Unit 2. Modeling Objects and Components with UML
Objectives

• To describe the activities in the object-oriented analysis and design process
• To introduce various models that can be used to describe an object-oriented analysis and design
• To show how the Unified Modelling Language (UML) may be used to represent these models
• To introduce models suitable for specifying Components-Based Software
Roughly ...

Requirements Elicitation

Requirements Specification

Analysis and Design

Go ahead

They could be using UML :-)
You are here!

Fundamentals of Software Engineering, R. Bahsoon
Several different notations for describing object-oriented designs were proposed in the 1980s and 1990s.

The Unified Modeling Language is an integration of these notations.

It describes notations for a number of different models that may be produced during OO analysis and design.

It is now *a de facto* standard for OO modelling.
UML Contributors

- http://www.uml.org/

- Rumbaugh OMT
- Jacobson OOSE
- Booch
- Meyer
- Shlaer-Mellor
- Harel
- Gamma et al
- Framework and pattern
- And many others

Major three (submission to OMG Jan 97, Acceptance Nov 97...)
http://www.omg.org/
“The three amigos”

April 1999, following OMG feedback
OMG Acceptance, Nov 1997
UML partners experience
Web: June '96
Serial '96
OOPSLA '95
Unified Method 0.8
Other methods
Booch Method
Rumbaugh's OMT
Jacobson's OOSE

UML 0.9 & 0.91
UML 1.0, 1.1
UML 1.4
UML 2.0

Selic's ROOM
UML Diagrams

- Sequence Diagrams
- Collaboration Diagrams
- State Diagrams
- Activity Diagrams
- Use Case Diagrams
- Class Diagrams
- Object Diagrams
- Component Diagrams
- Deployment Diagrams

Taken from [Booch 1999]
Models?

- The language of the designer
- ** Representations ** of the system to-be-built or as-built
- A complete description of a system from a particular perspective
- Vehicles for communication with various stakeholders
- Allow reasoning about some characteristics of a system
- Often captures both ** structural and behavioural ** (e.g., interaction) information
UML Diagrams

• Diagram: a view into the model
• In UML, there are nine standard diagrams
  - Static view: use case, class, object, component, deployment
  - Dynamic view: sequence, collaboration, state chart, activity
Some UML diagrams

- Use cases
- Class diagram
- Deployment
- Activity
- Sequence
- Collaboration
UML Diagrams

You are Here!

Sequence Diagrams
Use Case Diagrams
Class Diagrams
Object Diagrams
Component Diagrams
Deployment Diagrams
Collaboration Diagrams
State Diagrams
Activity Diagrams

Taken from [Booch 1999]
Use Cases

- What is use case modelling?
- What are actors?
- How to find actors?
- What are use cases?
- How to find use cases?
- How to construct a use case diagram?
- Detailing a use case...
What is a use case modelling

- Basis for a user-oriented approach to system development
  - Identify the users of the system (actors)
  - Identify the tasks they must undertake with the system (use cases)
  - Relate users & tasks (relationship)… help identify boundary

Capture system functionality as seen by users
Use cases

Built in early stages of development

- Specify the context of a system
- Plan iterations of development
- Validate a system’s architecture
- Drive implementation & generate test cases
- Developed by analysts & domain experts during requirements analysis
How to find actors?

• Observe direct users of the system—could be users or systems
  - What role do they play?
  - Who provides information to the system?
  - Who receives information from the system?

• Actors could be:
  - Principal
  - Secondary (External hardware, other systems, …)

• Describe each actor clearly and precisely (semantics)
  - Short name
  - Description

BookBorrower
This actor represents some one that make use of the library for borrowing books
Exercise

• Assume you have a requirements documents for a library system: identify all actors that interact with a system
• For each actor, write down the name and provide a brief textual description (i.e., describing the semantics of the actor)

<table>
<thead>
<tr>
<th>Actor</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name 1</td>
<td>Description</td>
</tr>
</tbody>
</table>
What are use cases?

• Things actors do with the system
  - A task which an actor needs to perform with the help of a system (e.g., Borrow a book)
  - A specific kind of a system

• Describe the behaviour of the system from a user’s standpoint

• Represented by ellipses
How to find use cases?

• Start with the list of actors and consider
  - What they need from the system (i.e. what use cases there are which have value for them)
  - Any other interactions they expect to interact with the system (i.e. which use cases they might take part in for someone’s else benefit)

• How do you know what is a use case?
  - Estimate frequency of use, examine differences between use cases, distinguish between “basic” and “alternative” course of events & create new uses when necessary
Describing use cases

Semantics detailed in text

Example:
Borrow copy of book

A book borrower presents a book. The system checks that the potential borrower is a member of the library & she does not have the maximum number of books.
Exercise

• Draft use case diagrams of a library system
Possible use cases...

- BookBorrower
  - Reserve book
  - Borrow copy of book
  - Return copy of book
  - Extend loan
  - Borrow journal
  - Return journal

- JournalBorrower
  - Borrow journal
  - Return journal

- Browser
  - Browse

- Librarian
  - Update catalog
Use case diagram of a library
Requirements example

Multi-purpose recycling machine must:
- receive & check items for customers,
- print out receipt for items received,
- print total received items for operator,
- change system information,
- signal alarm when problems arise.

Reference: Anthony Finkelstein, UCL
Example

Returning items is started by Customer when she wants to return cans, bottles or crates. With each item that the Customer places in the recycling machine, the system will increase the received number of items from Customer as well as the daily total of this particular type.

When Customer has deposited all her items, she will press a receipt button to get a receipt on which returned items have been printed, as well as the total return sum.

Particular instances of use would be different... The morning after the party Sarah goes to the recycling centre with three crates containing ....
Use case diagram

- Customer
  - Recycling Machine
    - Returning item
    - Generate report
    - Change item information
  - Operator
Extensions

- *Extensions* provide opportunities for:
  - *optional* parts
  - *alternative* complex cases
  - *separate* sub-cases
  - *insertion* of use cases
Refinement - <<extend>>

Note: the direction of the arrow from the less central case to the central one! Refuse loan and borrow copy of a book two different scenarios

<<extend>>
Refinement - «extend>>
Refinement

Extend Loan

Borrow copy of a book

Check for reservation

<<include>>

<<include>>
Use <<include>>

- How the system can reuse pre-existing component
- To show common functionality between use cases
- To develop the fact that project from existing components!

Note: <<include>> and <<extend>> are UML stereotypes used to attach additional classification
Developer

create model

run a model

observe behavior

collect statistics

Experimenter

<<include>>

<<extend>>

<<include>>
Refinement

Abstract use case

Concrete use case

Abstract actors

Concrete actors

Print

<<uses>>
Returning Item

<<uses>>
Daily Report

Receipt Receiver

inherits
Customer

inherits
Operator
Generalization

Journal borrower is a book borrower
Detailing a use case

• Writing a specification for the use case

• Good Practice
  - Preconditions: the system state before the case begin (i.e., facts, things that must be true)
  - Flow of events; the steps in the use case (i.e. actions...)
  - Postconditions: the system state after the case has been completed
Detailing a use case

Borrow a copy of book

• Precondition
  1. the BookBorrower is a member of the library
  2. the BookBorrower has not got more than the permitted number of books on loan

• Flow of events
  1. the use case starts when the BookBorrower attempts to borrow a book
  2. the librarian checks it is ok to borrow a book
  3. If …… (indicate an alternative path of action)

• Post-conditions
  1. the system has updated the number of books the BookBorrower has on loan
Exercise

• Select one of the use cases identified for the library system and create complete specification of each
• Use Structured English to show at least one alternative flow of events and at least one repeated action

Borrow copy of book

Preconditions
1.

Flow of events
1.
2.

Post conditions
1.
Scenarios

• Each time an actor interacts with a system, the triggered use cases instantiate a scenario
• Each case corresponds to a specific path through a use case with no branching
• Scenarios are typically documented as text along side the use case and activity diagrams
Write the scenarios for this diagram

- **Borrow copy of book**
  - **extension points**
  - status validation:
    - after confirming identity
  - <<extend>>
  - too many books on loan

- **Refuse loan**
Example- borrow copy of book

• Scenario 1

BookBorrower Joe B Borrows the library's only copy of using UML, when he has no other book on loan. The system is updated accordingly.

• Scenario 2

BookBorrower Ann tries to borrow the library's second copy of Software Engineering, but is refused because she has six books out on loan, which is her maximum allowance.
UML Diagrams

Covered

You are Here!
Activity Diagrams

• Activity diagrams show the dependencies and coordination between activities within a system
  - The activity flow should not get “stuck”
  - They can be used during the requirements elicitation process ...
  - help in identifying use cases of a system and operations involved in the realization of a use case

• Workflows and business processes
• Can be attached to any model element to model its dynamic behavior
Activity Diagrams

Reference: David Rosenblum, UCL
Swimlanes(i.e., main actors swimming on each lane)
UML Diagrams

Covered

You are Here!

Covered
Class: Simple Example

Class Name
Attributes
Operations

Customer
name : String
address : String
creditRating()
UML Class Icons

Optional Visibility Adornments
(+public, -private, #protected, ~package)

Name compartment

Optional attribute compartment

Optional operation compartment

Class name
(in italics if abstract)

Attribute name

Attribute type

Initial value

Operation with class scope and stereotype

Operation signatures

#title : String
#available : Boolean = true

<<constructor>>
+create()

+copiesOnShelf() : Int
+borrow(c:Copy)

Reference: D. Rosenblum, UCL
+, #, -

- + means public: public members can be accessed by any client of the class
- # means protected: protected members can be accessed by members of the class or any subclass
- - means private: private members can only be accessed by members of the same class
Analysis class

An analysis class abstracts one or more classes and/or subsystems in the system’s design

- Focuses on handling functional requirements
- Defines responsibilities (cohesive subsets of behaviour defined by the class)
- Defines attributes
- Expresses relationships the class is involved in
Approach 1: Data-Driven Design

Identify all the *data* in the system

- Divide into classes *before* considering responsibilities
- Common approach: *noun identification*
  - Identify *candidate classes by selecting all the nouns* and noun phrases in the requirements document
  - Discard inappropriate candidates
    » Redundant or omnipotent entities
    » Vague entities
    » Events or operations
    » Meta-language
    » Entities outside system scope
    » Attributes
  - Verbs and verb phrases *highlight candidate operations!*
Approach 1: Data-Driven Design

Some heuristics of what kind of things are classes [Shlaer and Mellor; Booch]...

- Tangible or “real-world” things - book, copy, course;
- Roles- library member, student, director of studies,
- Events- arrival, leaving, request;
- Interactions- meeting, intersection
Exercise

- Perform noun-verb analysis of your requirements document;
- Underline all the noun and noun phrases,
- Create a list of candidate classes (in examining the discard criteria, you may also identify some candidate attributes)
- Identify all verb and verb phrases
- Create a list of candidate operations and assign them to classes
Noun/Verb Analysis

Books and journals The library contains books and journals. It may have several copies of a given book. Some of the books are for short term loans only. All other books may be borrowed by any library member for three weeks. Members of the library can normally borrow up to six items at a time, but members of staff may borrow up to 12 items at one time. Only members of staff may borrow journals.

Borrowing The system must keep track of when books and journals are borrowed and returned, enforcing the rules described above.
Approach 2: Responsibility-Driven Design

- Identify all the responsibilities in the system
- Divide into classes before considering the classes' data
- Common approach: CRC cards
  - Class, Responsibilities, Collaborations
Example CRC Cards for a Library

<table>
<thead>
<tr>
<th>LibraryMember</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities</td>
<td></td>
</tr>
<tr>
<td>Maintain data about copies currently borrowed</td>
<td></td>
</tr>
<tr>
<td>Meet requests to borrow &amp; return copies</td>
<td>Copy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Copy</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities</td>
<td></td>
</tr>
<tr>
<td>Maintain data about particular copy of book</td>
<td>Book</td>
</tr>
<tr>
<td>Inform corresponding Book when borrowed/returned</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Book</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities</td>
<td></td>
</tr>
<tr>
<td>Maintain data about one book</td>
<td></td>
</tr>
<tr>
<td>Know whether there are borrowable copies</td>
<td></td>
</tr>
</tbody>
</table>
Exercise

• Perform responsibility-driven analysis for the system to identify potential classes:
  - Look at the requirements document and use cases
  - Identify the candidate classes
• Derive your CRC (i.e., Class, Responsibility, and collaborators)
First-Cut Class Diagram

- **Book**: 1
  - is a copy of
    - **Copy**: 1..*
      - **LibraryMember**: 0..1
        - **borrows/returns**: 0..1
      - **MemberOfStaff**: 0..1
        - **borrows/returns**: 0..1
      - **Journal**: 0..*
        - **borrows/returns**: 0..*

Relationships

- **Relationships** are connections between modeling elements
- Improve understanding of the domain, describing how objects work together
- Act as a sanity check for good modeling
- **Associations** are relationships between *classes*
  
  Examples
  » Object of class A sends a message to object of class B
  » Object of class A creates an object of class B
  » Object of class A has attribute whose values are objects of class B
  » Object of class A receives a message with argument of class B
- **Links** are relationships between *objects*
  - Links can be *instances* of associations (as in UML 1.4)
  - Allow one object to invoke operations on another object
UML Relationship Notation

- bidirectional / binary
- unidirectional
- aggregation
- composition

Reference: D. Rosenblum, UCL
Links Instantiate Associations

Reference: D. Rosenblum, UCL
Multiplicity of an Association

- Indicates the number of objects that can participate in a relationship at any point in time

Reference: D. Rosenblum, UCL
Generalisation and Inheritance

- A special kind of association
- Subclass *inherits* attributes and operations of superclass
  - And possibly extends superclass

```
LibraryPatron  Copy
  borrows/returns
    0..1     0..*

Researcher  Journal
  borrows/returns
    0..1     0..*
```

Book

is copy of

1

1..*

Another Inheritance Example

Staff Member
- salary: Int
- increaseSalary(Int)

Librarian
- assignSubject(String)

Tutor
- assignCourse(String)

Researcher
- beginProject(String)

Professor
- salary: Int
- increaseSalary(Int)
- assignCourse(String)
- beginProject(String)

Multiple inheritance
Part/Whole Associations

- **Aggregation:** Weak Ownership
  - The part objects can feature simultaneously in any number of other whole objects

```
Course 1..* Module 5..*
```

A module is part of a course

In fact,

5 or more modules are part of one or more courses
Part/Whole Associations

- Composition: Strong Ownership
  - The whole strongly owns its parts, so the parts cannot feature elsewhere

\[\text{CheckerBoard} \rightarrow 1 \text{ Square} \]

- NOTE: Not all 1-to-* relationships imply ownership
Association Classes

- Used to attach attributes to an association itself rather than the classes it associates
- Class association line must have the same name!
Example: Class Model

- Lecturer
- Module
- DirectorOfStudies
- Student
- HonoursCourse
- NonGraduatingStudent
- GraduatingStudent

Relationships:
- Lecturer teaches Module
- Module takes 6 Student
- DirectorOfStudies directs Lecturer
- Student directs Module
- HonoursCourse is on 1 Student
- Student is on 0..* GraduatingStudent
Another Example: Class Model
Example: Example Class Diagram
More Examples

Order
- dateReceived : Date
- isPrepaid : Boolean
- number : String
- price : Money
- dispatch()
- close()

Association

Customer
- name : String
- address : String
- creditRating()

Multiplicity
- Many-valued
- Mandatory

n
1

Fundamentals of Software Engineering, R. Bahsoon
More Examples

Diagram showing a UML class diagram with a generalization relationship between Customer, Corporate Customer, and Personal Customer classes. The Customer class has attributes for name and address, and a method for credit rating. The Corporate Customer class has additional attributes for contact name, credit rating, credit limit, and methods for remind and bill for month. The Personal Customer class has an attribute for credit card number.
More Examples

Classes Corporate Customer and Personal Customer have some similarities such as name and address, but each class has some of its own attributes and operations.

The class Customer is a general form of both the Corporate Customer and Personal Customer classes.
What Makes a ‘Good’ Analysis Class..

- Its name reflects its intent
- It is a crisp abstraction that models one specific element of the problem domain
- It has a small but defined set of responsibilities
- It has **high cohesion**
- It has **low coupling** with other classes

*homework: important!*

- **What is cohesion?**
- **What is coupling?**
Note...

• Noun/verb analysis and Responsibility-Driven analysis
  - Noun/Verb and responsibility complement each others
  - Often goes hand in hand with use cases
• First-cut class diagram (also referred to Class model)
• Refine the first-cut diagram into a detailed class diagram
Hint...
Environment: Demo

• Examples
  - Rational Rose sample
  - http://www.developers.net/external/249
UML Object Icons

- **Object name**: DSRsUMLBook : Book
- **Class name**: Book
- **Attribute name**: title = “Using UML”
- **Attribute value**

Operations and attribute types are *not* shown on object diagrams!

Reference: D. Rosenblum, UCL
Object Diagram

- Capture *class instances* and the *links* between them

![Object Diagram]

*Taken from [Booch 1999]*
Object Diagram

• Built during analysis & design
  - Illustrate data/object *structures*
  - Specify *snapshots*
• Developed by analysts, designers and implementers
Object Diagram

- John Smith: Student
- Sally Jones: Student
- Sarah McGrath: Student
- Scott Ambler: Student
- Ed Maloney: Student

- CSC 100a: Seminar
  - Term: "Fall"

- CSC 100b: Seminar
  - Term: "Spring"

- Course
  - Name: "Intro to Computer Science"
More Examples...
UML Diagrams

You are Here!

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered
Sequence diagrams

- Sequence diagrams demonstrate the behavior of objects in a use case by describing the objects and the messages they pass. The diagrams are read left to right and descending.

- Object interactions arranged in a time sequence (i.e., time-oriented)
Sequence diagrams

objects

Activation: i.e., object in active

message

destroy

return

Life-line
Sequence diagrams

- The example shows an object of class 1 start the behavior by sending a message to an object of class 2. Messages pass between the different objects until the object of class 1 receives the final message.
Sequence diagrams

- The example shows an object of class 1 start the behavior by sending a message to an object of class 2. Messages pass between the different objects until the object of class 1 receives the final message.
Sequence diagrams

- The example shows an object of class 1 start the behavior by sending a message to an object of class 2. Messages pass between the different objects until the object of class 1 receives the final message.
Sequence diagrams

- The example shows an object of class 1 start the behavior by sending a message to an object of class 2. Messages pass between the different objects until the object of class 1 receives the final message.
Example

- Self-service machine, three objects do the work we're concerned with
  - the front: the interface the self-service machine presents to the customer
  - the money register: part of the machine where monies are collected
  - the dispenser: which delivers the selected product to the customer
Example

- The instance sequence diagram may be sketched by using this sequences:
  - 1. The customer inserts money in the money slot
  - 2. The customer makes a selection
  - 3. The money travels to the register
  - 4. The register checks to see whether the selected product is in the dispenser
  - 5. The register updates its cash reserve
  - 6. The register has a dispenser deliver the product to the front of the machine
Example

The "Buy a product" scenario. Because this is the best-case scenario, it's an instance sequence diagram.
... But

• We have seen an instance of an interaction diagram— one possible sequence of messages
• Since a use case can include many scenarios
  – There is a need to show conditional behaviour
  – There is a need to show possible iterations
• A generic interaction diagram shows all possible sequences of messages that can occur
Showing conditional behavior

• A message may be **guarded** by a condition
• Messages are only sent if the guard evaluates to true at the time when the system reaches that point in the interaction

```
If i=0 then foo() Else bar()
```

```
If i=0 then foo() If i=1 then bar()
```
alt: Operators in interactions frames – UML 2.0

Alternative multiple fragment: only the one whose condition is true will execute
**Iterations (i.e., loop) - UML 1.0**

- * Indicates looping or iterations
- i:=1..2 means 2 iterations....

Result: ab  ab

If you have seen it?

Earlier UML versions: UML 1.0
Loop in UML 2.0

Loop: the fragment may execute multiple times, and the guard indicates basis for iterations.
Opt: Optional; the fragment executes only if the supplied condition is true. This is equivalent to an alt with one trace.
Sequence diagram of library

```
1: okToBorrow

2: borrow

2.1: borrowed
```
Showing timing constraints on a sequence diagram

```
A

{C - A < 5 sec}

1: okToBorrow

2: borrow

{borrowed' - borrowed < 1 sec}

2.1: borrowed

time
```
## Interaction types in sequence diagrams

<table>
<thead>
<tr>
<th>Interaction type</th>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous or call</td>
<td>➞</td>
<td>The ‘normal’ procedural situation. The sender loses control until the receiver finishes handling the message, then gets control back, which can optionally be shown as a return arrow.</td>
</tr>
<tr>
<td>Return</td>
<td>←</td>
<td>Not a message, but a return from an earlier message. Unblocks a synchronous send.</td>
</tr>
<tr>
<td>Flat</td>
<td>➞</td>
<td>The message doesn’t expect a reply; control passes from the sender to the receiver, so the next message (in this thread) will be sent by the receiver of this message.</td>
</tr>
<tr>
<td>Asynchronous</td>
<td>➞</td>
<td>The message doesn’t expect a reply, but unlike the flat case, the sender stays active and may send further messages.</td>
</tr>
</tbody>
</table>

Some UML versions use for both
Example

Asynchronous return

Student submitting a choice to the web

An e-mail sent to the system

Return

Asynchronous
Other notions: Branching

The life time of any object which could be affected by a conditional message is split into branches.
Opt: Optional; the fragment executes only if the supplied condition is true. This is equivalent to an alt with one trace.
Examples

• Refer to examples and printouts on sequence diagrams for optional extra features
Exercise

- Draft use case diagram for an ATM machine
- Use a Scenario of Interest
- Draw a simplified object diagram corresponding to the use cases
- Draft the corresponding sequence diagram
UML Diagrams

- Sequence Diagrams
- Collaboration Diagrams
- State Diagrams
- Activity Diagrams
- Use Case Diagrams
- Class Diagrams
- Object Diagrams
- Component Diagrams
- Deployment Diagrams

You are Here!
Collaboration diagrams

- Describe a specific scenario by showing the movement of messages between the objects
- Show a spatial organization of objects and their interactions, rather than the sequence of the interactions

Unlike a Sequence diagram, a collaboration diagram shows the relationships among the objects. A collaboration diagram does not show time (i.e., sequence)

- Keep in mind: Both are referred to as interaction diagrams but with different focus!
  - Sequence diagrams – message flows between objects based on time (i.e., sequence)
  - Collaboration diagrams – message flows between objects with no timing
ATM: Assume you have these objects

: User

: ATM

: Consortium

: Branch
First step to build a collaboration diagram

- Connect the objects

```
: User

: ATM

: Consortium  : Branch
```
Second step to build a collaboration diagram

1. Connect the objects
2. Draw the flow of messages

The collaboration diagram:
A simple collaboration, showing no interaction
Interaction shown on a collaboration diagram

1: okToBorrow

2: borrow

2.1: borrowed

aMember : BookBorrower

theLibraryMember : LibraryMember

theCopy : Copy

theBook : Book
Exercise

• Sketch a collaboration diagram for self-service machine, three objects do the work we're concerned with
  - **the front**: the interface the self-service machine presents to the customer
  - **the money register**: part of the machine where moneys are collected
  - **the dispenser**: which delivers the selected product to the customer

• Compare your collaboration diagram with that of a sequence diagram
UML Diagrams

Covered

Covered

You are Here!

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered
State Diagrams

- Also known as *statecharts* (invented by David Harel)
- Used primarily to model state of an object
- A class has at most one state machine diagram
  - Models how an object’s reaction to a message depends on its state
    » Objects of the same class may therefore receive the same message, but respond differently
Note: use of State diagrams

- Often used for modelling the behaviour of components (subsystems) of real time and critical systems....
Modelling states and events

The Book **states** could be:
- On shelf
- On loan
-maybe lost

The related **events** could be:
- Borrow
- return
Realising state diagrams

Return() → On shelf

borrow() ← On loan

Copy of book

:Book

onLoan

lost()

return()

onShelf

borrow()

final state

transition

initial state

state

event
Conditional notions

Conditional notation is used if the value of an object’s attributes determines the change of state (i.e., change the state under this condition...).

Important hint: For some guards use keywords like

**After** followed by expression

**When** followed by expression
Conditional notions

Means......
If balance<0, then change the state to overdrawn
If balance>=0, then change the state to Incredit

Important hint:
For expressing some events use keywords like
After followed by expression
When followed by expression
Conditional notions

Important hint:
For expressing some events use keywords like
**After** followed by expression
**When** followed by expression
Modelling states and substates

States of ATM machine itself...

- Idle
- busy
- Serving customer
- Customer served
- Out of order
Modelling substates

States of ATM machine itself... is rather trivial.... Let us see how we can model the sub state busy

Idle → busy

Serving customer → Customer served
Modelling substates for ATM machine

1. Reading card
2. Asking for PIN
   - Correct PIN entered
   - After three wrong trials
     - Reading card
     - Wrong pin
       - Retain card
       - Asking for transaction choice
         - Yes: Performing transaction
         - No: Asking do again?
           - Yes: Transaction successful
           - No: Reporting failure
             - Acknowledged by user
       - Choice made
         - Transaction failed
         - Card not readable
State diagram for ATM machine

Idle

Serving customer

Busy

Reading card

Asking for PIN

Correct

Wrong pin

Retain card

After Three wrong Trials

Correct

Out of order

Serving customer

Customer served

Retain card

Asking for transaction choice

Transaction successful

Transaction failed

Performing transaction

Yes

No

Customer served

After Three wrong Trials

Correct

Wrong pin

Retain card

Asking for PIN

Correct

Wrong pin

Retain card

Asking for transaction choice

Transaction successful

Transaction failed

Yes

No
Modelling concurrent states

States that occur in parallel
Exercise

- What are the states of the player?
- What are the events that cause state changes?
- What are the outputs that occur?
- What are the guards for the transitions?

Reference: David Rosenblum, UCL
• What would we model differently in an activity diagram for the player?
UML Diagrams

We are here

We are here
**Component Diagram**

- The component diagram's main purpose is to show the structural relationships between the components of a system.
- Component diagrams offer architects a natural format to begin modeling a solution.
- Component diagrams allow an architect to verify that a system's required functionality is being implemented by components.
- Helps to reason about non-functionalties.
- Developers find the component diagram useful because it provides them with a high-level, architectural view of the system that they will be building.
Architecture of the System

Client-tier

Middle-tier

Database-tier

SourceSafe Databases + SourceAnywhere Server

Internet TCP/IP

SourceAnywhere Client

three-tier style

Client-server style
N-tier architecture & components

Reference: Ivica Crnkovic
N-tier architecture & components

Reference: Ivica Crnkovic
Component Diagram shows a relationship between two components: an Order System component that uses the Inventory System component.
Component Diagram

All they mean the same: a component Order

UML version 2.0
Required/Provide Interface

- **Order**
  - Provided Interfaces:
    - OrderEntry
    - AccountPayable
  - Required Interfaces:
    - Person
Component Diagram

showing a component's relationship with other components, the lollipop and socket notation must also include a dependency arrow (as used in the class diagram). On a component diagram with lollipops and sockets, note that the dependency arrow comes out of the consuming (requiring) socket and its arrow head connects with the provider's lollipop.
Components Diagrams

- Architectural **connection** in UML 2.0 is expressed primarily in terms of **interfaces**
- Interfaces are **classifiers** with operations but no attributes
- Components have **provided** and **required** interfaces
  - Component implementations are said to **realize** their provided interfaces
  - A provided and required interface can be connected if the operations in the latter are a subset of those in the former, and the signatures of the associated operations are **compatible**
- **Ports** provide access between external interfaces and internal structure of components
- UML components can be used to model complex architectural connectors (like a CORBA ORB)
Component Diagrams

Ref: David Rosenblum, UCL
Exercise 1

Flight Booking service

Hotel Promotional Service

Car Hire Promotional Service

Sketch the components and interfaces corresponding to the given services.
Exercise 1

• Assume that Bob wants to book a holiday
  - Bob will book his holiday, where
    • He provides the following data: His origin airport, his destination, his dates of departure/return, and any other preference information (e.g., budget, luxury, etc)
  - Bob is interested in promotional offers for the period of his holiday
    • He wants to rent a car at his destination.
    • He wants to get good hotel deals during his stay.
Software Requirements

- After you book a holiday, the company shall provide the holidaymaker with promotional services, which include hotel deals and car rent promotional service at the destination and for the duration of her/his stay.
Flight Booking

Requires

Customer data

Provides

Flight Information
Customer Preference Information
Promotional Hotel Booking Service

Requires
- Flight Information
- Customer Preference Information

Provides
- List of available Hotels
You can even provide more services..

**Requires**
- Flight Information
- Customer Preference Information

**Provides**
- List of available Hotels
- List of Budget Hotel
- List of Five stars
- List of B&B
- List of 3 stars

Promotional Hotel Booking Service
Car Hire Promotional Service

**Requires**
- Flight Information
- Customer Preference Information

**Provides**
- List of available Car agents
- List of cars deals
Composite Structure in Component Diagrams

A composite structure depicts the internal realisation of component functionality

Ref: David Rosenblum, UCL
Ports

The ports and connectors specify how component interfaces are mapped to internal functionality.

Note that these 'connectors' are rather limited, special cases of the ones we've been considering in software architectures.

Ref: David Rosenblum, UCL
Ports

ports provide a way to model how component's provided/required interfaces relate to its internal parts
Ports

Connectors and ports also can be used to specify structure of component instantiations
Example

Stereotype
Guidelines to Componentization

• Keep components cohesive. A component should implement a single, related set of functionality.
  – This may be the user interface logic for a single user application, business classes comprising a large-scale domain concept, or technical classes representing a common infrastructure concept.

• Assign user interface classes to application components.
  – User interface classes, those that implement screens, pages, or reports, as well as those that implement “glue logic”.

• Assign technical classes to infrastructure components.
  – Technical classes, such as those that implement system-level services such as security, persistence, or middleware should be assigned to components which have the infrastructure stereotype.
Example

User interfaces assigned to application components

Infrastructure components
Guidelines to Componentization

• **Assign hierarchies to the same component.**
  - 99.9% of the time it makes sense to assign all of the classes of a hierarchy, either an inheritance hierarchy or a composition hierarchy, to the same component.

• **Identify business domain components.**
  - Because you want to minimize network traffic to reduce the response time of your application, you want to design your business domain components in such a way that most of the information flow occurs within the components and not between them.
    - Business domain components = services

• **Identify the “collaboration type” of business classes.**
  - Once you have identified the distribution type of each class, you are in a position to start identifying potential business domain components.
Example

Students, Facilities, Seminar, Schedule are Business Domain Components

Infrastructure components
Guidelines to Componentization

• Highly coupled classes belong in the same component.
  – When two classes collaborate frequently, this is an indication they should be in the same domain business component to reduce the network traffic between the two classes.

• Minimize the size of the message flow between components.
  – Merge a component into its only client. If you have a domain component that is a server to only one other domain component, you may decide to combine the two components.

• Define component contracts.
  – Each component will offer services to its clients, each such service is a component contract.
Guidelines to Componentization

Highly coupled classes belong in the same component

Ref: David Rosenblum, UCL
UML Diagrams

Covered

Covered

Covered

Covered

Covered

Covered

Covered

Covered

We are here

Taken from [Booch 1999]
Deployment Diagram

- Models the run-time configuration in a static view and visualizes the distribution of components in an application
- A component is deployed in which part of the software system architecture
- In most cases, it involves modeling the hardware configurations together with the software components that lived on
Deployment Diagram

- Deployment diagram depicts a static view of the run-time configuration of processing nodes and the components that run on those nodes.
  - **Node**: server, client etc.
- Deployment diagrams show the hardware for your system, the software that is installed on that hardware, and the middleware used to connect the disparate machines to one another!
- Models the run-time configuration in a static view and visualizes the distribution of components in an application.
- Deployment Diagrams
  A deployment diagram models the run-time architecture of a system.
  - It shows the configuration of the hardware elements (nodes) and shows how software elements and artifacts are mapped onto those nodes.
Node

- A Node is either a hardware or software element. It is shown as a three-dimensional box shape, as shown below.
Node Instance

- A node instance can be shown on a diagram.
  - An instance can be distinguished from a node by the fact that its name is underlined and has a colon before its base node type. An instance may or may not have a name before the colon.
  - The following diagram shows a named instance of a computer

![Diagram of a node instance](image)
Node Stereotypes

- A number of standard stereotypes are provided for nodes, namely «cdrom», «cd-rom», «computer», «disk array», «pc», «pc client», «pc server», «secure», «server», «storage», «unix server», «user pc». These will display an appropriate icon in the top right corner of the node symbol.
Artifact

- An artifact is a product of the software development process. That may include process models (e.g. use case models, design models etc), source files, executables, design documents, test reports, prototypes, user manuals, etc.
- An artifact is denoted by a rectangle showing the artifact name, the «artifact» keyword and a document icon, as shown below.
Association

- In deployment diagram, an association represents a communication path between nodes. The following diagram shows a deployment diagram for a network, depicting network protocols as stereotypes, and multiplicities at the association ends.
Node as container

A node can contain other elements, such as components or artifacts. The following diagram shows a deployment diagram for part of an embedded system, depicting an executable artifact as being contained by the motherboard node.
Example of three-tiers architectures

Many of real life web applications have three tier architectures
Deployment diagrams for three tiers

**UML 1.4**

Components deployed in an architecture

- **Database Server**
  - Oracle Database
  - Transaction Request

- **Application Server**
  - J2EE Server
  - Order Request

- **Client**
  - Web Browser
  - Swing Application
Example: Client server architectures
Example: Deployment diagram for a client server architecture

```
AppServer
<<TCP/IP>>
* 1

DisServer

<<deploy>>
<<deploy>>

<<artifact>>
orderProcessor.exe

<<artifact>>
networkScanner

<<deployment spec>>
networkScanner

<<artifact>>
 dbschema

<<manifest>>
ReposCustomer

<<manifest>>
ReposInternalRecords
```

UML 2.0
UML - End or the beginning?

[Diagram of UML models with various diagrams such as Sequence Diagrams, Use Case Diagrams, Class Diagrams, Object Diagrams, Collaboration Diagrams, Component Diagrams, State Diagrams, Activity Diagrams, Deployment Diagrams, Models, and Rational Software with a note taken from [Booch 1999].]
References to tools