Modeling with UML: Basic Notations

Software Engineering I
Lecture 2
7 November 2006

Prof. Bernd Bruegge, Ph.D.
Applied Software Engineering
Technische Universitaet Muenchen
Overview

• Odds and Ends
• Modeling
• The UML notation
• Use case diagrams
• Class diagrams
• Sequence diagrams
• Activity diagrams
Odds and Ends (1)

• Reading for this Week:
  • Chapter 1 and 2, Bruegge&Dutoit, Object-Oriented Software Engineering

• Software Engineering I Portal
  • [http://wwwbruegge.in.tum.de/static/contribute/Lehrstuhl/SoftwareTechnikWiSe05.htm](http://wwwbruegge.in.tum.de/static/contribute/Lehrstuhl/SoftwareTechnikWiSe05.htm)

• Lectures Slides:
  • Will be sent to you via e-mail if you are registered for this class.
Lecture Schedule

Tuesdays 12:15-13:45
✓ Oct 24: Introduction

• Oct 31: Modeling with UML moved to Nov 7
• Nov 7: Project Organization moved to January
• Nov 14: Functional Modeling
• Nov 21: Dynamic Modeling
• Nov 28: Architectural Styles
• Nov 30: Reuse
• Dec 5: No lecture
• Dec 12: Design Patterns
• Dec 19: Object Constraint Language

Wednesday 9:15-10:00
• Oct 25: Introduction ctd

• Nov 1: Holiday (Allerheiligen)
• Nov 8: Requirements Elicitation
• Nov 15: Object Modeling
• Nov 22: Design Goals
• Nov 29: Addressing Design Goals
• Dec 6: No lecture
• Dec 13: Interface Specification
• Dec 20: Mid-term

Always subject to Change!
What is modeling?

• Modeling consists of building an abstraction of reality

• Abstractions are simplifications because:
  • They ignore irrelevant details and
  • They only represent the relevant details

• What is relevant or irrelevant depends on the purpose of the model.

• Models can be used for 2 purposes:
  • Gain insight into the past and presence
  • Predict future behavior.
Example of a Model: A Street Map
Why should we model Software?

• Software is used in many appliances and everyday objects
• Software is getting increasingly more complex
  • Windows 2000: ~ 40 millions lines of code
  • A single programmer cannot manage this amount of code in its entirety
• Code is not easily understandable by developers who did not write it
• We need simpler representations for complex systems
  • Modeling is a mean for dealing with complexity.
What is a “good” Model?

• Interpretation I: Maps entities in R to entities in M
  • $f_M$: Relationship between entities in M
  • $f_R$: Relationship between entities in R

• Relationships that are valid in reality R are also valid in the model M.

• For a good model, the following is true:

\[ f_M \circ I = f_R \circ I \]
Model of Models of Models...

- Modeling is relative.
  - One can regard a model again as reality and make another model of it (with more abstractions)

The development of software systems can be seen as a sequence of transformations and validations of models: Analysis, System Design, Implementation
Software Development is a Sequence of Transformations

\[ f_{R} \rightarrow R \]
\[ f_{M1} \rightarrow M_{1} \]
\[ f_{M2} \rightarrow M_{2} \]
\[ \ldots \rightarrow \]

Analysis

Design

Implementation

Source code

Design model

Analysis model

Reality

Application Domain

Solution Domain

Reality

Analysis

Design

Implementation
Models must be falsifiable

- Karl Popper ("Objective Knowledge):
  - There is no absolute truth when trying to understand reality
  - One can only build theories, that are "true" until somebody finds a counter example

- **Falsification**: The act of disproving a theory or hypothesis

- The truth of a theory is never certain. We must use phrases like:
  - "by our best judgement", "using state-of-the-art knowledge"

- In software engineering any model is a theory:
  - We build models and try to find counter examples by:
    - Requirements validation, user interface testing, review of the design, source code testing, system testing, etc.

- **Testing**: The act of disproving a model.
Concepts and Phenomena

• **Phenomenon**
  - An object in the world of a domain as you perceive it
    - Examples: This lecture on November 7 at 12:30, my black watch

• **Concept**
  - Describes the common properties of phenomena
    - Example: All lectures on software engineering
    - Example: All black watches

• **A Concept is a 3-tuple:**
  - **Name:** The name distinguishes the concept from other concepts
  - **Purpose:** Properties that determine if a phenomenon is a member of a concept
  - **Members:** The set of phenomena which are part of the concept.
### Concepts, Phenomena, Abstraction and Modeling

<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
<th>Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>Watch</td>
<td>A device that measures time.</td>
<td>![Clock], ![Sand Timer]</td>
</tr>
</tbody>
</table>

**Definition Abstraction:**
- Classification of phenomena into concepts

**Definition Modeling:**
- Development of abstractions to answer specific questions about a set of phenomena while ignoring irrelevant details.
Abstract Data Types & Classes

• Abstract data type
  • A type whose implementation is hidden from the rest of the system

• Class:
  • An abstraction in the context of object-oriented languages
  • A class encapsulates state and behavior
    • Example: Watch

Unlike abstract data types, subclasses can be defined in terms of other classes using inheritance
  • Example: CalculatorWatch
Type and Instance

- **Type**:  
  - An concept in the context of programming languages  
  - Name: int  
  - Purpose: integral number  
  - Members: 0, -1, 1, 2, -2, ...

- **Instance**:  
  - Member of a specific type

- The type of a variable represents all possible instances of the variable

The following relationships are similar:

- Type <-> Variable
- Concept <-> Phenomenon
- Class <-> Object
Systems

• A **system** is an organized set of communicating parts
  • **Natural system**: A system whose ultimate purpose is not known
  • **Engineered system**: A system which is designed and built by engineers for a specific purpose

• The parts of the system can be considered as systems again
  • In this case we call them **subsystems**

Examples of natural systems:
• Universe, earth, ocean

Examples of engineered systems:
• Airplane, watch, GPS

Examples of subsystems:
• Jet engine, battery, satellite.
Systems, Models and Views

- A **model** is an abstraction describing a system or a subsystem
- A **view** depicts selected aspects of a model
- A **notation** is a set of graphical or textual rules for depicting models and views: formal notations, "napkin notations"

System: Airplane

**Models:**
- Flight simulator
- Scale model

**Views:**
- Blueprint of airplane components
- Electrical wiring diagram
- Fuel system
- Sound wave created by airplane
Views and models of a complex system usually overlap.
Systems, Models and Views (UML Notation)

Class Diagram

System * Model * View

Described by Depicted by

Airplane: System

Object Diagram

Scale Model: Model

Blueprints: View

Flight Simulator: Model

Fuel System: View

Electrical Wiring: View
Model-Driven Development

1. Build a platform-independent model of an applications functionality and behavior
   a) Describe model in modeling notation (UML)
   b) Convert model into platform-specific model

2. Generate executable from platform-specific model

Advantages:

- Code is generated from model ("mostly")
- Portability and interoperability

- Model Driven Architecture effort:
  - http://www.omg.org/mda/

- OMG: Object Management Group
Reality: A stock exchange lists many companies. Each company is identified by a ticker symbol.

Analysis results in analysis object model (UML Class Diagram):

```
Model-driven Software Development

public class StockExchange {
    public m_Company = new Vector();
}
public class Company {
    public int m_tickerSymbol;
    public Vector m_StockExchange = new Vector();
}
```
Application vs Solution Domain

- **Application Domain** (Analysis):
  - The environment in which the system is operating

- **Solution Domain** (Design, Implementation):
  - The technologies used to build the system

- Both domains contain abstractions that we can use for the construction of the system model.
Object-oriented Modeling

Application Domain (Phenomena)

System Model (Concepts) (Analysis)

- TrafficControl
- Aircraft
- TrafficController
- Airport
- FlightPlan

Solution Domain (Phenomena)

System Model (Concepts) (Design)

- MapDisplay
- FlightPlanDatabase
- Summary Display
- TrafficControl

© 2006 Bernd Bruegge
What is UML?

• UML (Unified Modeling Language)
  • Nonproprietary standard for modeling software systems, OMG
  • Convergence of notations used in object-oriented methods
    • OMT (James Rumbaugh and colleagues)
    • Booch (Grady Booch)
    • OOSE (Ivar Jacobson)

• Current Version 2.0
  • Information at the OMG portal http://www.uml.org/

• Commercial tools: Rational (IBM), Together (Borland), Visual Architect (business processes, BCD)

• Open Source tools: ArgoUML, StarUML, Umbrello

• Commercial and Opensource: PoseidonUML (Gentleware)
UML: First Pass

• You can model 80% of most problems by using about 20% UML
• We teach you those 20%

• 80-20 rule: Pareto principle
  • http://www.ephorie.de/hindle_pareto-prinzip.htm
UML First Pass

• **Use case diagrams**
  • Describe the functional behavior of the system as seen by the user

• **Class diagrams**
  • Describe the static structure of the system: Objects, attributes, associations

• **Sequence diagrams**
  • Describe the dynamic behavior between objects of the system

• **Statechart diagrams**
  • Describe the dynamic behavior of an individual object

• **Activity diagrams**
  • Describe the dynamic behavior of a system, in particular the workflow.
UML Core Conventions

• All UML Diagrams denote graphs of nodes and edges
  • Nodes are entities and drawn as rectangles or ovals
  • **Rectangles** denote classes or instances
  • **Ovals** denote functions

• Names of Classes are not underlined
  • SimpleWatch
  • Firefighter

• Names of Instances are underlined
  • myWatch:SimpleWatch
  • Joe:Firefighter

• An edge between two nodes denotes a relationship between the corresponding entities
UML first pass: Use case diagrams

Use case diagrams represent the functionality of the system from user’s point of view.
UML first pass: Class diagrams

Class diagrams represent the structure of the system
UML first pass: Class diagrams

Class diagrams represent the structure of the system.
Sequence diagrams represent the behavior of a system as messages (“interactions”) between different objects.
UML first pass: Statechart diagrams

Represent behavior of a single object with interesting dynamic behavior.
Other UML Notations

UML provides many other notations

- Activity diagrams for modeling work flows
- Deployment diagrams for modeling configurations (for testing and release management)
What should be done first? Coding or Modeling?

• It all depends….

• **Forward Engineering**
  - Creation of code from a model
  - Start with modeling
  - Greenfield projects

• **Reverse Engineering**
  - Creation of a model from existing code
  - Interface or reengineering projects

• **Roundtrip Engineering**
  - Move constantly between forward and reverse engineering
  - Useful when requirements, technology and schedule are changing frequently.
UML Basic Notation Summary

• UML provides a wide variety of notations for modeling many aspects of software systems
• For now we have concentrated on a few notations:
  • Functional model: Use case diagram
  • Object model: Class diagram
  • Dynamic model: Sequence diagrams, statechart
Additional References

• Martin Fowler

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

• Commercial UML tools
  • Rational Rose XDE for Java
    • http://www-306.ibm.com/software/awdtools/developer/java/
  • Together (Eclipse, MS Visual Studio, JBuilder)

• Open Source UML tools
  • http://java-source.net/open-source/uml-modeling
  • ArgoUML, UMLet, Violet, ...