Chapter 2, Modeling with UML, Part 2

Outline of this Lecture

- 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.

What is UML? Unified Modeling Language

- Convergence of different notations used in object-oriented methods, mainly
  - OMT (James Rumbaugh and colleagues), OOSE (Ivar Jacobson), Booch (Grady Booch)
- They also developed the Rational Unified Process, which became the Unified Process in 1999

UML

- Nonproprietary standard for modeling systems
- Current Version: UML 2.2
  - Information at the OMG portal http://www.uml.org/
- Commercial tools:
  - Rational (IBM), Together (Borland), Visual Architect (Visual Paradigm), Enterprise Architect (Sparx Systems)
- Open Source tools http://www.sourceforge.net/
  - ArgoUML, StarUML, Umbrello (for KDE), PoseidonUML
- Example of research tools: Unicase, Sysiphus
  - Based on a unified project model for modeling, collaboration and project organization
    - http://unicase.org
    - http://sysiphus.in.tum.de/

UML: First Pass

- You can solve 80% of the modeling problems by using 20% UML
- We teach you those 20%
- 80-20 rule: Pareto principle

Vilfredo Pareto, 1848-1923
Introduced the concept of Pareto Efficiency.
Founder of the field of microeconomics.

UML First Pass (covered in Last Lecture)

- 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 Basic Notation: First Summary**

- UML provides a wide variety of notations for modeling many aspects of software systems.
- In the first lecture we concentrated on:
  - Functional model: Use case diagram
  - Object model: Class diagram
  - Dynamic model: Sequence diagrams, statechart
- Now we go into a little bit more detail...

**UML Use Case Diagrams**

- **Use case model:** The set of all use cases that completely describe the functionality of the system.
- **Actor** represents a role, that is, a type of user of the system.
- **Use case** represents a class of functionality provided by the system.

**Actors**

- An actor is a model for an external entity which interacts (communicates) with the system:
  - User
  - External system (Another system)
  - Physical environment (e.g., Weather)
- An actor has a unique name and an optional description.
- Examples:
  - **Passenger:** A person in the train
  - **GPS satellite:** An external system that provides the system with GPS coordinates.

**Textual Use Case Description Example**

1. **Name:** Purchase Ticket
2. **Participating actor:** Passenger
3. **Entry condition:**
   - Passenger stands in front of ticket distributor
   - Passenger has sufficient money to purchase ticket
4. **Exit condition:**
   - Passenger has ticket
5. **Flow of events:**
   1. Passenger selects the number of zones to be traveled
   2. Ticket Distributor displays the amount due
   3. Passenger inserts money, at least the amount due
   4. Ticket Distributor returns change
   5. Ticket Distributor issues ticket
6. **Special requirements:** None.

**Uses Cases can be related**

- **Extends Relationship**
  - To represent seldom invoked use cases or exceptional functionality
- **Includes Relationship**
  - To represent functional behavior common to more than one use case.
The <<extends>> Relationship

- <<extends>> relationships model exceptional or seldom invoked cases
- The exceptional event flows are factored out of the main event flow for clarity
- The direction of an <<extends>> relationship is to the extended use case
- Use cases representing exceptional flows can extend more than one use case.

The <<includes>> Relationship

- <<includes>> relationship represents common functionality needed in more than one use case
- <<includes>> behavior is factored out for reuse, not because it is an exception
- The direction of a <<includes>> relationship is to the using use case (unlike the direction of the <<extends>> relationship).

Class Diagrams

- Class diagrams represent the structure of the system
- Used during requirements analysis to model application domain concepts
- During system design to model subsystems
- During object design to specify the detailed behavior and attributes of classes.

Classes

- A class represents a concept
- A class encapsulates state (attributes) and behavior (operations)
- Each attribute has a type
- Each operation has a signature

The class name is the only mandatory information

Instances

- An instance represents a phenomenon
- The attributes are represented with their values
- The name of an instance is underlined
- The name can contain only the class name of the instance (anonymous instance)

Actor vs Class vs Object

- Actor
  - An entity outside the system to be modeled, interacting with the system ("Passenger")
- Class
  - An abstraction modeling an entity in the application or solution domain
  - The class is part of the system model ("User", "Ticket distributor", "Server")
- Object
  - A specific instance of a class ("Joe, the passenger who is purchasing a ticket from the ticket distributor").
**Associations**

The multiplicity of an association end denotes how many objects the instance of a class can legitimately reference.

**1-to-1 and 1-to-many Associations**

- **1-to-1 association**
  - Country
    - name: String
  - City
    - name: String

- **1-to-many association**
  - Polygon
    - draw()
  - Point
    - x: Integer
    - y: Integer

**Many-to-Many Associations**

- StockExchange
  - Company
    - tickerSymbol

**From Problem Statement To Object Model**

Problem Statement: A stock exchange lists many companies. Each company is uniquely identified by a ticker symbol.

**Class Diagram:**

- StockExchange
  - Lists
  - Company
    - tickerSymbol

**From Problem Statement to Code**

Problem Statement: A stock exchange lists many companies. Each company is identified by a ticker symbol.

**Class Diagram:**

- StockExchange
  - Lists
  - Company
    - tickerSymbol

**Java Code**

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

**Aggregation**

- An aggregation is a special case of association denoting a "consists-of" hierarchy.
- The aggregate is the parent class, the components are the children classes.

A solid diamond denotes composition: A strong form of aggregation where the life time of the component instances is controlled by the aggregate. That is, the parts don't exist on their own ("the whole controls/destroys the parts")
Qualifiers

Without qualification

- Qualifiers can be used to reduce the multiplicity of an association

With qualification

- Qualifiers can be used to reduce the multiplicity of an association

Inheritance

- Inheritance is another special case of an association denoting a “kind-of” hierarchy
- Inheritance simplifies the analysis model by introducing a taxonomy
- The children classes inherit the attributes and operations of the parent class.

Object Modeling in Practice

Class Identification: Name of Class, Attributes and Methods

Is Foo the right name?

Object Modeling in Practice: Brainstorming

Is Foo the right name?
Object Modeling in Practice: More classes

1) Find New Classes
2) Review Names, Attributes and Methods

Account
Bank
Amount
Deposit(), Withdraw(), GetBalance()
CustomerId

Customer
Name

Object Modeling in Practice: Associations

1) Find New Classes
2) Review Names, Attributes and Methods
3) Find Associations between Classes
4) Label the generic associations
5) Determine the multiplicity of the associations
6) Review associations

Account
Bank
Amount
Deposit(), Withdraw(), GetBalance()
CustomerId

Customer
Name

Bank
Name

Practice Object Modeling: Find Taxonomies

Bank
Name

Account
Amount
Deposit(), Withdraw(), GetBalance()
CustomerId

Customer
Name

Savings Account
Withdraw()
Checking Account
Withdraw()
Mortgage Account
Withdraw()

Practice Object Modeling: Simplify, Organize

Show Taxonomies separately

Account
Amount
Deposit(), Withdraw(), GetBalance()
CustomerId

Customer
Name

Savings Account
Withdraw()
Checking Account
Withdraw()
Mortgage Account
Withdraw()

Practice Object Modeling: Simplify, Organize

Use the 7+-2 heuristics or better 5+-2!

Se l ec t Zone ( )

p i ckupChange ( )

p i ckUpT i cke t ( )

i n se r tCo ins ( )

T i cke tMach ine

Focus on Controlflow

Messages are Operations on participating Object

Used during analysis
- To refine use case descriptions
- to find additional objects ("participating objects")

Used during system design
- Fine grain interface communication
- To refine use case descriptions

Messages are represented by arrows

Activation is represented by narrow rectangles.

Sequence Diagrams
Sequence Diagrams can also model the Flow of Data

- The source of an arrow indicates the activation which sent the message
- Horizontal dashed arrows indicate data flow, for example return results from a message.

Sequence Diagrams: Iteration & Condition

- Iteration is denoted by a * preceding the message name
- Condition is denoted by boolean expression in [ ] before the message name

Creation and destruction

- Creation is denoted by a message arrow pointing to the object
- Destruction is denoted by an X mark at the end of the destruction activation
- In garbage collection environments, destruction can be used to denote the end of the useful life of an object.

Sequence Diagram Properties

- UML sequence diagram represent behavior in terms of interactions
- Useful to identify or find missing objects
- Time consuming to build, but worth the investment
- Complement the class diagrams (which represent structure).

Outline of this Class

- A more detailed view on
  - Use case diagrams
  - Class diagrams
  - Sequence diagrams
  - Activity diagrams

Activity Diagrams

- An activity diagram is a special case of a state chart diagram
- The states are activities ("functions")
- An activity diagram is useful to depict the workflow in a system
Activity Diagrams allow to model Decisions

- Open Incident
- Notify Police Chief
- Allocate Resources
- [not fire & highPriority]
  - Notify Fire Chief
- [fire & highPriority]
- Allocate Resources
- Decisions

Activity Diagrams can model Concurrency

- Synchronization of multiple activities
- Splitting the flow of control into multiple threads

Activity Diagrams: Grouping of Activities

- Activities may be grouped into swimlanes to denote the object or subsystem that implements the activities.

Activity Diagram vs. Statechart Diagram

Statechart Diagram for Incident
- Focus on the set of attributes of a single abstraction (object, system)
- Event causes state transition

Activity Diagram for Incident
- Focus on dataflow in a system
- Completion of activity causes state transition
- Triggerless transition

UML Summary

- UML provides a wide variety of notations for representing many aspects of software development
  - Powerful, but complex
- UML is a programming language
  - Can be misused to generate unreadable models
  - Can be misunderstood when using too many exotic features
- We concentrated on a few notations:
  - Functional model: Use case diagram
  - Object model: class diagram
  - Dynamic model: sequence diagrams, statechart and activity diagrams