Introduction to UML, the Unified Modeling Language

Jean-Paul Rigault
School of Computing (ESSI)
University of Nice Sophia Antipolis
Email: jpr@essi.fr

Outline

- Introduction: modeling and OO modeling
- Application modeling with Use Cases
- Class and object modeling
- State modeling

Introduction

- What is modeling?
- What is object-oriented modeling?
- What is UML?
- Technical activities in OO modeling
What is Modeling?

- Modeling is customary in engineering disciplines
- A model is a simplification of reality
  - Handling the real-world complexity
- The four aims of modeling
  - Visualize the system
  - Specify its structure and behavior
  - Provide a template to help us build the system
  - Document decisions made

Examples of models
- Textual description
- Catalogue, Data book...
- Blueprint
- Scale model
- Formal descriptions
  - Equations
  - Finite state machines
  - Set theoretic descriptions...
- etc.

What is Object-Oriented Modeling?

Technical activities in OO modeling

- Business model
  - Application modeling (Product definition, Requirement Analysis)
- Object modeling
  - Analysis and Design
  - Code and unit testing

What is Modeling?

Principles of Modeling

- The choice of a way of modeling influences
  - how a problem is tackled
  - how a solution is elaborated
- No single model is sufficient
  - Different levels of abstraction (refinement)
  - Different points of view

Example: models of a house

- Different levels of refinement
  - 3D view
  - rough floor plan
  - floor plan with room dimensions
- Different points of view
  - architect’s view
  - mason’s view
  - plumber’s view
  - electrician’s view

What is Object-Oriented Modeling?

Abstraction
Vehicle
Boat
Van
Aircraft
Car

Domain analysis
Basic Concepts

Real World

Exec

Test
Verif
Valid

Simulation

Modeling
Realization

Code

Vehicle

Plane
Boat
Car
Overview of UML

A Brief History

- Booch (Rose) Good for design and construction
- Rumback (OMT) Good for analysis and data-intensive systems
- Jacobson (OOSE) Good for the capture of requirements

<table>
<thead>
<tr>
<th>Coad-Yourdon</th>
<th>OMG request</th>
<th>OMG vote</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shlaer-Mellor</td>
<td>Rational UML</td>
<td>OMG 1994</td>
</tr>
<tr>
<td>etc.</td>
<td>1991</td>
<td>1992</td>
</tr>
<tr>
<td>Many OOAD methods (&gt; 50)</td>
<td>1993</td>
<td>1994</td>
</tr>
</tbody>
</table>

Overview of UML

What is UML and what is it for?

- **Software intensive systems...**
  - Enterprise information systems; Banking and financial services
  - Telecommunications
  - Transportation; Defense; Aerospace
  - Medical electronics
  - Scientific applications
  - Distributed Web-based services
  - etc.
- **But also non-software systems**
  - Workflow
  - Patient healthcare systems
  - Hardware design...

- Visualizing
- Specifying Architecture and behavior
- Constructing Allow code generation
- Documenting Textual and graphic descriptions

the artifacts of a software-intensive system

Overview of UML

What is a UML Model?

- **A Use Case view**
  - Functional requirements
- **Several object views**
  - Different concerns (architecture, behaviour...)
  - Different levels of description (analysis, design, implementation...)
Overview of UML

UML and Software Methodologies

- The UML is methodology independent
- However it is better suited to a development process that is
  - Use case driven
  - Architecture centric
  - Iterative and incremental

Technical activities in OO modeling

UML Views for Technical Activities

- Integration, validation, maintenance, management, etc.
- Code and unit testing
- Analysis and Design
  - CLASSES SEQUENCES STATES
- Object modeling
  - CLASSES
- Application modeling
  - USE CASES, ACTIVITY
- Business modeling
  - USE CASES, ACTIVITIES

Introduction to UML

Application Modeling

(Product Definition/Requirement Analysis) with Use Cases

Modeling Requirements in UML

- Functional requirements
  - Modeled as Use Cases
- Non-functional requirements
  - Some are specific to one use case
  - Some relate to technical issues addressed by implementation diagrams and models
  - Other in some supplementary documents, out of the UML scope...
The Use Case Approach

What is a Use Case?

- Set of sequences of actions that a system performs and that yields an observable result
- A set of related services provided by the system, together with scenarios of use
- Examples
  - Place an order, Subscribe to a service
  - Land an aircraft, Pilot a car...

What is an Actor?

- Actors are anything which interface with the system
  - People, other systems, hardware, software, networks...
  - They are roles, not necessarily full-fledged objects
  - Actors are usually named by nouns corresponding to roles
  - Actors participate or are interested in the result of Use Cases
- Actors are not part of the system
  - They are just at the boundary; They won’t produce code!

The Use Case Approach

A Simplified ATM

- The ATM is associated with only one bank
- It gives access to all the accounts the customer owns in the bank
- Several types of transactions can be chained in a single session
  - Withdrawal, deposit, transfer between the customer’s own accounts, consulting an account
  - The customer may obtain a printed receipt for each transaction

Several sorts of Actors
- Humans, systems...; Primary and secondary
- Primary actor
  - Triggers actions in the system
  - At least one per use case
- Secondary actors
  - Support the activity, do not initiate it
- Stakeholders
  - Interested in the actions, but not necessarily participating
The Use Case Approach
A Simplified ATM

- The card contains
  - an ID number
  - an expiration date
  - a secret code
- The card must be verified by the ATM
  - The customer cannot make more than 3 attempts to enter the correct code, otherwise the card is swallowed
- Interface
  - Keyboard and screen, with menus
  - The customer may cancel the current transaction at meaningful points

Minimal connection with the bank
- When a session is opened, a connection makes it possible to known all the accounts the customer owns
- Every night, the information about the current day transactions are transferred to the bank

Manual interventions: an employee
- empties the machine and/or fills it with bank notes
- supplies consumables and does any maintenance that may be needed

The Use Case Approach
Modeling Use Cases

- Informal descriptions (non UML)
  - Describing scenarios
  - Just prose
- More formal description (UML models)
  - Use Case diagrams
  - Activity diagrams
  - (Sequence and Collaboration diagrams)

Informal UC Description
Actors

- Identifying actors
  - Who uses, installs, starts up, shuts down, maintains the system?
  - Who provides (gets) information to (from) the system?
  - What other systems use the system?
  - Who has non-functional needs with respect to the system?

ATM example
- Customer
- Bank (computer system)
- Technician

What about the card?
Informal UC Description

Top Level Use Cases

- A full set of related system functionalities
- Identifying use cases
  - Actors expectations
  - Creating, consulting, updating or deleting system information?
  - Notification of an actor state change or of an external event
- Start to end
  - a complete use of the system
- ATM example
  - Handle Session: Run a customer complete session from start (card insertion) to end (card retrieval or swallowing)
  - Transfer Daily: Nightly retrieval of information by the Bank computer system
  - Maintain: Run a maintenance session

Informal UC Description

Use Cases and Scenarios

- Use cases classify the ways of using the system
- To each use case correspond many possible usage scenarios
  - A scenario is a sequence of interaction between the system and one or several actors
  - It expresses a particular path to traverse its use case
- Thus scenarios
  - are instances of use cases
  - specify use cases

Informal UC Description

Use Cases and Scenarios

- Each use case has generally
  - a primary scenario: typical usage
  - several secondary (more exceptional) scenarios
- A scenario involves one or several actors
  - Usually one primary actor, and possibly supporting ones
    - It may have an initiator actor (often, the primary one)
  - A scenario may be of interest to stakeholders, which are not always participating actors
    - Example: regulation policy, board of directors, share owners...

Informal UC Description

Alistair Cockburn’s Style

- Describe a full Use Case
  - with a primary (typical) scenario
  - and possible variants
- Involve
  - a primary actor, a possible initiator actor (may be the primary) and possible supporting actors
  - the interest of some stakeholders
- Several possible patterns
  - Brief (a simple sentence/paragraph)
  - Informal text, yet obeying simple writing rules...
  - Highly structured text (fully-dressed)
Informal UC Description
A. Cockburn’s Style: Brief

Handle Session Use Case

The Customer inserts the card into the ATM. After verification and authentication, the customer may perform one or several transactions. When done, the customer gets the card back and leaves.

Informal UC Description
A. Cockburn’s Style: Informal

Use Case: Handle Session
Primary actor (and initiator): Customer
Supporting actor: Bank

The customer inserts the card into the ATM. The ATM verifies the card and authenticates the customer. The ATM retrieves customer information from the Bank. The customer selects a transaction among withdrawal, deposit, transfer, and consult. After completion of the transaction, the Customer may decide to select another transaction or to quit (in which case, he/she gets the card back).

If the card is invalid or the customer fails to authenticate, the card is swallowed.

Informal UC Description
A. Cockburn’s Style: Fully-dressed (1)

- For each use case
  - Actors and stakeholders: as for informal
  - Pre- and post-conditions
    - Pre-condition:: must be true before starting (initiating) the UC
    - Post-condition: guaranteed to be true after UC completion
  - One primary scenario (composed of 2-8/10 steps)
  - Several variants associated with the (primary) scenario steps
  - Other information: Exceptions and errors (specific variants), Data needed, Constraints...

Informal UC Description
A. Cockburn’s Style: Fully-dressed (2)

- Primary scenario steps and variants are numbered
  - 1,2,3... in primary
  - 2a for first variant, expressed in informal style, of step 2
  - 2b1,2b2,2b3... for variant b, expressed in fully-dressed, form, of step 2
- For each step
  - Express a functionality in a simple sentence
  - The subject of the sentence is one of the actors or the system
  - Use positive verbs in active form
Informal UC Description
A. Cockburn’s Style: Fully-dressed  (3)

Use Case: Handle Session
Primary actor (initiator): Customer
Supporting actor: Bank
Precondition: ATM is free
Primary Scenario
1. Customer inserts the card into ATM.
2. ATM asserts card validity.
3. Customer authenticates with ATM.
4. ATM gets Customer information from Bank.
5. Customer performs a transaction.
7. ATM ejects card. Customer takes it
Postcondition: ATM is free and all transactions have been recorded

Informal UC Description
Best Practices

- Give a meaningful name to each use case
  - Usually an active verbal form
  - The name should correspond to the objective of the expected (business) service
- Do not model outside the system!
  - Bad Customer use cases
  - choosing a particular ATM machine
  - deciding how much money to withdraw
- Do not model at too low level!
  - Bad Customer top level use cases
    - insert the card
    - enter secret code, get ticket
    - select transaction type, select accounts, select amount...
    - Even worse
    - push the ENTER button on the keyboard...

Informal UC Description
A. Cockburn’s Style: Fully-dressed  (4)

Use Case: Handle Session
Primary actor (initiator): Customer
Supporting actor: Bank
Precondition: ATM is free
Primary Scenario
1. Customer inserts the card into ATM.
2. ATM asserts card validity.
3. Customer authenticates with ATM.
4. ATM gets Customer information from Bank.
5. Customer performs a transaction.
7. ATM ejects card. Customer takes it
Postcondition: ATM is free and all transactions have been recorded

Informal UC Description
Best Practices

- Give a meaningful name to each use case
  - Usually an active verbal form
  - The name should correspond to the objective of the expected (business) service
- Do not model outside the system!
  - Bad Customer use cases
  - choosing a particular ATM machine
  - deciding how much money to withdraw
- Do not model at too low level!
  - Bad Customer top level use cases
    - insert the card
    - enter secret code, get ticket
    - select transaction type, select accounts, select amount...
    - Even worse
    - push the ENTER button on the keyboard...

Variants
2a Card is invalid: ATM swallows it; this ends the session.
3a Customer fails to authenticate: ATM swallows card; this ends the session.
4a The connection with Bank is impossible or interrupted: ATM ejects the card; this ends the session.
5a Customer quits immediately.
6a Customer chooses to perform another transaction: go to step 5.
7a Customer does not get card back quickly enough: ATM beeps
  * Customer may cancel session: ATM ejects the card.

Formal UC Description
Top Level UC Diagram
The realization of each top level use case requires sub-functionalities:
- Lower level Use Cases, UC fragments...
- ... which are not real (“start-to-end”) use cases

Functional-like decomposition, not really object-oriented!
Hierarchical decomposition: relationships between use cases:
- Inclusion
- Generalization
- (Extension)

The interpretation of « extend » varies:
- The UML standard does not specify the nature of the condition at the “extension point”
- Corporate policies may apply!

Note the direction of the dependency arrow:
- The lower level (extending) use case depends on the higher one
- The higher level one may not depend on the extension
- This remark helps to distinguish « extend » from « include »
The specialized and generalized use cases are in a “sort of” relationship.

- Specialized
  - Handle Transaction
  - Handle Deposit
  - Handle Withdrawal
  - Handle Consult
  - Handle Transfer

Generalized

It may be convenient to apply generalization to actors.

- A frequent flyer and a regular passenger are both passengers
  - Each of them may play the (general) role of Passenger in a use case
  - However, e.g., a frequent flyer may interact with specific use cases

Use Case: Handle Session
Primary actor (initiator): Customer
Supporting actor: Bank
Precondition: ATM is free
Primary Scenario
1. Customer inserts the card into ATM.
2. ATM asserts card validity.
3. Customer authenticates with ATM.
4. ATM gets Customer information from Bank.
5. Customer performs a transaction.
7. ATM ejects card, Customer takes it
Postcondition: ATM is free and all transactions have been recorded

Avoid « extend »
- unless specific corporate policy!

Usually actors are not repeated on decomposed diagrams
- unless this can bring some added information...
  - Interaction of an actor with a specific sub-UC
  - Interaction of a specialized actor with a sub-UC
- or specific corporate policy!

Do not over-decompose
- Sub-UC should have “functional substance” (that is, in general, several significant sub-scenarios)
Scenarios are instances of use cases

Natural language for expressing scenarios
- has a strong power of expression
- lacks precision
- makes it difficult to express information like timing and concurrency

Need for formal (UML) expression of scenarios and use cases

Sequence diagrams
- Only one scenario per diagram
- Accent on the time flow
- At the Use Case level, express scenarios involving the system and some of its actors
- May be used also as a mean to model business activity (inter-actor scenarios)

Activity diagrams
- Several scenarios (even a full use case) at once
- Sort of (concurrent) flow charts...

Express a flow of control
- Composed of Action States (or Activities) connected by Transitions
- Transitions may be conditional
- A Transition is fired if its origin Activity has completed its action and if its (optional) condition is true

Diagram:
- Time
- Customer
- ATM
- Bank
- Insert card
- Prompt for selection
- Select transaction
- Produce result
- Quit
- Eject card
- Get customer info

Condition: [date OK]
Transition
Verify Date
Action State
Verify Code
Formal UC Description
UC Scenarios: Activity Diagrams

Handle Session UC
- Verify Card
  - [ not Card_OK ]
    - Swallow
      - [ not Passed ]
        - Get Customer Info
      - [ Passed ]
        - Authenticate Customer
          - [ Card_OK ]
            - Handle Transaction
              - [ not Quit ]
                - Eject
                  - [ Quit ]

- Fork and join bars
  - Express concurrency
  - Fork spawns several concurrent activities
  - Join waits for all incoming activities to complete

- Logical concurrency
  - Avoid enforcing non relevant sequencing
  - The designer will have the freedom to choose how to implement (even in non-concurrent ways)

Formal UC Description
UC Scenarios: Activity Diagrams

Process Order UC
- Receive Order
  - Establish Invoice
    - Prepare Delivery
      - Deliver

- User Interface
- Accounting Dept
- Delivery Dept

Formal UC Description
UC Scenarios: Activity Diagrams

Process Order UC
- Receive Order
  - Order
    - Establish Invoice
      - Prepare Delivery
        - Invoice
          - Deliver
Formal UC Description
UC Scenarios: Activity Diagrams

- Accent is on the flow of control
- Describe at once a full set of scenarios
- Describe concurrent operations
- Created objects and responsibilities of objects may be represented
  - They may become too complex
  - Risk to do functional analysis

Practice of Use Case Models
Organizing the Model

Use Case Model
Conclusion: Difficulty and Drawbacks

- Use Case modeling is difficult
  - Homogeneity, completeness, consistency...
- UML “formalism” is simple, even simplistic
  - No real semantics
  - No formal description of textual scenarios
  - Need for predefined interpretations, corporate policies
- Textual scenarios awkwardly express complex control
  - Loops, conditionals...

Use Case Model
Conclusion: Advantages

- Simple to elaborate, understand, and communicate
  - Even to non-computer scientists
- Focus on user needs, not on solution or architecture
  - Avoid architectural drift in object-orientation
  - Ease traceability from functional needs to implementation
- Facilitate setting up integration tests
  - From use cases, one can derive test cases
Introduction to UML

Class and Object Modeling
(Object-Oriented Analysis)

Object-Orientation Overview
A World of Objects

- The (application) world is composed of objects
- These objects are linked together
  - Static relationships (links)
- These objects react to stimuli (messages)
  - Either internal or external
  - Originating from other objects or from outside the system
- These objects have an internal state
  - Internal data (attributes) and status of the links with other objects
  - The state may change when the objects are stimulated

Analysis Model

- Object-orientation overview
- The elements of analysis models
- Class diagrams
- Realizing use cases and scenarios
- Practice of analysis models

Object = Identity + State + Behaviour

Objects should be distinguishable
The identity is independent from the state

Internal data values
Status of links with other objects

Operations, events, messages...
Public interface

Objects have “crisp” conceptual boundaries (Booch, 1994)
Object-Oriented Overview

What is an Object?

- **Static information: architectural aspect**
  - List of operations (interface)
    - The messages the object can accept and react to
  - State values
    - Possible values of internal data (attributes)
    - Possible links with other objects, that are message transport media
- **Dynamic behaviour: control aspect**
  - State evolution and messages sent to other objects
  - Triggered by message flows

Object-Oriented Overview

Candidates for Objects

- **Material (tangible) things**
  - Cars, planes, devices, people...
  - Invoices, documents...
- **More abstract things**
  - Flights, transactions...
- **People roles**
  - Client, server, customer...
- **Functions, processes**
  - Reifying, objectifying...
- **And many other things**

Object-Oriented Overview

Abstraction: Points of View

- The system analyst should capture the view(s) of an object that is/are appropriate for the application and model them in a coherent way.

Object-Oriented Overview

What's in a Class?

- A group of objects sharing common properties:
  - common structure:
    - same attributes
    - same possible relations with other objects
  - common behaviour: same operations
- An abstract data type
- A model to instantiate objects
  - A class defines the possible behaviours and the information structure of all its instances
**Object-Orientation Overview**

**What's in a Class?**

- Different instances may have their operations activated in different ways, different sequences... Hence, they may be in different states
- Different instances, even if they have the same attribute values, have different identities

**Instance** = a particular object belonging to a class

---

**Object-Orientation Overview**

**What's in a Class?**

- Class = Abstract Data Type (ADT)
- ADT= value set + operations
  - Built-in types (integer, real, boolean)
  - User defined types
- User defined types are treated as “first class citizen”
  - Copying objects,
  - Passing objects as parameters or return value,
  - Using operators on objects

---

**Object-Orientation Overview**

**Links between Objects**

- The architecture of the object system
  - Permanent (or semi-permanent) links
- Examples
  - A travel agency is connected to an airline consortium
  - A car has wheels, engine, body...
  - A PC is part of the Internet
  - An invoice refers to a Customer
  - John owns a Rolls Royce car
  - John’s car is on the road to London
  - Mary is employed by Amadeus
  - Jacques married Bernadette

---

**Object-Orientation Overview**

**Object States**

- A car may be running, parked, or broken down
- An order may be emitted, shipping, cancelled, completed, archived...
- A flight may be on-time, delayed, cancelled, hijacked...
- Jacques married Bernadette
  - The existence of the wedding link induces a particular state for both participants (each of them is married)
Object-Oriented Overview

Communication by Messages

- **Usual asymmetric**
  - The emitter knows (designates) the receiver
  - The receiver does not know the emitter

- A message triggers an operation of the object
  - The object may change state as a consequence of the operation
  - Usually a message and its associated operation bear the same name

- **Message may carry values (parameters)**

**Example:**
- A travel agency reserves seat 3A for Mr. Smith on flight BA349 next Monday

After reservation, the state of the flight has changed (may be it became full?)
The Elements of Analysis Models

Use Case Realization

- **Realizing a use case**
  - Finding classes collaborating for realizing the use case
  - Finding their attributes and operations
  - Finding relationships between these classes
  - Refining the use case and scenarios description to take the new classes into account

- A group of classes collaborating for realizing a use case is called a **Collaboration**

![Diagram showing collaboration between use case and classes](image)

The Elements of Analysis Models

Dependency

- **Realization**
  - Semantic relationship between classifiers wherein one component specifies a contract that the other guarantees to carry out
  - Usually relationship between some interface and its implementation (in a broad sense)

![Diagram showing realization between use case and product definition model](image)

The Elements of Analysis Models

Dependency

- A semantic relationship between two things
- When the target changes, the source might have to change
- Generally unidirectional
- Does not imply a static (permanent) kind of relationship

![Diagram showing dependency between analysis model, person, and document](image)

Analysis Model

- Object-orientation overview
- The elements of analysis models
  - Class diagrams
  - Realizing use cases and scenarios
  - Practice of analysis models

![Diagram showing analysis model](image)
Class Diagrams

- Associations
- Generalization
- Generalization and aggregation
- Abstract classes and operations
- Stereotypical classes
- Constraints
- Best practices

Class Diagrams

Class Diagrams
Class, Attribute, and Operation

<table>
<thead>
<tr>
<th>Person</th>
<th>name</th>
<th>Data carried by each instance of the class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>attributes</td>
<td>Object behavior</td>
</tr>
<tr>
<td></td>
<td>operations</td>
<td>Operations that can be applied to each instance</td>
</tr>
</tbody>
</table>

Class Diagrams

Class Diagrams
Classes for the ATM

<table>
<thead>
<tr>
<th>Card</th>
<th>ATM interface</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Class Diagrams

Class Diagrams
Association and Link

- Association
- Link
- A structural relationship describing a set of links
- Some special cases (aggregation, composition)
- Generally implies mutual knowledge
- Corresponds to stable or (semi-)permanent relationship

<table>
<thead>
<tr>
<th>Man</th>
<th>wedlock</th>
<th>Woman</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bill : Man</th>
<th>hilary : Woman</th>
</tr>
</thead>
</table>
Class Diagrams
Attribute, Operation, Association

Person
- name: string
- address: string

Company
- name
- location = mainpremisse
- hire(Person)
- dismiss(Person)

Attributes have a name (and a type)
Operations may have typed arguments; they can be inferred from scenarios
Default values of attributes are possible
Associations have a name

Class Diagrams
Instance of a Class (Object)

Object (Instance)

Person

bill : Person
name = “Bill Gates”
address = “Redmond, WA”

Jane : Person
name = “Jane B.”
address = “Paris, FR”

Class Diagrams
Associations: Names and Roles

Company
- employs
  - role

Person
- parent
  - role

Class Diagrams
Associations: Multiplicity

Person
- parent
  - role

Woman
- wedlock
  - role

Company
- employs
  - role

Person
- parent
  - role

Man
- wedlock
  - role
Class Diagrams
Associations: Multiplicity

- 2
  - exactly 2
- 1..4
  - from 1 to 4
- 1..*
  - at least 1
- *
  - 0 or more (0..*)
- 2, 4..7, 10..*
  - 2, 4 to 7, 10 or more
- C
  - default is (exactly) 1

Class Diagrams
Associations: Association as Class

- Exactly one instance of class Job per link
- The lifetime of the association Job is the same as the one of the link

Class Diagrams
Associations: Aggregation

- Whole/Parts relationships
  - "is part of", "is composed of"
  - Antisymmetric
- Two forms
  - Weak aggregation (simply Aggregation)
  - Strong aggregation (Composition)

Class Diagrams
Associations: (Weak) Aggregation

- Weak coupling
  - Sharing is possible
  - Lifetime of component is not dependent of composite

- An instance of Person may be employed by several companies
- An instance of Person may be part of several families
Class Diagrams
Associations: Composition

- **Strong coupling**
  - No sharing
  - Life-time of component usually starts and ends with composite

![Diagram showing associations: Stock, Car, Wheel, Engine, and Body with cardinality relationships.]

Class Diagrams
Class Diagram for the ATM

- **Card** is inserted into **ATM interface**
- **Session** related to **Customer Information**
- **Transaction** is involved with **Transaction Record**

Class Diagrams
Generalization

- **Relationship**
  - Between classes
  - No cardinality
  - Anti-symmetric and transitive
- **"Is A", "Is A Sort Of"**
  - Substitutability principle
- **The sub-class depends on its base class(es)**
- **Support of generalization in UML**
  - Multiple inheritance is supported
  - Multiple classification indirectly & partially supported
  - Dynamic classification is almost not supported
**Class Diagrams**

**Generalization: Substitutability**

- **B derives from A**
  - Any instance of B is also an instance of A for any purpose of an A
  - Any instance of B may be substituted to an instance of A
- **B may**
  - Add new properties to the ones inherited from A
  - Redefine operations inherited from A
- **B cannot get rid of properties inherited from A**

**Class Diagrams**

**Generalization: Polymorphism**

**Class Diagrams**

**Generalization: Multiple Inheritance**

**Class Diagrams**

**Abstract Classes and Operations**

- **Abstract class**
  - Class with no instances
  - Usually at the top of a class hierarchy
- **Abstract operation**
  - An operation without implementation (i.e., realization)
  - An abstract operation makes the whole class abstract
  - If derived classes do not define the operation implementation, they are also abstract
Class Diagrams
Abstract Classes and Operations

Abstract class (top of hierarchy)

Figure
draw()

Ellipse
draw()

Rectangle
draw()

Concrete classes

Circle

Square

Class Diagrams
Stereotypical Classes

- Entity class
  - Long-lived (persistent) information

«entity»

- Control class
  - Coordination, sequencing, control...

«control»

- Boundary class
  - Interaction between the system and its actors

«boundary»

Class Diagrams
Stereotypical Classes

- Restrict, clarify, or specify the semantics of model elements
  - Can be applied to any kind of model element

- Constraints may be expressed by
  - Free text
  - Special constraint language (OCL)
  - Constraints are enclosed within braces {}

- Predefined constraints
  - Type of constraints (stereotype, pre- and post conditions...)
  - Individual constraints
A condition that must be verified by any instance at any time

### Constraints: Class Invariant

- **Adult**
  - **age**: Integer
  - **is_emancipated()**: Boolean

#### Constraint stereotype

- «invariant»
  ```
  { age >= 18 or is_emancipated() }
  ```

### Constraints: Pre- and Post-Conditions

- **Person**
  - married: Boolean
  - gender: enum (M, F)

#### Constraint stereotype

- «precondition that must be verified by any instance at any time»

- **marrype(Person)**

### Constraints: Class Contract

- **Before an operation**
  - Invariants and pre-condition must be true

- **After an operation**
  - Invariant and post-condition should be true

- **Generalization and constraints (substitutability)**
  - Ideally, a sub-class should also verify its base classes invariants
  - Ideally an operation overridden in a sub-class may weaken the pre-condition and strengthen the post-condition of its base class(es) (“Demand less, promise more” principle)
  - UML does not really enforce these rules
Class Diagrams
Constraints: Predefined Constraints

Company 
  0..1 employs * 
  manages * 

Person

Unemployment Agency 
  0..1

Class Diagrams
Best Practices

Avoid meaningless classes
- No attributes nor operations
- No significance with respect to the problem...
- At the analysis level, classes represent abstraction of the application, they are not a programming hack to reuse code

Avoid too big classes
- Many attributes, many operations, not all of them are needed at a given time
- Too many responsibilities, represent several abstractions...
- Use inheritance, aggregation, association instead

Avoid explicit special operations
- Set, get, constructors, destructors, assignments....

Attribute versus association
- Use attribute notation for primitive (non-object) types
- For objects prefer aggregation or even regular association, with roles

Class Diagrams
Best Practices

RoundObject

Balloon
Wheel

Ball
Egg
SeaUrchin

Restaurant

tables
cards
dishes
waiters
patrons

numberOfMeals()

Table
serve

Waiter

Patron

seat(table)

Card

add(dish)

Dish

Car

color: String
power: Integer

owner

Person

Engine
Avoid explicit circular dependency
- In particular, a base class should not depend on its sub-classes

Beware of the “fragile base” class problem

Avoid redundancies
- Do not represent the same information with two different ways
  - Attribute and association
  - Redundant attributes

Do not confuse dependency and association
- An association describes a relationship between instances
- An association usually implies mutual dependency

This implies the diagram on the left, but adds information related to instances

At analysis level, avoid programming artifacts

Also violate previous slide advice
Analysis Model

- Object-orientation overview
- The elements of analysis models
- Class diagrams
- Realizing use cases and scenarios
- Practice of analysis models

Realizing Use Cases and Scenarios

- Use Case realization as a Collaboration
- Realizing scenarios
- Sequence diagrams
- Collaboration diagrams
- Interaction diagrams: comparison
- Activity diagrams and objects
- Best practices

Realizing UC and Scenarios

UC Realization as Collaboration (1)

- List all the classes (or roles) that together realize a given use case
- Allow traceability from Use Case Model to Class Model

Realizing UC and Scenarios

UC Realization as Collaboration (2)

- Collaboration
  - Defines an interaction
  - A society of roles, objects, or other elements to provide some cooperative and synergic behavior
  - Possibly structural as well as behavioral
  - Used to model Patterns (design or analysis patterns)
Realizing UC and Scenarios

Realizing Scenarios

- The requirement scenarios do not involve objects (and classes)
  - Except for actors
  - There is a need to revisit the scenarios in the light of the analysis classes
  - The refined scenarios are part of the use case realizations (collaborations)
- Using formalisms will
  - make descriptions more precise
  - help the analysis work

Realizing UC and Scenarios

Sequence Diagrams

- Describe one scenario
- Involve objects and messages (stimuli)
  - Messages exchanged between instances are called stimuli
  - A stimulus can be calling an operation (with possible parameters)
    - An operation is attached to a particular class
    - An operation call is a synchronous communication
  - A stimulus can be sending a signal
    - A signal is itself a classifier, a priori independent of classes
    - Roughly speaking, the instances of a signal are its occurrences
    - Sending a signal is an asynchronous communication
- Accent is on the time flow of events

“Scenario diagrams”

- Interaction diagrams: one scenario at a time
  - Sequence diagrams
  - Collaboration diagrams
- Activity diagrams: several scenarios (a full use case?)
Realizing UC and Scenarios

Sequence Diagrams

- Describe one scenario
- Involve objects and messages (stimuli)
- Accent is on messages exchanged and links between objects
- Do not confuse
  - Collaboration: the realization of a use case, which usually appears in a Use Case diagram
  - Collaboration diagram: a form of interaction diagram expressing a scenario
Realizing UC and Scenarios

Interaction diagrams: Comparison

Sequence and collaboration diagrams

- Common properties
  - For expressing scenarios in the analysis model, both forms are almost equivalent, although the focus is different
  - Only one scenario at a time
  - Help to define necessary operations of classes
  - The number of diagrams may become huge
  - Risk of redundancy in the diagrams for scenarios with only little variations (difficult to maintain)
  - Some useful features are awkwardly represented (e.g., loops and conditionals)

- Differences
  - Sequence: temporal aspect (stress on message/signal sequencing)
  - Collaboration: architectural aspect (objects and links, roles of objects...)
  - For the design model, collaboration diagrams may be richer
    - Roles of objects
    - Some form of concurrency
    - Sequence nesting...

Realizing UC and Scenarios

Best Practices

- Expressing use case and scenarios in the customer’s language (usually natural) remains mandatory
- Do not use too much formalism!
  - Avoid redundant scenario descriptions
  - For instance, remember that sequence and collaboration diagrams are equivalent and that it is generally useless to use both to describe the same scenario
  - Also recall that activity diagrams contain several scenarios
- Limit the number of objects, swimlanes, and interactions in diagrams

Analysis Model

- Object-orientation overview
- The elements of analysis models
- Class diagrams
- Realizing use cases and scenarios
  → Practice of analysis models
Practice of Analysis Models
Organizing the Model

TheSystem
UC Mode

Practice of Analysis Models
Model Global Properties

- Scenarios (and activity diagrams) should be attached to Use Case realizations (i.e., Collaborations)
- All scenarios of interest should be represented in a “scenario diagram”
  - It is almost impossible (not suitable?) to describe all scenarios
- Completeness of the Analysis Model
  - Every Use Case should have its own realization (Collaboration) listing the classes that “implement” it
  - A class should be part of at least one UC realization (Collaboration)

Practice of Analysis Models
Scenarios-Collaboration Completeness

- A class contributing to a UC realization (Collaboration) should appear in at least one “scenario diagram”
- If an instance appears in a “scenario diagram”, its class should be associated with the corresponding UC realization (Collaboration)

Practice of Analysis Models
Interaction-Class Completeness

- A stimulus received by an object should be part of the interface of its class (it is usually one of its operations)
- Ideally, public operations should appear in some “scenario diagram”
State Modeling
(Object-Oriented Analysis)

State-Transition Diagrams
- Attached to a class (with a reactive behaviour)
- Describe the dynamic behaviour of any instance when receiving events
- Relationships between events and states
- Deterministic finite state machines
  - Define the object states, their sequences, and the corresponding transitions
  - Express the control behaviour of an instance in any scenario where it may appear
  - Model borrowed from Harel’s StateCharts

State and Events

Events
- Trigger state transitions
- An event can be
  - A stimulus: operation call or signal
    - A stimulus originates from some object (possibly the receiving one)
  - An operation call requires some processing from the receiving object
  - A signal notifies some information; it is an object
  - A condition changing its truth value
  - A watch dog elapsing
- Events may carry value (parameters)
- Events have no duration

States
- Abstraction of attribute values and link status
- The state of an object specifies
  - The way it reacts to received events
    - A state specifies the events to which it is sensitive
    - Some events may be ignored when in a state
  - The activities the object performs while in this state
- State duration is not null
- Two predefined sorts of states
  - Initial state: corresponds to an instance “creation”
  - Final state: corresponds to an instance “destruction”
State and Events
States and Events Separation

- Events separate states
- States separate events

State-Transition Diagrams
States

- **initial state**
- **final state**

**action()**: some processing, with an ideally null duration; usually corresponds to a (private) operation of the class

**activity()**: as action() except that it may have non-null duration

State name

- do: activity()
- entry/action()
- exit/action()
- event/action()

Activity to perform while in the state
Action to perform when entering the state
Action to perform when leaving the state
Action to perform in response to an event without leaving the state

State-Transition Diagrams
States and Transitions

- State entry
  - The entry actions are performed
  - The do activity is started
- **While in state, wait for**
  - Either the termination of the do activity (normal termination)
  - Or an event listed in a event/action clause
  - Or an event in the trigger part of an outgoing transition (preemption)
- Normal termination
  - If there exists a normal termination transition the condition of which holds (or is absent), execute the exit actions and traverse transition
  - Otherwise keep waiting
- Event/action clause
  - Suspend the do activity, execute the action, and resume the do activity
- Preemption
  - Abort the do activity, perform the exit actions, and take the transition

As usual in UML all items are optional:
If no label, normal termination transition

1/10/2014 09:50  © 2004 Jean-Paul Rigault  137

1/10/2014 09:50  © 2004 Jean-Paul Rigault  138

1/10/2014 09:50  © 2004 Jean-Paul Rigault  139

1/10/2014 09:50  © 2004 Jean-Paul Rigault  140
State-Transition Diagrams

States and Transitions

- ST diagrams represent deterministic finite state machines
- For a given state, its outgoing transitions must be exclusive
  - If two outgoing transitions are triggered by the same event (or no event at all), their guards must be exclusive
- UML state-machine model does not cope with the problem of simultaneous events
  - Events are supposed to be serialised prior to be delivered
  - Hence two events cannot be simultaneous

State-Transition Diagrams

Simple ST Diagrams

- At gate
  - do: boarding()
    - go: [boarding terminated]
    - [gate free]
- Taxiing
  - do: reach_runway()
    - [speed < Limit]
    - clear_for_take_off [runway lined up]
    - /^ATC_request_for_landing
- Taking off
  - do: take_off()
    - airborne
- Landing
  - do: land()
- Flying
  - do: follow_flight_plan()

State-Transition Diagrams

Simple ST Diagrams

- At gate
- Taxiing
- Taking off
- Landing
- Flying

This diagram omits details of states and transitions: this does not mean they do not exist; this simply means they are not shown!

State-Transition Diagrams

Simple ST Diagrams

- White move
  - Black move

- White win
- Black win
- Null

- White win
- Black win
- null
- White win
State-Transition Diagrams
Simple ST Diagrams

- Idle
- Visible

button_down/display_menu()
button_up/erase_menu()
do: execute()
cursor_move/mark_selection()

Spontaneous transition
(termination of do:)

State-Transition Diagrams
Hierarchical ST Diagrams

- Neutral
- Backward

F N B

Neutral

The history pseudo-state memorises the current sub-state and forces its re-entry when re-entering the macro-state

State-Transition Diagrams
Hierarchical ST Diagrams

- Neutral
- Backward

1st
2nd
3rd

Forward

stop up down

State-Transition Diagrams
Hierarchical ST Diagrams

- Neutral
- Backward

F N B

Neutral

The history pseudo-state memorises the current sub-state and forces its re-entry when re-entering the macro-state

State-Transition Diagrams
Hierarchical ST Diagrams

- Neutral
- Backward

F N B

Neutral

The history pseudo-state memorises the current sub-state and forces its re-entry when re-entering the macro-state

State-Transition Diagrams
Hierarchical ST Diagrams

- Neutral
- Backward

F N B

Neutral

The history pseudo-state memorises the current sub-state and forces its re-entry when re-entering the macro-state

State-Transition Diagrams
Hierarchical ST Diagrams

- Neutral
- Backward

F N B

Neutral

The history pseudo-state memorises the current sub-state and forces its re-entry when re-entering the macro-state

State-Transition Diagrams
Hierarchical ST Diagrams

- Neutral
- Backward

F N B

Neutral

The history pseudo-state memorises the current sub-state and forces its re-entry when re-entering the macro-state
Introduction to UML

Annex 1
A Brief Object-Oriented Glossary

Object-Oriented Glossary

- **Link**
  - Static relationship between objects
  - Provide infrastructure for exchanging messages

- **Object**
  - Abstraction of a concept in the application world
  - Has crisp boundaries
  - Owns a state, a behaviour, and an identity
  - Has links with other objects
  - Exchanges messages with them

- **Abstraction**
  - Separate usage information from implementation details

- **Modularity**
  - Group semantically related things together

- **Encapsulation**
  - Hide unnecessary information

- **Message**
  - Dynamic flow of information between objects
  - Need links to travel

Object-Oriented Glossary

- **Association**
  - Relationship between classes, modelling links that the instances of the classes may have together
  - Model of the links between instances of classes
  - Links are instances of associations

- **Class**
  - Group of objects with identical structure and behaviour
  - Model for a group of objects
  - Defines a type

- **Type**
  - Set of values together with legal operations on these values

- **Derived class**
  - Inherits from all members (attributes and operations) of its super class
  - Can only add new members or redefine some operations

- **Abstract class**
  - Has no instance
  - Defines common attributes and operations of a set of classes

- **Abstract operation**
  - Only defines the signature of an operation, not its implementation

- **Polymorphism**
  - Ability of related objects to answer the same message, while in different ways, depending on their class

Object-Oriented Glossary

- **Instance of a class**
  - Object following the model provided by the class

- **Instance**
  - Object belonging to a class
Introduction to UML

Annex 2
Synopsis of UML Model Elements

Model Elements

Model Element

"Thing" Abstractions, first-class citizen in the model

Relation Tie "things" together

Diagram Group interesting collections of things and express (some of) their relationships

A Conceptual View of UML

- Model elements
- Rules
- Common mechanisms
- A UML model

Model Elements

Summary

Model Element

"Thing"

Relation

Diagran

Model Element

"Thing"

Relation

Diagram

Model Element

"Class"

Interface

Collaboration

Use Case

Interation

State Machine

Note

Package

Structural

Behavioural

Organizational

Annotational

Interaction

Use Case

Collaboration

Activity

Deployment

Class

Object

State-Transition

Component

Note

Generalization

Realization

Dependency

Association

Design Elements

Summary

Note
Model Elements

Examples of “things” (1)

- “Classifiers”: elements that may have instances

  Place an order
  A use case
  person.cpp
  A component

  Person
  name
  address
  move()
  marry()
  A class
  FileServer
  A node

Model Elements

Examples of “things” (2)

- Packages
  - The only organisational element
    - Represent (a form of) sub-model
    - Hierarchical organisation
    - Interpackage dependencies

  - Multiple meanings
    - Represent phases/activities of the development process
      - Analysis model, design model, test model...
    - Represent parts of the system
      - User interface, data base access, libraries...
    - Mere storage for related model elements...

Model Elements

Examples of “things” (3)

- Notes
  - Explanatory parts, comments
  - May be attached to any modeling element
  - Complement the model
  - Free format, pseudo-code, constraint language, formal notations...

  Product Definition Model
  This is a package

Model Elements

Relationships (4)

- Dependency
  - A semantic relationship between two things
  - When the target changes, the source might have to change

- Realisation
  - Semantic relationship between classifiers wherein one thing specifies a contract that the other guarantees to carry out

- Association
  - A structural relationship describing a set of links
  - Some special cases (aggregation, composition)

- Generalisation
  - Objects of the specialised thing are substitutable for objects of the more general one
**Model Elements**

**Relationships: Dependency**

- A semantic relationship between two things
- When the target changes, the source might have to change
- Generally unidirectional
- *Does not imply a static (permanent) kind of relationship*

**Diagram Examples**

- **Static diagrams**
  - Class diagram
  - Object diagram
  - Use case diagram
  - Sequence diagram
  - Collaboration diagram
  - Statechart diagram
  - Activity diagram
  - Component diagram
  - Deployment diagram

- **Dynamic diagrams**
  - Static (implementation) diagrams

**Rules**

- **Well-formed model**
  - Rules for
    - Names
    - Scopes
    - Visibility
    - Integrity
    - Execution

- **Badly-formed models**
  - Elided
  - Incomplete
  - Inconsistent

- **Badly-formed models cannot be avoided during the project life-cycle**

**Common Mechanisms**

- **Specifications**
  - (Usually textual-like) representations stating the exact details of a model element (its properties)

- **Adornments**
  - Graphic or textual adornments supplying extra semantics

- **Extensibility mechanisms**
  - Stereotypes
  - Tagged values
  - Constraints
Common Mechanisms
Specifications

A stereotype can be attached to any model element
- UML 1.": No more than one stereotype per element
- It qualifies (particularises) a model element semantics
- Predefined and user-defined stereotypes

Common Mechanisms
Stereotypes

Tagged values and properties
- Constraints

- {pre: age > 18 and single()}

- move()
- marry()
- single()
A UML Model

- A set of Model Elements
  - Hierarchical organisation owing to packages
  - (Partially) visible through diagrams
  - Consistent and complete
  - Manipulable by tools, verifiable, modifiable...
  - Exchangeable (XML)

- A UML model is not reducible to a set of diagrams

UML Diagrams

<table>
<thead>
<tr>
<th>Class</th>
<th>Architecture, structure</th>
<th>Analysis and Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>Architecture of a particular instance of the model</td>
<td>Analysis, Design, Test...</td>
</tr>
<tr>
<td>Use Case</td>
<td>Application needs</td>
<td>Requirement modeling</td>
</tr>
<tr>
<td>Sequence</td>
<td>Scenario of use, exchange of messages between objects</td>
<td>Requirement modeling, Design, Test</td>
</tr>
<tr>
<td>Collaboration</td>
<td>Cooperation between objects, Design Patterns</td>
<td>Requirement modeling, Design, Test</td>
</tr>
<tr>
<td>Statechart</td>
<td>Dynamic behavior</td>
<td>Design, Test, Simulation</td>
</tr>
<tr>
<td>Activity</td>
<td>Business procedures</td>
<td>Requirement modeling, Design, Test, Configuration Management</td>
</tr>
<tr>
<td>Component</td>
<td>Software resource organization</td>
<td>Design, Test, Configuration Management</td>
</tr>
<tr>
<td>Deployment</td>
<td>Run-time organization, distribution...</td>
<td>Requirement modeling, Design...</td>
</tr>
</tbody>
</table>
Synopsis of Notations

Use Case Diagrams

- Place phone call
- Place conference call
- Receive phone call
- Receive additional call
- Use scheduler

Cellular network
actor
User
association
System boundary
Cellular Telephone

Synopsis of Notations

Class Diagrams

- Company
- Department
- Person
- ContactInformation

Association

Synopsis of Notations

Object Diagrams

- d1 : Department
  name = "Sales"
- d2 : Department
  name = "R&D"
- d3 : Department
  name = "US Sales"
- p : Person
  name = "Erin"
  employeeID = 4362
  title = "VP of Sales"
  address = "1472 Miller St."

Synopsis of Notations

Sequence Diagrams

- thread : Thread
- toolkit : Toolkit
- a1 : run(3)
- run()
- call()
- create()
- handleException()
- destroy()
Synopsis of Notations
Activity Diagrams

Synopsis of Notations
State-Transition Diagrams

Synopsis of Notations
Component Diagrams

Synopsis of Notations
Deployment Diagrams
References

The Object-Oriented Approach
Methodology

- *The Unified Software Development Process*
  Ivar Jacobson, Grady Booch, James Rumbaugh
  Addison Wesley, 1999

- *Object Solutions: Managing the Object-oriented Project*
  Grady Booch
  Addison Wesley, 1996

- *Surviving Object-oriented Projects*
  Alistair Cockburn
  Addison Wesley, 1997

The Unified Modeling Language
Tutorial Books

- *The Unified Modeling Language User Guide*
  Grady Booch, James Rumbaugh, Ivar Jacobson
  Addison Wesley, 1999.

- *The Unified Modeling Language Reference Guide*
  James Rumbaugh, Grady Booch, Ivar Jacobson
  Addison Wesley, 1999

- *UML in a Nutshell*
  Sinni S Alhiri
  O'Reilly, 1998
The Unified Modeling Language
Using the UML

- Visual Modeling with Rational Rose 2000 and UML
  Terry Quatrani, Addison Wesley, 2000
- Writing Efficient Use Cases
  Alistair Cockburn, Addison Wesley, 2001
- Applying Use Cases: A Practical Guide
  Geri Schneider, Jason P. Winters
  Addison Wesley, 1998
- UML Distilled
  Martin Fowler (with Kendall Scott)
  Addison Wesley, 1997

The Unified Modeling Language
Official References

- OMG Unified Modeling Language Specification
  OBJECT MANAGEMENT GROUP, INC.
  Version 1.5, 2003

- URL's:
  - http://www.omg.com

Object-Oriented Approach & UML
Miscellaneous

- The Object Constraint Language
  Jos Warmer, Anneke Kleppe
  Addison Wesley, 1999
- Design Patterns: Elements of Reusable Object-oriented Software
  Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
  Addison Wesley, 1995
- Applying UML and Patterns,
  Craig Larman
- Model Driven Architecture
  - David S. Frankel
    Wiley, 2003

OMG Unified Modeling Language Specification
OBJECT MANAGEMENT GROUP, INC.
Version 1.5, 2003