.
Swimlanes in an activity diagram showing a workflow
State vs activity diagrams

• Both diagrams describe behaviors, by state changes and actions, respectively
• In UML1 they are equivalent (in AD states are actions)
• In UML2 they differ: ActivityD are based on Petri Nets, StateD on Harel automata
• Also their typical usage is different: SD are single context, AD multiple context
State vs activity diagrams

White

Black

Pe2e4

Pe7e5

Cg1f3

Cb8c6
Exercise

Which is the maximum degree of parallelism in this activity diagram?
Behavior diagrams: interaction

Balla: Dynamism of a Dog on a Leash, 1912
Modeling Interaction

- **Statechart diagram**
  - Depicts the flow of control inside an object using *states* and *transitions* (finite state machines)

- **Activity diagram**
  - Describes the *control flow* among objects by actions organized in workflows (Petri Nets)

- **Sequence diagram**
  - Depicts objects’ interaction by highlighting the *time ordering* of method invocations

- **Communication (collaboration) diagram**
  - Depicts the *message flows* among objects
Interaction diagrams

- A use case diagram presents an **outside view** of the system.
- The **inside behavioral view** of a system is shown by interaction diagrams.
- Interaction diagrams describe how use cases are realized in terms of interacting objects.
- Two types of interaction diagrams:
  - Sequence diagrams
  - Collaboration (Communication) diagrams
Sequence diagram

- A sequence diagram describes a sequence of method calls among objects.
- There are several types of method calls.
Sequence diagram

A SD highlights the objects involved in an activity

http://www.agilemodeling.com/artifacts/sequenceDiagram.htm
Sequence diagram: flow

Sequence of message sending
Using a SD for workflow

customer

| user | sponsor | business analyst | software architect | project manager | development manager | team leaders | developers | technical specialists | test designer | testers |

vendor

need

need

promise

order

feasibility study

gather requirements

analyse req.

design system

define timeline and milestones

create team

manage timeline

load and review

develop

test

release

acceptance

release

release
Consistency among diagrams

We can derive the dependencies shown in a class diagram from the interactions defined in a sequence diagram.
Exercise

Draw a sequence diagram showing how a customer interacts with a travel agency, a station and a train to reach some destination.

Draw a sequence diagram to show how a user prints a document on a printer, and a counter keeps a count of printed pages.
Communication (collaboration) diagram

• Communication diagrams show the message flow between objects in an application
• They also imply the basic associations (relationships) between classes
• Communication diagrams are drawn in the same way as sequence diagrams (and can be semantically equivalent to them)
1.1: ok := validate()
1.2 [ok]: deliver(c)
1.2.1 [not in stock]: back order(p)
1.2.2: get address()
Communication diagram

Collaboration

: Order
  1.1 : ok := validate()

: Payment
  1.2 [ok] : deliver(c)

: Customer
  1 : place an order(c)
  1.2.2 : get address()

: Product
  1.2.1 [not in stock] : back order(p)

: Supplier
Sequence and communication diagrams

- These two diagrams are equivalent
Exercise

• Draw a communication diagram showing how a customer interacts with a travel agency, a station and a train to reach some destination

• Draw a communication diagram to show how a user prints a document on a printer, and a counter keeps a count of printed pages
Basic diagrams we have seen

Diagram

- Structure Diagram
  - Class Diagram
    - Composite Structure Diagram
  - Component Diagram
  - Object Diagram
  - Deployment Diagram
  - Package Diagram

- Behavior Diagram
  - Activity Diagram
  - Use Case Diagram
  - State Machine Diagram
  - Interaction Diagram
    - Sequence Diagram
    - Interaction Overview Diagram
    - Communication Diagram
    - Timing Diagram
Other diagrams

Diagrams we have seen in this lecture:

• Use case, class, object, statechart, activity, interaction (sequence and collaboration)

We could add (using UML 1.*):

• Component, Deployment

We could add (using UML 2.*):

• Composite structure, Package, Interaction
  Overview, Timing
Usage survey

Survey on Usage of UML 1.0 Diagrams

- 91.63% for Class
- 76.01% for Sequence
- 73.43% for Use Case
- 47.50% for Activity
- 41.14% for Communication (2001, No Magic, Inc.)
- 35.37% for Object
- 35.91% for State
- 34.79% for Package
- 30.35% for Component
- 27.94% for Deployment
Main diagrams

The main diagrams that are used in most views are:

• Use case diagram
• Class diagram
• Sequence diagram
• Activity diagram
Discuss

• Which diagrams are most useful in each lifecycle phase?
Diagrams in lifecycle

Requirements -> Design -> Implementation

Use Case

- Class diagram
- Sequence diagram
- Activity diagrams and Statecharts
Conclusions

- UML is a notation still evolving, defined by a metamodel.
- It offers several diagram types, in order to describe different views on a model.
- Basic diagrams are: use cases, classes, behaviors (statechart+activity), interactions (sequence+communication).
- Several tools available.
- Needs a process to be used consistently and effectively.
Summary

• UML includes a number of diagram-based notations to model software systems using an object oriented approach
• UML is not a process (it needs a process, like for instance the RUP)
• It is not proprietary: it is an OMG (Object Management Group) and ISO standard
Exercise

• Draw, on some game-playing domain:
  – A class diagram
  – An object diagram
  – A statechart
  – A sequence diagram
  – A communication diagram
  – An activity diagram
Questions

• What is a software model?
• Which are the UML canonical diagrams?
• What is a use case?
• What is a class diagram?
• How do we describe a tree-like data structure in a class diagram?
• What is an interaction diagram?
• What is the difference between statecharts and activity diagrams?
Readings

• On use cases

• On class diagrams
  www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/index.html

• On sequence diagrams
UML Specification Documents

• OMG, UML Specification v. 1.5, 2003
• OMG, Meta Object Facility 2.0, 2006
• OMG, UML Superstructure 2.1.2, 2007
• OMG, UML Infrastructure 2.1.2, 2007
References on using UML

- Pilone and Pitman, *UML 2.0 in a Nutshell*, OReilly, 2005
Useful sites

• www.uml.org  **Documents defining the standard**
• www.omg.org
• www.agilemodeling.com/essays/umlDiagrams.htm
• softwarestencils.com/uml  **Images reusable in a graphic editor**
• www-306.ibm.com/software/awdtools/rmc/library
• www.cs.gordon.edu/courses/cs211/ATMExample
• opensource.objectsbydesign.com
• www.cragsystems.co.uk/ITMUML/index.htm  **Online courseware**
• www.eclipse.org/modeling/mdt/uml2/docs/articles/Getting Started with UML2/article.html
Tools

• Eclipse + several plugins, like Omondo
• www-01.ibm.com/software/rational/ Rational Rose
• jazz.net New IBM platform
• argouml.tigris.org Argo or Poseidon
• www.borland.com/us/products/together/index.html Borland Together
• www.visual-paradigm.com Visual Paradigm suite
• www.magicdraw.com/
• www.tabletuml.com
• abstratt.com/
• www.umlgraph.org
• code.google.com/p/raf-uml
• metauml.sourceforge.net Beautiful UML diagrams in LaTeX
Questions?