Introduction to UML: Structural and Use Case Modeling

Cris Kobryn
Co-Chair UML Revision Task Force

cris.kobryn@telelogic.com

Overview

- Tutorial series
- Quick tour
- Structural modeling
- Use case modeling
Tutorial Series

- Lecture 1: Introduction to UML: Structural and Use Case Modeling
- Lecture 2: Behavioral Modeling with UML
- Lecture 3: Advanced Modeling with UML

[Note: This version of the tutorial series is based on OMG UML Specification v. 1.4, OMG doc# ad/01-02-13, adopted in May 2001.]
Tutorial Goals

What you will learn:
- what the UML is and what is it not
- UML’s basic constructs, rules and diagram techniques
- how the UML can model large, complex systems
- how the UML can specify systems in an implementation-independent manner

What you will not learn:
- object methods or processes
- metamodeling techniques
Quick Tour

- Why do we model?
- What is the UML?
- Foundation elements
- Unifying concepts
- Language architecture
- Relation to other OMG technologies
Why do we model?

- Provide structure for problem solving
- Experiment to explore multiple solutions
- Furnish abstractions to manage complexity
- Reduce time-to-market for business problem solutions
- Decrease development costs
- Manage the risk of mistakes
The Challenge

Tijuana “shantytown”:
http://www.macalester.edu/~jschatz/residential.html
The Vision

Fallingwater:
http://www.adelaide.net.au/~jpolias/FLW/Images/FallingWater.jpeg
Why do we model graphically?

- Graphics reveal data.
  - Edward Tufte
    - *The Visual Display of Quantitative Information, 1983*

- 1 bitmap = 1 megaword.
  - Anonymous visual modeler
Quick Tour

- The UML is a graphical language for
  - specifying
  - visualizing
  - constructing
  - documenting
  the artifacts of software systems

- Added to the list of OMG adopted technologies in November 1997 as UML 1.1

- Most recent minor revision is UML 1.4, adopted in May 2001.

- Next major revision will be UML 2.0, planned to be completed in 2002
UML Goals

- Define an easy-to-learn but semantically rich visual modeling language
- Unify the Booch, OMT, and Objectory modeling languages
- Include ideas from other modeling languages
- Incorporate industry best practices
- Address contemporary software development issues
  - scale, distribution, concurrency, executability, etc.
- Provide flexibility for applying different processes
- Enable model interchange and define repository interfaces
Introduction to UML 12

OMG UML Evolution

Updated from [Kobryn 01a].
<table>
<thead>
<tr>
<th>OMG UML Contributors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aonix</td>
</tr>
<tr>
<td>Colorado State University</td>
</tr>
<tr>
<td>Computer Associates</td>
</tr>
<tr>
<td>Concept Five</td>
</tr>
<tr>
<td>Data Access</td>
</tr>
<tr>
<td>EDS</td>
</tr>
<tr>
<td>Enea Data</td>
</tr>
<tr>
<td>Hewlett-Packard</td>
</tr>
<tr>
<td>IBM</td>
</tr>
<tr>
<td>I-Logix</td>
</tr>
<tr>
<td>InLine Software</td>
</tr>
<tr>
<td>Intelicorp</td>
</tr>
<tr>
<td>Kabira Technologies</td>
</tr>
<tr>
<td>Klasse Objecten</td>
</tr>
<tr>
<td>Lockheed Martin</td>
</tr>
<tr>
<td>Microsoft</td>
</tr>
<tr>
<td>ObjecTime</td>
</tr>
<tr>
<td>Oracle</td>
</tr>
<tr>
<td>Ptech</td>
</tr>
<tr>
<td>OAO Technology Solutions</td>
</tr>
<tr>
<td>Rational Software</td>
</tr>
<tr>
<td>Reich</td>
</tr>
<tr>
<td>SAP</td>
</tr>
<tr>
<td>Softeam</td>
</tr>
<tr>
<td>Sterling Software</td>
</tr>
<tr>
<td>Sun</td>
</tr>
<tr>
<td>Taskon</td>
</tr>
<tr>
<td>Telelogic</td>
</tr>
<tr>
<td>Unisys</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
OMG UML 1.4 Specification

- UML Summary
- UML Semantics
- UML Notation Guide
- UML Example Profiles
  - Software Development Processes
  - Business Modeling
- Model Interchange
  - Model Interchange Using XMI
  - Model Interchange Using CORBA IDL
- Object Constraint Language
Tutorial Focus: the Language

- language = syntax + semantics
  - syntax = rules by which language elements (e.g., words) are assembled into expressions (e.g., phrases, clauses)
  - semantics = rules by which syntactic expressions are assigned meanings

- UML Notation Guide – defines UML’s graphic syntax
- UML Semantics – defines UML’s semantics
Foundation Concepts

- Building blocks
- Well-formedness rules
Building Blocks

The basic building blocks of UML are:

- model elements (classes, interfaces, components, use cases, etc.)
- relationships (associations, generalization, dependencies, etc.)
- diagrams (class diagrams, use case diagrams, interaction diagrams, etc.)

Simple building blocks are used to create large, complex structures

- cf. elements, bonds and molecules in chemistry
- cf. components, connectors and circuit boards in hardware
Diagram: Classifier View

Element

Carbon

<<covalent>>

C

<<covalent>>

C

Hydrogen

<<covalent>>

C

H

Introduction to UML
Diagram: Instance View

```
:Hydrogen -:Carbon -:Carbon -:Hydrogen

:Hydrogen -:Hydrogen

:Hydrogen -:Hydrogen
```

Introduction to UML
Well-Formedness Rules

- Well-formed: indicates that a model or model fragment adheres to all semantic and syntactic rules that apply to it.

- UML specifies rules for:
  - naming
  - scoping
  - visibility
  - integrity
  - execution (limited)

- However, during iterative, incremental development it is expected that models will be incomplete and inconsistent.
Example of semantic rule: Class [1]

**English:** If a Class is concrete, all the Operations of the Class should have a realizing Method in the full descriptor.

**OCL:** not self.isAbstract implies self.allOperations->forall (op | self.allMethods->exists (m | m.specification->includes(op)))
Well-Formedness Rules (cont’d)

Example of syntactic rules: Class

- **Basic Notation:** A class is drawn as a solid-outline rectangle with three compartments separated by horizontal lines.

- **Presentation Option:** Either or both of the attribute and operation compartments may be suppressed.

Example of syntactic guideline: Class

- **Style Guideline:** Begin class names with an uppercase letter.
Unifying Concepts

- classifier-instance dichotomy
  - e.g., an object is an instance of a class OR a class is the classifier of an object

- specification-realization dichotomy
  - e.g., an interface is a specification of a class OR a class is a realization of an interface

- analysis-time vs. design-time vs. run-time
  - modeling phases ("process creep")
  - usage guidelines suggested, not enforced
Language Architecture

- Metamodel architecture
- Package structure
The attribute fare of the PassengerTicket class is an instance of the metaclass Attribute.

The operation issue of the PassengerTicket class is an instance of the metaclass Operation.

Represents the User Object layer of the 4-layer metamodel architecture pattern.

From [Kobryn 01b].

The attribute fare of the PassengerTicket class is an instance of the metaclass Attribute.
From [Kobryn 01b].
Relationships to Other Modeling Technologies

From [Kobryn 01b].
Structural Modeling

- What is structural modeling?
- Core concepts
- Diagram tour
- When to model structure
- Modeling tips
- Example: Interface-based design
What is structural modeling?

- Structural model: a view of a system that emphasizes the structure of the objects, including their classifiers, relationships, attributes and operations.
# Structural Modeling: Core Elements

<table>
<thead>
<tr>
<th>Construct</th>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>a description of a set of objects that share the same attributes, operations, methods, relationships and semantics.</td>
<td></td>
</tr>
<tr>
<td><strong>interface</strong></td>
<td>a named set of operations that characterize the behavior of an element.</td>
<td>![interface symbol]</td>
</tr>
<tr>
<td><strong>component</strong></td>
<td>a modular, replaceable and significant part of a system that packages implementation and exposes a set of interfaces.</td>
<td>![component symbol]</td>
</tr>
<tr>
<td><strong>node</strong></td>
<td>a run-time physical object that represents a computational resource.</td>
<td>![node symbol]</td>
</tr>
<tr>
<td>Construct</td>
<td>Description</td>
<td>Syntax</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>--------</td>
</tr>
<tr>
<td>constraint¹</td>
<td>a semantic condition or restriction.</td>
<td>{constraint}</td>
</tr>
</tbody>
</table>

¹ An extension mechanism useful for specifying structural elements.
### Structural Modeling: Core Relationships

<table>
<thead>
<tr>
<th>Construct</th>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>association</strong></td>
<td>A relationship between two or more classifiers that involves connections among their instances.</td>
<td></td>
</tr>
<tr>
<td><strong>aggregation</strong></td>
<td>A special form of association that specifies a whole-part relationship between the aggregate (whole) and the component part.</td>
<td>←</td>
</tr>
<tr>
<td><strong>generalization</strong></td>
<td>A taxonomic relationship between a more general and a more specific element.</td>
<td></td>
</tr>
<tr>
<td><strong>dependency</strong></td>
<td>A relationship between two modeling elements, in which a change to one modeling element (the independent element) will affect the other modeling element (the dependent element).</td>
<td>←</td>
</tr>
</tbody>
</table>
### Structural Modeling: Core Relationships (cont’d)

<table>
<thead>
<tr>
<th>Construct</th>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>realization</td>
<td>a relationship between a specification and its implementation.</td>
<td></td>
</tr>
</tbody>
</table>
Structural Diagram Tour

- Show the static structure of the model
  - the entities that exist (e.g., classes, interfaces, components, nodes)
  - internal structure
  - relationship to other entities

- Do not show
  - temporal information

- Kinds
  - static structural diagrams
    - class diagram
    - object diagram
  - implementation diagrams
    - component diagram
    - deployment diagram
Static Structural Diagrams

- Shows a graph of classifier elements connected by static relationships.

- kinds
  - class diagram: classifier view
  - object diagram: instance view
**Window**

{abstract,
 author=Joe,
 status=tested}

+size: Area = (100,100)
#visibility: Boolean = true
+default-size: Rectangle
#maximum-size: Rectangle
-xptr: XWindow*

size: Area
visibility: Boolean

display ()
hide ()

-attachXWindow(xwin:Xwindow*)

**Fig. 3-20, UML Notation Guide**
Classes: compartments with names

Reservation

operations

guarantee()
cancel ()
change (newDate: Date)

responsibilities

bill no-shows
match to available rooms

exceptions

invalid credit card

Fig. 3-23, UML Notation Guide
Classes: method body

PoliceStation

alert (Alarm)

1 station

*

BurglarAlarm

isTripped: Boolean = false

report ()

{ if isTripped
  then station.alert(self) }
Fig. 3-27, *UML Notation Guide*
Interfaces: Shorthand Notation

Fig. 3-29, UML Notation Guide
Interfaces: Longhand Notation

Fig. 3-29, *UML Notation Guide*
Associations

Company * Job 1..* Person
  employer employee
  |
  Job
  salary boss 0..1
  worker *

Manages

Person

Account {Xor}

Corporation

Fig. 3-40, UML Notation Guide
Association Ends

Polygon

1

Contains

Point

3..*

{ordered}

GraphicsBundle

1

+vertex

color
texture
density

-bundle

Fig. 3-41, UML Notation Guide
Ternary Associations

Fig. 3-44, *UML Notation Guide*

Introduction to UML
Composition

Window

scrollbar [2]: Slider
  title: Header
  body: Panel

Fig. 3-45, UML Notation Guide
Composition (cont’d)

Window

scrollbar:Slider 2

title:Header 1

body:Panel 1

Fig. 3-45, UML Notation Guide
Generalization

Shape

Polygon  Ellipse  Spline  . . .

Separate Target Style

Shape

 Polygon  Ellipse  Spline  . . .

Shared Target Style

Fig. 3-47, UML Notation Guide

Introduction to UML
Generalization

Fig. 3-48, *UML Notation Guide*

Introduction to UML
Fig. 3-50, *UML Notation Guide*

ClassA  «friend»  ClassB

«call»  «instantiate»

ClassC  «refine»

ClassC combines two logical classes

ClassD  «friend»  operationZ()

ClassE
Dependencies

Fig. 3-51, *UML Notation Guide*
Derived Attributes and Associations

\{ \text{age} = \text{currentDate} - \text{birthdate} \} \quad \quad \text{birthdate} \\
\quad \quad \quad \quad \quad \quad \quad \text{/age}

\text{Company} \quad 1 \quad \text{Department} \quad * \\
\quad \quad \text{employer} \quad 1 \quad \text{department} \\
\quad \quad \quad \text{employer} \quad * \quad \text{WorksForDepartment} \\
\quad \quad \quad \quad \quad * \quad \text{Person} \\
\quad \quad \quad \text{/WorksForCompany} \\
\quad \quad \quad \quad \quad \{ \text{Person.employer} = \text{Person.department.employer} \} \\

\text{Fig. 3-52, UML Notation Guide}
triangle : Polygon

center = (0, 0)
vertices = ((0, 0), (4, 0), (4, 3))
borderColor = black
fillColor = white

triangle : Polygon

scheduler

Fig. 3-38, UML Notation Guide
Composite objects

awindow : Window

horizontalBar:ScrollBar

verticalBar:ScrollBar

moves

surface:Pane moves

title:TitleBar

Fig. 3-39, UML Notation Guide
Fig. 3-46, *UML Notation Guide*
Constraints and Comments

Fig. 3-17, *UML Notation Guide*
Class Diagram Example

Adapted from Fig. 23 [EJB 2.0].
Implementation Diagrams

- Show aspects of model implementation, including source code structure and run-time implementation structure

- Kinds
  - component diagram
  - deployment diagram
Component Diagram

- Shows the organizations and dependencies among software components

- Components may be
  - specified by classifiers (e.g., implementation classes)
  - implemented by artifacts (e.g., binary, executable, or script files)
Fig. 3-99, *UML Notation Guide (corrected)*

Introduction to UML
Fig. 3-95, *UML Notation Guide*
Fig. 3-96, *UML Notation Guide*
Deployment Diagram

- Shows the configuration of run-time processing elements and the software components, processes and objects that live on them
- Deployment diagrams may be used to show which components may run on which nodes
Fig. 3-97, *UML Notation Guide*
Introduction to UML

Deployment Diagram (2/2)

Fig. 3-98, UML Notation Guide

Introduction to UML
When to model structure

- Adopt an opportunistic top-down+bottom-up approach to modeling structure
  - Specify the top-level structure using “architecturally significant” classifiers and model management constructs (packages, models, subsystems; see Tutorial 3)
  - Specify lower-level structure as you discover detail re classifiers and relationships
- If you understand your domain well you can frequently start with structural modeling; otherwise
  - If you start with use case modeling (as with a use-case driven method) make sure that your structural model is consistent with your use cases
  - If you start with role modeling (as with a collaboration-driven method) make sure that your structural model is consistent with your collaborations
Structural Modeling Tips

- Define a “skeleton” (or “backbone”) that can be extended and refined as you learn more about your domain.
- Focus on using basic constructs well; add advanced constructs and/or notation only as required.
- Defer implementation concerns until late in the modeling process.
- Structural diagrams should
  - emphasize a particular aspect of the structural model
  - contain classifiers at the same level of abstraction
- Large numbers of classifiers should be organized into packages (see Lecture 3)
Example: Point-of-Sale

The following example shows how UML can model the interfaces for a Point of Sale application originally specified in CORBA IDL. From [Kobryn 01b].
module POS
{
    typedef long    POSId;
    typedef string  Barcode;

    interface InputMedia
    {
    typedef string OperatorCmd;
    void        BarcodeInput(in Barcode Item);
    void        KeypadInput(in OperatorCmd Cmd);
    
    
    interface OutputMedia
    {...};
    interface POSTerminal
    {...};
};

Ch. 26, CORBA Fundamentals and Programming (2nd ed.), [Siegel 00]
Introduction to UML

From [Kobryn 2001b]
From [Kobryn 2001b]
Model Fragment from POS Example

From [Kobryn 2001b]
XML Generated by XMI Facility

From [Kobryn 2001b]

```xml
<XMI xmlns:xmi='http://www.omg.org/UML/1.3'>
  <XMI.header>
    <XMI.metamodel xmi.name='UML' xmi.version='1.3'/>
  </XMI.header>
  <XMI.content>
    <!-- POS_Example_R2 [Model] -->
    <UML:Model xmi.id='G.0'
      name='POS_Example_R2' visibility='public' isSpecification='false'
      isRoot='false' isLeaf='true' isAbstract='false' >
      <UML:Namespace.ownedElement>
        <!-- POS_Example_R2::Tax [Class] -->
        <UML:Class xmi.id='S.1'
          name='Tax' visibility='public' isSpecification='false'
          isRoot='true' isLeaf='true' isAbstract='false'
          isActive='true'
          namespace='G.0' clientDependency='G.1' />
    </UML:Model>
  </XMI.content>
</XMI>
```

...
Use Case Modeling

- What is use case modeling?
- Core concepts
- Diagram tour
- When to model use cases
- Modeling tips
- Example: Online HR System
What is use case modeling?

- use case model: a view of a system that emphasizes the behavior as it appears to outside users. A use case model partitions system functionality into transactions (‘use cases’) that are meaningful to users (‘actors’).
# Use Cases: Core Elements

<table>
<thead>
<tr>
<th>Construct</th>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>use case</strong></td>
<td>A sequence of actions, including variants, that a system (or other entity) can perform, interacting with actors of the system.</td>
<td><img src="image" alt="UseCaseName" /></td>
</tr>
<tr>
<td><strong>actor</strong></td>
<td>A coherent set of roles that users of use cases play when interacting with these use cases.</td>
<td><img src="image" alt="ActorName" /></td>
</tr>
<tr>
<td><strong>system boundary</strong></td>
<td>Represents the boundary between the physical system and the actors who interact with the physical system.</td>
<td><img src="image" alt="system boundary" /></td>
</tr>
</tbody>
</table>
**Use Cases: Core Relationships**

<table>
<thead>
<tr>
<th>Construct</th>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>association</td>
<td>The participation of an actor in a use case. i.e., instance of an actor and instances of a use case communicate with each other.</td>
<td></td>
</tr>
<tr>
<td>generalization</td>
<td>A taxonomic relationship between a more general use case and a more specific use case.</td>
<td></td>
</tr>
<tr>
<td>include</td>
<td>a relationship from a <em>base</em> use case to an <em>inclusion</em> use case, specifying how the behavior for the base use case contains the behavior defined for the inclusion use case. <em>The base use case depends on the inclusion use case.</em> Compare: extend.</td>
<td>&lt;&lt;include&gt;&gt;</td>
</tr>
</tbody>
</table>

**Introduction to UML**
### Use Cases: Core Relationships (cont’d)

<table>
<thead>
<tr>
<th>Construct</th>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>extend</td>
<td>A relationship from an <em>extension</em> use case to a <em>base</em> use case, specifying how the behavior for the extension use case augments (subject to conditions in the extension) the behavior defined for the base use case. <em>The base use case does not depend on the extension use case.</em> Compare: include.</td>
<td>&lt;&lt;extend&gt;&gt;</td>
</tr>
</tbody>
</table>

---

*Introduction to UML* 77
Use Case Diagram Tour

- Shows use cases, actors and their relationships
- Use case internals can be specified by text and/or interaction diagrams (see Lecture 2)
- Kinds
  - use case diagram
  - use case description
Use Case Diagram

Fig. 3-53, *UML Notation Guide*

Introduction to UML
Use Case Relationships

Fig. 3-54, *UML Notation Guide*

Introduction to UML 80
Actor Relationships

Salesperson

1

Place
Order

1 *

Establish
Credit

1 *

Supervisor

Fig. 3-55, UML Notation Guide
Use Case Description: Change Flight

■ **Actors**: traveler, client account db, airline reservation system

■ **Preconditions**:
  - Traveler has logged on to the system and selected ‘change flight itinerary’ option

■ **Basic course**
  - System retrieves traveler’s account and flight itinerary from client account database
  - System asks traveler to select itinerary segment she wants to change; traveler selects itinerary segment.
  - System asks traveler for new departure and destination information; traveler provides information.
  - If flights are available then
  - ...
  - System displays transaction summary.

■ **Alternative courses**
  - If no flights are available then …
When to model use cases

- Model user requirements with use cases.
- Model test scenarios with use cases.
- If you are using a use-case driven method
  - start with use cases and derive your structural and behavioral models from it.
- If you are not using a use-case driven method
  - make sure that your use cases are consistent with your structural and behavioral models.
Use Case Modeling Tips

- Make sure that each use case describes a significant chunk of system usage that is understandable by both domain experts and programmers.
- When defining use cases in text, use nouns and verbs accurately and consistently to help derive objects and messages for interaction diagrams (see Lecture 2).
- Factor out common usages that are required by multiple use cases:
  - If the usage is required use «include»
  - If the base use case is complete and the usage may be optional, consider use «extend»
- A use case diagram should:
  - Contain only use cases at the same level of abstraction
  - Include only actors who are required
- Large numbers of use cases should be organized into packages (see Lecture 3)
Example: Online HR System

Online HR System

- Locate Employees
- Update Employee Profile
- Update Benefits
- Access Travel System
- Access Pay Records

{if currentMonth = Oct.}

{readOnly}

Employee

Manager

Healthcare Plan System

Insurance Plan System
Online HR System: Use Case Relationships

Update Medical Plan

Update Dental Plan

Update Insurance Plan

Extension points
benefit options:
after required enrollments

Employee

<<include>>

<<include>>

<<include>>

Update Benefits

<<extend>>
employee requests
reimbursement option

<<extend>>
employee requests
stock purchase option

Elect Reimbursement for Healthcare

Elect Stock Purchase

Extension condition
name and location

Elect Reimbursement for Healthcare

Employee

<<include>>

<<include>>

<<include>>
Online HR System: Update Benefits Use Case

**Actors:** employee, employee account db, healthcare plan system, insurance plan system

**Preconditions:**
- Employee has logged on to the system and selected ‘update benefits’ option

**Basic course**
- System retrieves employee account from employee account db
- System asks employee to select medical plan type; **include** Update Medical Plan.
- System asks employee to select dental plan type; **include** Update Dental Plan.
- ...

**Alternative courses**
- If health plan is not available in the employee’s area the employee is informed and asked to select another plan...
Wrap Up

- Ideas to take away
- Preview of next tutorial
- References
- Further info
Ideas to Take Away

- UML is effective for modeling large, complex software systems.
- It is simple to learn for most developers, but provides advanced features for expert analysts, designers, and architects.
- It can specify systems in an implementation-independent manner.
- 10-20% of the constructs are used 80-90% of the time.
- Structural modeling specifies a skeleton that can be refined and extended with additional structure and behavior.
- Use case modeling specifies the functional requirements of systems in an object-oriented manner.
Behavioral Modeling with UML

- Behavioral modeling overview
- Interactions
- Collaborations
- Statecharts
- Activity graphs
References


Further Info

Web:
- UML 1.4 RTF: [www.celigent.com/omg/umlrdf](http://www.celigent.com/omg/umlrdf)

Email
- uml-rtf@omg.org
- Cris.Kobryn@telelogic.com

Conferences & workshops