Introduction to Unified Modeling Language

Overview of architectural views and UML 2 diagrams
Contents

Introducing Modeling with UML

4+1 Architectural Views

UML 2 Diagrams
What Is UML?

• A language (notation) for modeling object-oriented systems
• A standard maintained by the Object Management Group
• A modeling language including 13 diagrams
• A means for visualizing, specifying, constructing, and documenting software systems

→ http://www.uml.org
Not all components of UML 2 are supported by modeling tools yet. Some tools still use UML 1.4.
Why do we model?
Why Do We Model?

- Furnish abstractions to manage complexity
- Provide structure for problem solving
- Experiment to explore multiple solutions

Modeling allows the following business benefits:

- Reduce time-to-market for business problem solutions
- Decrease development costs
- Manage the risk of mistakes
Why do we need a modeling language like UML?
Why Do We Need UML?

• Graphical notation
  - A picture is worth a thousand words

• Standard communication language

• Provides multiple diagrams for capturing different architectural views

• Promotes component reusability

UML is a standard language for visualizing, specifying, constructing, and documenting software systems
How can we benefit from modeling tools like MagicDraw UML?
How Can We Benefit from Using UML Modeling Tool?

- Repository of reusable model artifacts
- Visualize in multiple dimensions and levels of detail
- Use automated layout and visualization tools
- Harvest models from legacy systems
- Generate documentation from modeling environment
- Analyze traceability through relationships between elements
- Incremental development and refactoring
- Teamwork for parallel development of large systems
- Integration with other development tools
Contents

Introducing Modeling with UML

4+1 Architectural Views

UML 2 Diagrams
UML Architectural Views and Diagrams

UML defines 13 diagrams that describe 4+1 architectural views

<table>
<thead>
<tr>
<th>Structural View</th>
<th>Implementation View</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class Diagram</td>
<td>Component Diagram</td>
</tr>
<tr>
<td>Object Diagram</td>
<td>Composite Structure Diagram</td>
</tr>
<tr>
<td>Composite Structure Diagram</td>
<td></td>
</tr>
<tr>
<td>(Package Diagram)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Behavioral View</th>
<th>Environment View</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence Diagram</td>
<td>Deployment Diagram</td>
</tr>
<tr>
<td>Communication Diagram</td>
<td></td>
</tr>
<tr>
<td>State Diagram</td>
<td></td>
</tr>
<tr>
<td>Activity Diagram</td>
<td></td>
</tr>
<tr>
<td>Interaction Overview Diagram</td>
<td></td>
</tr>
<tr>
<td>Timing Diagram</td>
<td></td>
</tr>
</tbody>
</table>

Use Case View
- Use Case Diagram

4+1 architectural views model was proposed by Philippe Kruchten, IBM
Use Case View

• The most important architectural view
• Describes use cases that provide value for the users
• Essential use cases are used as proof of concept for implementation architecture
• Use cases may be visualized in UML use case diagram
• Each use case may have multiple possible scenarios
• Use case scenarios could be described:
  ▪ Using textual descriptions;
  ▪ Graphically, using UML activity diagrams.
Structural View

• Represents structural elements for implementing solution for defined requirements

• Defines
  - Object-oriented analysis and design elements;
  - Domain and solution vocabulary;
  - System decomposition into layers and subsystems;
  - Interfaces of the system and its components.

• Is represented by static UML diagrams:
  - Class diagrams in multiple abstraction levels;
  - Package diagrams;
  - Composite structure diagrams (new in UML 2).
Behavioral View

- Represents dynamic interaction between system components for implementing requirements
- Shows distribution of responsibilities
- Allows to identify interaction and coupling bottlenecks
- A means for discussing non-functional requirements
  - Performance, maintenance, …
- Is especially important for distributed systems
- Is represented by dynamic UML diagrams:
  - Sequence and/or communication diagrams;
  - Activity diagrams;
  - State diagrams;
  - Interaction overview diagram (new in UML 2);
  - Timing diagrams (new in UML 2).
Implementation View

- Describes implementation artifacts of logical subsystems defined in structural view;
- May include intermediate artifacts used in system construction (code files, libraries, data files, …)
- Defines dependencies between implementation components and their connections by required and provided interfaces

- Is represented by these UML diagrams:
  - Component diagrams;
  - Composite structure diagrams (new in UML 2).
Environment View

- Represents system’s hardware topology
- Defines how software components are deployed on hardware nodes
- Useful for analyzing non-functional requirements
  - Reliability, scalability, security, …
- Provides information for system installation and configuration

- Is represented by
  - UML deployment diagram
Contents

Introducing Modeling with UML

4+1 Architectural Views

UML 2 Diagrams
Use Case Diagram

- Describes the functionality provided by system
- Contains actors, use cases, and relationships
Class Diagram

- Describes static structure of the system
- Contains classes and relationships

```
Student
- name : String
- birthday : date
- email : String
- home : Address
+ Student( name : String )
+ setEmail( email : String )
+ getEmail() : String
+ registerForClass( class : Class )
...

Class
- code : String
- semester : String
- status : int
+ Class( course : Course, section : String, semester : String, status : int )
+ setStatus( status : int )
+ getStatus() : int
...

Graduate
+ applyForAssistance( class : Class )
...

Instructor
- name : String
- homeCampus : String
+ Instructor( name : String )
...

Student
- part : String [0..*]
+ addParty( party : String )
+ getParties() : String[]

Undergraduate
- parties : String [0..*]

Course
- title : String
- description : String
- credits : int
+ getTitle() : String
+ setCredits( credits : int )
+ getCredits() : int
+ getCourse() : String
...

Course
- registered : 0..*
- course : 1
- assistants : 0..*
- teacher : 1

Course
- registered : 0..*
- course : 1
```
Object Diagram

- Shows an example of objects with slots and links that could be instantiated from defined classes and relationships
- Validates class diagrams

<table>
<thead>
<tr>
<th>DB: Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>name = Daniel Brookshier</td>
</tr>
<tr>
<td>homeCampus = Plano, TX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>: Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>code = CS112</td>
</tr>
<tr>
<td>semester = Spring</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>uml2: Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>title = Applying UML 2 with MagicDraw</td>
</tr>
<tr>
<td>credits = 5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DS: Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>name = Darius Silingas</td>
</tr>
<tr>
<td>homeCampus = Kaunas, LT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>: Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>code = CS221</td>
</tr>
<tr>
<td>semester = Autumn</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>req: Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>title = Requirements Management</td>
</tr>
<tr>
<td>credits = 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>: Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>code = BM101</td>
</tr>
<tr>
<td>semester = Spring</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DB: Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>name = Daniel Brookshier</td>
</tr>
<tr>
<td>homeCampus = Plano, TX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>: Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>code = CS112</td>
</tr>
<tr>
<td>semester = Spring</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>uml2: Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>title = Applying UML 2 with MagicDraw</td>
</tr>
<tr>
<td>credits = 5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DS: Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>name = Darius Silingas</td>
</tr>
<tr>
<td>homeCampus = Kaunas, LT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>: Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>code = CS221</td>
</tr>
<tr>
<td>semester = Autumn</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>req: Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>title = Requirements Management</td>
</tr>
<tr>
<td>credits = 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>: Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>code = BM101</td>
</tr>
<tr>
<td>semester = Spring</td>
</tr>
</tbody>
</table>
Package Diagram

- Decomposes system into **logical units of work**
- Describe the **dependencies** between logical units of work
- Provide views of a system from multiple **levels of abstraction**
Composite Structure Diagram (1)

- Shows the **internal structure** of a classifier, including its **interaction points** to other parts of the system.
- More useful for modeling hardware, real-time systems, integrated device modeling.

```
Car

<table>
<thead>
<tr>
<th>front : Wheel [2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>rear : Wheel [2]</td>
</tr>
<tr>
<td>: Axle</td>
</tr>
<tr>
<td>engine : Engine [1]</td>
</tr>
</tbody>
</table>

Engine

| p : PowerPort |

Wheel

| a : Axle |
```

Powertrain

Power
Composite Structure Diagram (2)

- Shows the configuration and relationship of parts that together perform the behavior of the containing classifier
- Useful for defining static structure of collaboration patterns
Activity Diagram

- Shows a **procedural flow** for a process
- Useful for **workflow modeling**
- Supports **parallel behavior** for multithreaded programming

```plaintext
: Student
  Choose Class
  Apply for Class
  Enroll Student
  Check Schedule

Registation System
  Check for Availability
    [no seats left]
    [seats available]
  Check for Prerequisite Course Completion
    [does not qualify]
    [qualifies]
  Check for Schedule Conflict
    [no conflict]
  Add Class to Schedule
```

© 2003-2006 No Magic, Inc. Exclusively for No Magic Use
Sequence Diagram (1)

- Describes how a process is performed by a group of objects by a **sequential set of interactions**
- Provides an **object-oriented** view of a procedural views
- Facilitates **assignment of responsibilities** to classes
- Helps finding out **new methods and new classes**
- Shows **timing** very explicitly

- (Diagram on next slide)
Sequence Diagram (2)

1: submit()

2: registerForClass(student=ME, class=uml2)

3: validateRegistration(student=ME, class=uml2)

4: seatsAvailable()

5: true

6: getPrerequisites()

7: preCourses

8: hasTakenCourses(courses=preCourses)

9: true

10: getSchedule()

11: schedule

12: schedule

13: hasConflicts(class=uml2)

14: false

15: addClass(class=uml2)

16: sendNotification()
Communication Diagram

- Provides an alternative view to the sequence diagram in a format based on structure rather than time
- Emphasizes how objects interact with each other
- More efficient use of space
State Diagram

- Describes how an object changes its state that govern its behavior in response to stimuli from the environment.

State Diagram:

- **Announced**
  - at (4 weeks before class)

- **Registration Open**
  - at (1 week before class)

- **Registration Closed**
  - when (reports prepared)

- **[students >= 5]**
  - **[students < 5]**

- **[students >= 5]**
  - **[students < 5]**

- **Running**
  - when (grades finalized)

- **Finished**
Component Diagram

- Describes software components that make up a system, their interfaces (optional) and relationships
Deployment Diagram

- Describes the **configuration of hardware** in a system in terms of nodes and connections
- Describes the **physical relationships between software and hardware**
- Displays how **artifacts are installed** and move around a distributed system
But That’s Not All…

UML provides extensions to the language to create new types of diagrams

UML Profiles define a set of extensions for a specific usage, e.g. new domains, technologies, or methods

- Stereotypes «Process»
- Tagged Values approval_status="draft"
- Constraints {deliver within 48 hours}
- Customizable Icons
Extending UML – Robustness Diagram

Student

Registration Clerk

ClassBrowser

<<boundary>>

NotificationService

<<control>>

RegistrationManager

<<control>>

Student

Course

Class

Schedule

Class

RegistrationForm

<<boundary>>

ClassManager

<<control>>

© 2003-2006 No Magic, Inc. Exclusively for No Magic Use
Where To Go To Learn More

UML Web Resources:
- [http://www.objectsbydesign.com](http://www.objectsbydesign.com)
  - UML and OO links, forums, and resources
  - UML developer zone
  - Magazine with many UML related articles
- [http://www.omg.org](http://www.omg.org)
  - The UML Specification and other UML resources

UML Books
- **UML Bible** by Tom Pender
- **UML Distilled** by Martin Fowler & Kendal Scott
- **Applying UML & Patterns** by Craig Larman