Chapter 2, Modeling with UML, Part 1

Odds and Ends

• Reading for this Lecture:
  • Chapter 1 and 2, Bruegge&Dutoit, Object-Oriented Software Engineering
• Lectures Slides:
  • Will be posted before each lecture.
Overview for the Lecture

• Three ways to deal with complexity
  • Abstraction and Modeling
    • (Abstraction -> Hiding details)
  • Decomposition
    • A complex problem or system is broken down into parts that are easier to conceive
  • Hierarchy
    • a hierarchy can be modelled as a rooted tree

• Introduction into the UML notation
• First pass on:
  • Use case diagrams
  • Class diagrams
  • Sequence diagrams
  • Statechart diagrams
  • Activity diagrams

What is the problem with this Drawing?
Abstraction

• Complex systems are hard to understand
  • The 7 +- 2 phenomena
    • Our short term memory cannot store more than 7+-2 pieces at the same time -> limitation of the brain
    • My Phone Number: 498928918204
  • Chunking:
    • Group collection of objects to reduce complexity
    • 4 chunks:
      • State-code, Area-code, Local-Prefix, Internal-Nr
Abstraction

- Complex systems are hard to understand
  - The 7 +- 2 phenomena
    - Our short term memory cannot store more than 7+-2 pieces at the same time -> limitation of the brain
    - My Phone Number: 003909123842261

- Chunking:
  - Group collection of objects to reduce complexity
  - State-code, Area-code, Local Prefix, Internal-Nr

Phone Number

Country-Code
  - 0039

Area-Code
  - 091

Local-Prefix
  - 238 42

Internal-Nr
  - 261

Abstraction

- Abstraction allows us to ignore unessential details

- Ideas can be expressed by models
Models

- A model is an abstraction of a system
  - A system that no longer exists
  - An existing system
  - A future system to be built.

Why model software?

- Software is getting increasingly more complex
  - Windows XP > 40 millions of 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
We use Models to describe Software Systems

- **Object model**: What is the structure of the system?
- **Functional model**: What are the functions of the system?
- **Dynamic model**: How does the system react to external events?

- **System Model**: Object model + functional model + dynamic model

2. Technique to deal with Complexity: Decomposition

- A technique used to master complexity (“divide and conquer”)
- Two major types of decomposition
  - Functional decomposition
  - Object-oriented decomposition

  - **Functional decomposition**
    - The system is decomposed into modules
    - Each module is a major function in the application domain
    - Modules can be decomposed into smaller modules.
Decomposition (cont’d)

• Object-oriented decomposition
  • The system is decomposed into classes ("objects")
  • Each class is a major entity in the application domain
  • Classes can be decomposed into smaller classes

• Object-oriented vs. functional decomposition

Which decomposition is the right one?

Functional Decomposition

System Function

Top Level functions

Level 1 functions

Level 2 functions

Load R10

Add R1, R10

Machine instructions
Functional Decomposition

• The functionality is spread all over the system
• Maintainer must understand the whole system to make a single change to the system
• Consequence:
  • Source code is hard to understand
  • Source code is complex and impossible to maintain
  • User interface is often awkward and non-intuitive.
What is This?

An Eskimo!

A Face!
Class Identification

- **Basic assumptions:**
  - We can find the *classes for a new software system*: Greenfield Engineering
  - We can identify the *classes in an existing system*: Reengineering
  - We can create a *class-based interface to an existing system*: Interface Engineering.
3. Hierarchy

- So far we got abstractions
  - This leads us to classes and objects
  - “Chunks”

- Another way to deal with complexity is to provide relationships between these chunks
- One of the most important relationships is hierarchy
- 2 special hierarchies
  - "Part-of" hierarchy
  - "Is-kind-of" hierarchy.

### Part-of Hierarchy (Aggregation)

```
  Computer
   / \                                    
  /   \                                   
/     \                                   
I/O Devices      CPU     Memory
               /     /         /
            /     /         /
           /     /         /
          Cache   ALU     Program Counter
```

M. Cossentino
Is-Kind-of Hierarchy (Taxonomy)

Where are we?

- Three ways to deal with complexity:
  - Abstraction, Decomposition, Hierarchy
- Object-oriented decomposition is good
  - Unfortunately, depending on the purpose of the system, different objects can be found
- How can we do it right?
  - Start with a description of the functionality of a system
  - Then proceed to a description of its structure
- Ordering of development activities
  - Software lifecycle
**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 designs”

**System: Airplane**

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

**Views:**
- Blueprint of the airplane components
- Electrical wiring diagram, Fuel system
- Sound wave created by airplane

**Systems, Models and Views (“Napkin” Notation)**

Views and models of a complex system usually overlap.


Systems, Models and Views (UML Notation)

Class Diagram

```
System * Model * View
Described by  Depicted by
```

Object Diagram

- Airplane: System
- Scale Model: Model
- Flight Simulator: Model
- Blueprints: View
- 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:
- 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):

```
<table>
<thead>
<tr>
<th>StockExchange *</th>
<th>Lists *</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Company</td>
</tr>
<tr>
<td></td>
<td>tickerSymbol</td>
</tr>
</tbody>
</table>
```

**Implementation** results in source code (Java):

```java
public class StockExchange {
    private m_Company = new Vector();
};
public class Company {
    private int m_tickerSymbol;
    private 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)

Solution Domain (Phenomena)

System Model (Concepts) (Analysis)

System Model (Concepts) (Design)

TrafficController
Aircraft
TrafficController
Airport
FlightPlan

UML Package
MapDisplay
Summary Display
FlightPlanDatabase
TrafficControl

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: UML 2.4.1
    - 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), Astah, Violet
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