Génie logiciel
et gestion de projets

UML Overview
& UML Class Diagrams

Roel Wuyts
ULB
2005/2006

http://decomp.ulb.ac.be/education/GL0506/
Object-Oriented Modelling

Question: given some problem, how to develop an object-oriented system to address that problem?

✦ Object-Oriented Modelling to get there
✦ Give object-oriented solution to problem
✦ Most can be used with different processes (waterfall, incremental, ...)
✦ Will use UML diagrams

Note: other approaches could be used
✦ e.g. functional decomposition, ER diagrams, ...
In 1994, more than 50 OO methods!

- Meyer
  - Before and after conditions
- Harel
  - Statecharts
- Gamma, et al
  - Frameworks and patterns,
- Booch
  - Booch method
- Rumbaugh
  - OMT
- Jacobson
  - OOSE
- Wirfs-Brock
  - Responsibilities
- Embley
  - Singleton classes and high-level view
- HP Fusion
  - Operation descriptions and message numbering
- Shlaer - Mellor
  - Object lifecycles
- Odell
  - Classification
- Odell
  - Responsibilities
- Wirfs-Brock
  - Responsibilities
- Embley
  - Singleton classes and high-level view
- HP Fusion
  - Operation descriptions and message numbering
- Gamma, et al
  - Frameworks and patterns,
- Harel
  - Statecharts
- Booch
  - Booch method
- Rumbaugh
  - OMT
- Jacobson
  - OOSE
- Wirfs-Brock
  - Responsibilities
- Embley
  - Singleton classes and high-level view
- HP Fusion
  - Operation descriptions and message numbering
- Gamma, et al
  - Frameworks and patterns,
- Harel
  - Statecharts
- Booch
  - Booch method
- Rumbaugh
  - OMT
- Jacobson
  - OOSE
- Wirfs-Brock
  - Responsibilities
- Embley
  - Singleton classes and high-level view
- HP Fusion
  - Operation descriptions and message numbering
- Gamma, et al
  - Frameworks and patterns,
- Harel
  - Statecharts
- Booch
  - Booch method
- Rumbaugh
  - OMT
- Jacobson
  - OOSE
- Wirfs-Brock
  - Responsibilities
- Embley
  - Singleton classes and high-level view
- HP Fusion
  - Operation descriptions and message numbering
- Gamma, et al
  - Frameworks and patterns,
- Harel
  - Statecharts
- Booch
  - Booch method
- Rumbaugh
  - OMT
- Jacobson
  - OOSE
- Wirfs-Brock
  - Responsibilities
- Embley
  - Singleton classes and high-level view
- HP Fusion
  - Operation descriptions and message numbering
- Gamma, et al
  - Frameworks and patterns,
- Harel
  - Statecharts
- Booch
  - Booch method
- Rumbaugh
  - OMT
- Jacobson
  - OOSE
- Wirfs-Brock
  - Responsibilities
- Embley
  - Singleton classes and high-level view
- HP Fusion
  - Operation descriptions and message numbering
- Gamma, et al
  - Frameworks and patterns,
- Harel
  - Statecharts
- Booch
  - Booch method
- Rumbaugh
  - OMT
- Jacobson
  - OOSE
- Wirfs-Brock
  - Responsibilities
- Embley
  - Singleton classes and high-level view
- HP Fusion
  - Operation descriptions and message numbering
- Gamma, et al
  - Frameworks and patterns,
- Harel
  - Statecharts
- Booch
  - Booch method
- Rumbaugh
  - OMT
- Jacobson
  - OOSE
- Wirfs-Brock
  - Responsibilities
- Embley
  - Singleton classes and high-level view
- HP Fusion
  - Operation descriptions and message numbering
- Gamma, et al
  - Frameworks and patterns,
- Harel
  - Statecharts
- Booch
  - Booch method
- Rumbaugh
  - OMT
- Jacobson
  - OOSE
- Wirfs-Brock
  - Responsibilities
- Embley
  - Singleton classes and high-level view
- HP Fusion
  - Operation descriptions and message numbering
- Gamma, et al
  - Frameworks and patterns,
Designed by Booch, Rumbaugh, Jacobson (3 amigos)

- Started in 1994; version 1.0 finished in 1997
- Version 1.5 (1.4.2) since July 2004: current
- Version 2.0 in beta since 2004 - final late 2005

To end the OO method wars: standard

Standard adopted by OMG (also known from Corba)
General Goals of UML

Model systems using OO concepts

Establish an explicit coupling to conceptual as well as executable artifacts

To create a modeling language usable by both humans and machines

Models different types of systems (information systems, technical systems, embedded systems, real-time systems, distributed systems, system software, business systems, UML itself, ...)

UML Worldview

Two concepts:

✧ Views

✧ Diagrams
Each view is a projection of the complete system.

Each view highlights particular aspects of the system.

Views are described by a number of diagrams.

No strict separation, so a diagram can be part of more than one view.
UML Views…

Logical View

Component View

Use Case View

Deployment View

Concurrency View
UML Diagrams...

- Use-Case diagram (see later)
- Class diagram (see later)
- Object diagram
- State diagram
- Sequence diagram
- Collaboration diagram
- Activity diagram
- Component diagram
- Deployment diagram
State Diagram

Represent the behaviour of a class in terms of (evolution of) its state.
Sequence Diagram

Temporal representation of objects and their interactions.

Diagram showing interactions between objects: Computer, PrinterServer, Printer, and Queue.
Collaboration Diagram

Spatial representation of objects, relations and interactions.

Diagram:

- **Computer**
  - 1: Print(file)

- **PrinterServer**
  - [printer free] 1.1: Print(file)
  - [printer busy] 1.2: Store(file)

- **Queue**
  - [printer free] 1.1: Print(file)

- **Printer**
Object Diagram

Represents objects and their relations; corresponds to simplified collaboration diagrams (no message sends).

![Object Diagram Image]

- **Author**:
  - name: String
  - age: Integer

- **Computer**:
  - name: String
  - memory: Integer

- **Bob's Job PC**:
  - name = "Dell 466"
  - memory = 64

- **Bob's Home PC**:
  - name = "Compaq Pentium MMX"
  - memory = 32

- **Class Diagram**

- **Object Diagram**
Activity Diagram

Represent the behaviour of one operation in terms of actions.
Component Diagram

Represent the physical components of a system.
Deployment Diagram

Represent the deployment of a system on hardware.
Some mental exercise:

- Semantic of UML diagrams is described in UML!
  - Each diagram is an instance of a UML meta model
  - In other words: each diagram shows a simplified view of the meta model
  - UML Meta Model describes all possibilities
Common UML: Elements

Core : Element

- used for both model elements as visual elements

  - model element: represents system abstraction while modelling

  - visual element: textual or graphical representation of a model element that allows a user to interact
Meta-model part: Element

- Packet
  - Model
    - Element
      - Model Element
      - Visual Element

.projects

contains 0..1
references 0..*
Across UML, a number of common mechanisms are used:

- Stereotypes
- Tagged Values
- Notes
- Constraints
- Dependency relationships
- (type, instance) and (type, class) dichotomies
Meta model part: common mechanisms
Stereotypes

- Allows to define a new kind of model element based on an existing one
- Basically adds extra semantics
- There are predefined stereotypes
Tagged Values

Tagged values

✦ Name-value pairs of information
  ▸ (name, value)

✦ Hold additional information about elements
Comment attached to one or more elements

Holds no semantic information, just information

✧ use stereotypes for semantic information

Graphically:
Constraints

Restrictions that limit the usage of an element or the semantics of an element

No syntax specified

- Can be natural text, pseudo-code, mathematical expressions, OCL (object constraint language), ...
Dependency Relationships

One-directional usage relationship between two model elements (called source and target)

Notes or constraints are valid sources for dependency relationships

Graphical notation:

```
Client  Provider
```

(client depends on provider)
Common UML: Primitive Types

Primitive types:

- *Boolean*: enumerated type {true, false}
- *Expression*: string with some semantics
- *List*: ordered sequence, possible indexed
- *Multiplicity*: see next
- *Name*: string used to indicate element
- *Point*: tuple (x,y,z) that indicates point in space
- *String*: list of characters designated with a name
- *Time*: represents absolute or relative time
- *Non-interpreted*: blob
Primitive type: multiplicity

Non-empty set of positive integers

Syntax:

\[
\text{multiplicity} ::= [\text{interval} | \text{number}] \{\text{‘,‘ multiplicity}\}
\]
\[
\text{interval} ::= \text{number} “..” \text{number}
\]
\[
\text{number} ::= \text{positive\_number} | \text{name} | “*”
\]

Examples:

1
1, 3..4, 5..8
0..*
Common UML: Packet

Groups model elements

- All elements belong to a package
  - There is a root package for the system
- Can contain other packets, since packets are model elements themselves

Enforces namespace

- Two elements in a different package can have the same name
Packet

Graphical notation

```
<table>
<thead>
<tr>
<th>Node</th>
<th>Packet</th>
</tr>
</thead>
<tbody>
<tr>
<td>String name</td>
<td>contents</td>
</tr>
<tr>
<td>next : Node</td>
<td>addressee : Node</td>
</tr>
<tr>
<td>accept(p : Packet)</td>
<td>isReceiverFor(n : Node)</td>
</tr>
<tr>
<td>send(p : Packet)</td>
<td></td>
</tr>
</tbody>
</table>
```
Package import

Elements can be shared between packages

The import relation between packages is modelled using a dependency relationship stereotyped with import
Package visibility

Elements contained in a package are not visible to the outside world

- except when they are declared *public* (using stereotypes, for example)
- otherwise they are *implementation* (not visible)
Class Diagrams

Static model type

✦ A view of the system in terms of classes and relationships

Classes not only describe attributes but also behaviour!

Description of object types.

✦ Attributes and behaviour of a type of objects

✦ All objects are instances of a certain class
A Class in UML…

A rectangle divided into 3 compartments:

- name, attributes, operations

<table>
<thead>
<tr>
<th>Name</th>
<th>Uppercase, bold</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes</td>
<td>lowercase</td>
</tr>
<tr>
<td>operations</td>
<td>+, -, # for visibility</td>
</tr>
<tr>
<td></td>
<td>= for defaults</td>
</tr>
<tr>
<td></td>
<td>{ } for enumerations</td>
</tr>
<tr>
<td></td>
<td>underline : static</td>
</tr>
</tbody>
</table>
Some examples

**Invoice**

+ amount : Real  
+ date : Date = Current date  
+ customer : String  
- administrator : String = "Unspecified"  
- number of invoices : Integer  
+ status : Status = unpaid {unpaid, paid}

**Figure**

# size : Size  
# pos : Position  
+ draw()  
+ scaleFigure(percent : Integer = 25)  
+ returnPos() : Position
Relationships Between Classes

An association is a connection between classes

- “usage”

A generalization is a relationship between a more general and a more specific element

- “inheritance”

A refinement is a relationship between two descriptions of the same thing but at different levels of abstraction

A realization is a relationship between elements where one carries out what the other specifies
Associations

Specify structural relationships

Specifies that objects are interconnected

Can be implemented in a lot of ways

- through instance variables
- through arguments of methods
- through auxiliary classes
- ...

Association Relationship

Drawn as line between classes

By default bidirectional

✦ but particular direction can be indicated

Can contain multiplicities

✦ a range that tells us how many objects are linked

9 examples to show different aspects and give some additional information...
Model the following:

- An author can use a computer.
Example 1 Solution

```
Author
uses
association
name

association
direction

Computer

association
```
On association directions...
Example 2

Model the following:

- A user can own 0 or more cars. Each car is owned by 1 or more persons
Solution Example 2

A Person Owns 0 or many Cars
A Car is Owned by 1 or many Persons
Common Multiplicities

<table>
<thead>
<tr>
<th>Multiplicity</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>optional</td>
<td>0..1</td>
</tr>
<tr>
<td>zero or more</td>
<td>0..* or *</td>
</tr>
<tr>
<td>at least one</td>
<td>1..*</td>
</tr>
<tr>
<td>exactly one</td>
<td>1 or left blank</td>
</tr>
</tbody>
</table>

(See primitive types for all possibilities)
Example 3

Model the following:

✧ Nodes can be connected to other nodes.
Recursive Associations

Recursive association: connecting a class to itself

[Diagram of a recursive association with a class labeled 'Node' and an association labeled 'Connects']
Example 4

Model the following:

- A Canvas contains many Figures which are identified by an identifier
Qualified Associations

Specifies how a certain object at the many end is identified (+/- a key)

A Canvas contains many Figures which are identified by a figure id
Qualified associations as restrictions

Qualified association subdivides the referenced set of objects into partitions where, viewed from the initial object, each partition may occur only once.

Example: employees are partitioned: all employees with the same initials belong to one partition.

<table>
<thead>
<tr>
<th>company</th>
<th>initials</th>
<th>employee</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULB</td>
<td>RW</td>
<td>Roel Wuyts</td>
</tr>
<tr>
<td>ULB</td>
<td>TM</td>
<td>Thierry Massart</td>
</tr>
<tr>
<td>UMH</td>
<td>TM</td>
<td>Tom Mens</td>
</tr>
</tbody>
</table>
Example 5

Model the following:

- An insurance company has insurance contracts, which refer to one or more customers.
  - A customer has insurance contracts (zero or more), which refer to one insurance company.
  - An insurance contract is between an insurance company and one or more customers. The insurance contract refers to both a customer (or customers) and an insurance company.
  - The insurance contract is expressed in an (zero or one) insurance policy (a written contract of insurance). The insurance policy refers to the insurance contract.
Example 5 solution

```
1..* has ref to Customer 0..1
is expressed in

expresses an
1..* has ref to

refers to
```

Diagram:

- **Insurance Company**
- **Insurance Policy**
- **Insurance Contract**
- **Customer**
Example 6

Model the following:

- An insurance contract belongs to a customer or to a company. A customer can have multiple insurances. An insurance contract has an ordered collection of 1 or more customers.
Example 6: ‘or’ associations

Put constraint between the associations

Insurance Company

<table>
<thead>
<tr>
<th>has</th>
</tr>
</thead>
<tbody>
<tr>
<td>1..*</td>
</tr>
<tr>
<td>refers to</td>
</tr>
</tbody>
</table>

Insurance Contract

| ▲ has |
| * |

Customer

| { ordered } |
| refers to |

| { or } |
| 1..* |
Other constraints

Some widely used constraints:

✧ `{ordereded}`, `{or}`, `{xor}`

✧ `{implicit}`: specifies that the relationship is not manifest but, rather, is only conceptual

✧ `{changeable}`: Links between objects may be added, removed, and changed freely

✧ `{addonly}`: New links may be added from an object on the opposite end of the association

✧ `{frozen}`: A link, once added from an object on the opposite end of the association, may not be modified or deleted
Example 7

Model the following:

✦ An elevator control manipulates four elevators. On each link between the elevators and the elevator control, there is a queue. Each queue stores the requests from both the elevator control and the elevator itself. When the elevator control chooses an elevator to perform a request from a passenger outside the elevator, the elevator control reads each queue and chooses the elevator that has the shortest queue. The choice could also be made using some clever algorithm.
Example 8: association classes

Class Queue is needed for the association (can have state/methods needed to implement the algorithm)
Example 8

A reservation on a passenger train consists of a passenger (for whom the seat is reserved), a seat (which is being reserved), and a train (time of reservation). Besides such simple reservations, group reservations are also allowed. Who will actually occupy which seat is left over to the group.
Example 9: ternary associations

ternary association

<table>
<thead>
<tr>
<th>Train</th>
<th>Reservation</th>
<th>Seat</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td></td>
<td>CarriageNumber</td>
</tr>
<tr>
<td>number</td>
<td></td>
<td>Number</td>
</tr>
<tr>
<td></td>
<td>1..*</td>
<td>1..*</td>
</tr>
<tr>
<td>Passenger</td>
<td></td>
<td></td>
</tr>
<tr>
<td>name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>title</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1..*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
n-ary associations

Associations in general relate n classes

✦ Most of them are binary, but three (as in previous example) or more are possible

Can usually be transformed in binary associations:

A whole-part association (whole owns the part)

Describes different levels of abstraction

Note: Diamond can only be at one end!
The parts can only exist if the whole exists

✧ they are destroyed with the whole
Composition Aggregation…

Other notations expressing the same:

```
     Window
        Contains
          * Text
          * Listbox
          * Button
          * Menu

     Window
          * Text
          * Listbox
          * Button
          * Menu
```
A shared aggregation is one in which the parts may be parts in any wholes, as shown in Figure 18. That an aggregation is shared is shown by its multiplicity. The aggregation is shared if the multiplicity on the wholeside is other than one (1). Shared aggregation is a special case of a normal aggregation.

### Diagram

**Team**

* \(*

**Person**

* \(*

Members

**Remix**

* \(*

{ordered}

**Sound clips**

* \(*

Contains

* : shared aggregation
Association or Aggregation?

The decision to use association or aggregation is a matter of judgment and is often arbitrary.

E.g. What type of relationship should be used to model a car with its tires?

- If the application is a service center, the only reason you care about the tire is because it is part of the car, so use an aggregation.

- If the application is a tire store, you will care about the tire independent of the car, then the relationship should be an association.
Recap: associations

Connect classes

- between 1, 2, 3, ... classes

Can contain multiplicities, directions, role names

Constraints can be used to express additional requirements (more on constraints later)

Association classes can be used for more difficult associations that have behaviour from their own

Aggregation: part-whole composition
Inheritance, ‘is-a’ relationship

The more specific may be used where the more general is allowed

Remember: Using Inheritance for code reuse, or just because it looks nice is a dangerous practice!
Refinement

“A relationship that represents a fuller specification of something that has already been specified at a certain level of detail.”

Useful for modelling

- a relation between the analysis version and the design version, or between a clean implementation and an optimized but potentially difficult variation

\[A \xleftrightarrow{\text{refine}} B\]  
B refines A
Class refinement examples

Segment
length: Number
stretch(k : Integer)

Segment'
xInitial: Number
xFinal: Number
stretch(k : Integer)

length = xFinal - xInitial

Person
name: String
address: Address
livesIn(location: String) : Boolean
firstName: String
lastName: String
printOn(Stream s)
“A semantic relationship between classifiers in which one classifier specifies a contract that another classifier guarantees to carry out”
Implementing realization

In Java: implemented with interfaces

```
<<interface>>
IRuleAgent

addRule()
changeRule()
explainAction()
```

In C++: implemented with fully abstract classes

```
IRuleAgent

addRule()
changeRule()
explainAction()
```

AccountBusinessRules
In older versions of UML, the notation of refinement was the notation which is now used for realization.
Wrap-up

Class Diagrams model statical elements (classes and their relationships)

For use by architects and developers

Relations between classes:

- association
- aggregation
- composition aggregation
- refinement
- realization
- inheritance
UML models all kinds of systems.

It is a modelling language, not a process!

It consists of

✦ Views (5)
✦ Diagrams (9)

It is based on a meta model and hence the different diagrams share similar elements

✦ and they are extensible