Introduction to UML: Structural Modeling and Use Cases

Cris Kobryn
Co-Chair UML Revision Task Force
November 2000
Overview

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

- Lecture 1: Introduction to UML: Structural Modeling and Use Cases
- Lecture 2: Behavioral Modeling with UML
- Lecture 3: Advanced Modeling with UML
- Lecture 4: Metadata Integration with UML, MOF and XMI
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
- how UML, XMI and MOF can facilitate metadata integration

What you will not learn:

- Object Modeling 101
- object methods or processes
- Metamodeling 101
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
The UML is a graphical language for specifying, visualizing, constructing, and 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 1.3, adopted in November 1999.
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
OMG UML Evolution

2002
(planned major revision)

2001
(planned minor revision)

Q4 2000
(planned minor revision)

1999

1998

1997
(adopted by OMG)

UML 1.1
(1997, adopted by OMG)

UML 1.2
(1998)

UML 1.3
(Q4 2000, planned minor revision)

UML 1.4
(2001, planned minor revision)

UML 1.5
(2002, planned major revision)

UML 2.0
(2002, planned major revision)

ISO Publicly Available Specification
[read only]

Editorial revision with no significant technical changes.

The expected result of OMG’s formal liaison with ISO.

[backward compatible]
<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 1.3 Specification

- UML Summary
- UML Semantics
- UML Notation Guide
- UML Standard Profiles
  - Software Development Processes
  - Business Modeling
- UML CORBAfacility Interface Definition
- UML XML Metadata Interchange DTD
- 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
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

Hydrogen

<<covalent>>

C

<<covalent>>

C

C

<<covalent>>

C

H
Diagram: Instance View

- Carbon
  - Hydrogen
  - Hydrogen

- Hydrogen
  - Carbon
  - Carbon

- 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:**
  
  \[
  \text{not self.isAbstract } \implies \text{self.allOperations->forall } (\text{op } | \text{self.allMethods->exists } (\text{m } | \text{m.specification->includes(op)}))
  \]
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
Metamodel Architecture

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.

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

From *Modeling CORBA Applications with UML* chapter in [Siegel 00].
Introduction to UML
Relation to Other OMG Technologies

OMG UML

- Meta Object Facility
- XML Metadata Interchange (XMI) Facility
- UML XML Document Type Definition
- UML CORBA Facility Interface Definition

OMG UML Layers:

- **Metadata Layer**
  - XML Metadata Interchange (XMI) Facility
- **Specification Layer**
  - OMG UML
  - UML Profile for CORBA
  - UML Profiles for Business Domains
- **Customization Layer**
  - UML Profiles for CORBA
  - UML Profiles for Business Domains

*** In process, not yet adopted
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.
<table>
<thead>
<tr>
<th>Construct</th>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>a description of a set of objects that share the same attributes, operations, methods, relationships and semantics.</td>
<td>![class symbol]</td>
</tr>
<tr>
<td>interface</td>
<td>a named set of operations that characterize the behavior of an element.</td>
<td>![interface symbol]</td>
</tr>
<tr>
<td>component</td>
<td>a physical, replaceable part of a system that packages implementation and provides the realization of a set of interfaces.</td>
<td>![component symbol]</td>
</tr>
<tr>
<td>node</td>
<td>a run-time physical object that represents a computational resource.</td>
<td>![node symbol]</td>
</tr>
</tbody>
</table>
### Structural Modeling: Core Elements (cont’d)

<table>
<thead>
<tr>
<th>Construct</th>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint¹</td>
<td>a semantic condition or restriction.</td>
<td></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>
<tr>
<td>Construct</td>
<td>Description</td>
<td>Syntax</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>realization</td>
<td>a relationship between a specification and its implementation.</td>
<td></td>
</tr>
</tbody>
</table>
Introduction to UML

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
Fig. 3-17, *UML Notation Guide*
Classes: compartments with names

Reservation

operations

guarantee()
cancel ()

deriresibilities

bill no-shows
match to available rooms

exceptions

invalid credit card

Fig. 3-20, *UML Notation Guide*
**Classes: method body**

*PoliceStation*

alert ()

1 station

* * *

*BurglarAlarm*

isTripped: Boolean = false

report () { if isTripped then station.alert(self)}

---

Fig. 3-21, *UML Notation Guide*
Interfaces

Fig. 3-24, *UML Notation Guide*
Associations

Fig. 3-31, *UML Notation Guide*

Introduction to UML
Association Ends

Polygon

\[ \text{Contains} \]

Point

\{ordered\}

GraphicsBundle

-\text{bundle}

\begin{align*}
\text{color} & \\
\text{texture} & \\
\text{density} & 
\end{align*}

\[ +\text{points} \]

\[ 3..* \]

Fig. 3-32, *UML Notation Guide*
Ternary Associations

```
Year
    season *

Team
    *           *
    team       goalkeeper

Player

Record
goals for
goals against
wins
losses
ties

Fig. 3-31, UML Notation Guide
```
Composition

Window

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

Window

scrollbar 2

Slider

header 1
body 1

Fig. 3-36, *UML Notation Guide*
Composition

Window

scrollbar:Slider

1

title:Header

1

body:Panel

2

Fig. 3-36, UML Notation Guide
Generalization

Shape

Separate Target Style

Polygon  Ellipse  Spline  . . .

Shape

Shared Target Style

Polygon  Ellipse  Spline  . . .

Fig. 3-38, UML Notation Guide
Generalization

Vehicle

power

{overlapping}

power

venue

{overlapping}

WindPowered

Vehicle

MotorPowered

Vehicle

Land

Vehicle

Water

Vehicle

Truck

Sailboat

Fig. 3-39, UML Notation Guide
Fig. 3-41, *UML Notation Guide*
Fig. 3-42, *UML Notation Guide*
Objects

Fig. 3-29, *UML Notation Guide*

* scheduler

triang le : P olygon

c e n t e r = (0,0)
ve r t i c e s = ((0,0), (4,0), (4,3))
bo r d e r C o l o r = b l a c k
fi ll C o l o r = w h i t e

triang le : P olygon

triang le : P olygon

triang le

Introduction to UML
Composite objects

awindow : Window

horizontalBar:ScrollBar

verticalBar:ScrollBar

moves

surface:Pane moves

title:TitleBar

Fig. 3-30, UML Notation Guide
downhillSkiClub:Club
  ↓
  ↓
  ↓
Joe:Person
Chris:Person
Jill:Person
  ↓
  ↓
  ↓
member
member
member
officer
officer
officer
treasurer
president
member

Fig. 3-37, UML Notation Guide
Constraints and Comments

* Member-of *

Person

| {subset} |

Committee

Represents an incorporated entity.

1 Chair-of *

Person

| | | | | | |

Person

| | | | | | |

Person

| | | | | | |

Person

| | | | | | |

Person

| | | | | | |

Person

| | | | | | |

Person

| | | | | | |

Person

| | | | | | |

Person

| | | | | | |

Person

| | | | | | |

Person

| | | | | | |

Person

| | | | | | |

Person

| | | | | | |

Person

| | | | | | |

Person

| | | | | | |

Person

| | | | | | |

Figure 3-15, UML Notation Guide

Introduction to UML 52
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 include
  - source code components
  - binary code components
  - executable components
mymailer : Mailer
          +Mailbox  +RoutingList
            -MailQueue

Fig. 3-84, *UML Notation Guide*
Component Diagram

Scheduler

Reservations

Planner

Update

GUI

Fig. 3-81, 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. 3-82, *UML Notation Guide*
Deployment Diagram (cont’d)

Node1

«cluster»

x y

«become»

Node2

«cluster»

x y

«database»

w z

Fig. 3-83, UML Notation Guide
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: Interface-based design

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]
From *Modeling CORBA Applications with UML* chapter
in [Siegel 00].

**Point-of-Sale**

```
«IDLinterface»
IPOSTerminal
+storeRef : Store
+storeAccessRef : StoreAccess
+outputMediaRef : OutputMedia
+taxRef : Tax
+POSid : Integer
+itemBarcode : Integer
+itemQuantity : Integer
+itemInfo : ItemInfo
+itemPrice : Currency
+itemTaxPrice : Currency
+itemExtension : Currency
+saleSubtotal : Currency
+taxableSubtotal : Currency
+saleTotal : Currency
+saleTax : Currency
+POSlist : List
+initialization()
+login()
+printPOSsalesSummary()
+printStoreSalesSummary()
+setItemQuantity()
+sendBarcode()
+endSale()
```

```
«IDLinterface»
IStore
+totals : Totals
+POSlist : List
+initialization()
+login()
+getPOStotals()
+updateStoreTotals()
```

```
«IDLinterface»
ITax
+rate : float
+initialization()
+calculateTax()
+findTaxablePrice()
```

```
«IDLinterface»
IOutputMedia
+outputText()
```

```
«IDLinterface»
IStoreAccess
+depotRef : Depot
+taxRef : Tax
+storeMarkup : float
+storeId : Integer
+initialization()
+findPrice()
```

```
«IDLinterface»
OutputMedia
```

```
«IDLinterface»
IInputMedia
+POSref : POSTerminal
+initialization()
+barcodeInput()
+keypadInput()
```

```
POSTerminal
OutputMedia
```

```
InputMedia
```

```
Store
```

```
Tax
```

```
StoreAccess
```

```
«IDLinterface»
IPOSterminal
+initialization()
+barcodeInput()
+keypadInput()
```
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 Case Modeling: Core Elements

<table>
<thead>
<tr>
<th>Construct</th>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>use case</td>
<td>A sequence of actions, including variants, that a system (or other entity) can perform, interacting with actors of the system.</td>
<td>UseCaseName</td>
</tr>
<tr>
<td>actor</td>
<td>A coherent set of roles that users of use cases play when interacting with these use cases.</td>
<td>ActorName</td>
</tr>
<tr>
<td>system boundary</td>
<td>Represents the boundary between the physical system and the actors who interact with the physical system.</td>
<td></td>
</tr>
<tr>
<td>Construct</td>
<td>Description</td>
<td>Syntax</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<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>extend</td>
<td>A relationship from an extension use case to a base use case, specifying how the behavior for the extension use case can be inserted into the behavior defined for the base use case.</td>
<td>&lt;&lt;extend&gt;&gt;</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>
</tbody>
</table>
**Use Case Modeling: Core Relationships (cont’d)**

<table>
<thead>
<tr>
<th>Construct</th>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>include</td>
<td>An relationship from a base use case to an inclusion use case, specifying how the behavior for the inclusion use case is inserted into the behavior defined for the base use case.</td>
<td>&lt;&lt;include&gt;&gt;</td>
</tr>
</tbody>
</table>

Introduction to UML 68
Use Case Diagram Tour

- Shows use cases, actor 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

Customer

Telephone Catalog

Check status

Place order

Sales person

Fill orders

Customer

Establish credit

Shipping Clerk

Supervisor

Fig. 3-44, UML Notation Guide

Introduction to UML
Use Case Relationships

Supply Customer Data
Order Product
Arrange Payment

«include» «include» «include»

Place Order

1 *
Extension points
additional requests:
after creation of the order

«extend»
the salesperson asks for
the catalog

Request Catalog

Fig. 3-45, UML Notation Guide

Introduction to UML
Actor Relationships

Salesperson

1 * Place
1 * Order

Supervisor

1 * Establish
1 * Credit

Fig. 3-46, *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 \(<\text{include}\>)
  - If the base use case is complete and the usage may be optional, consider use \(<\text{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

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

(If currentMonth = Oct.)

Employee

Manager

Healthcare Plan System

Insurance Plan System

Introduction to UML
Online HR System: Use Case Relationships

Update Medical Plan

Update Dental Plan

Update Insurance Plan

Update Benefits

Extension points

benefit options:

after required enrollments

<<include>>

<<include>>

<<include>>

Employee

Update Benefits

<<extend>>

employee requests reimbursement option

<<extend>>

employee requests stock purchase option

Elect Reimbursement for Healthcare

Elect Stock Purchase

extension point
name and location

extension condition
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
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 a system in an object-oriented manner.
Behavioral Modeling with UML

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

- *OMG UML Specification v. 1.3*, OMG doc# ad/06-08-99


Further Info

Web:
- UML 1.4 RTF: www.celigent.com/omg/umlrtf
- OMG UML Resources: www.omg.org/uml/

Email
- uml-rtf@omg.org
- c kobryn@acm.org

Conferences & workshops
- OMG UML Workshop: UML in the .com Enterprise, Palm Springs, California, Nov. 2000
- UML World 2001, location and dates TBA