

THE UNIFIED MODELING LANGUAGE FOR OBJECT-ORIENTED DESIGN

Barrett R. Bryant

Department of Computer and Information Sciences
University of Alabama at Birmingham
1300 University Blvd.
Birmingham, Alabama 35294-1170, U. S. A.
bryant@cis.uab.edu

References:

Grady Booch, *Object-Oriented Analysis and Design with Applications*, Benjamin/Cummings, 1994.

Hans-Erik Eriksson and Magnus Pinker, *UML Toolkit*, John Wiley and Sons, 1998.

<http://www.rational.com/uml>

Outline of the Talk

- Object-oriented software development
- UML
- Example of using UML

Algorithmic vs. Object-Oriented Decomposition

G. Booch, *Object-Oriented Analysis and Design with Applications*,
Benjamin/Cummings, 1994, Figures 1-2 and 1-3.

Object-Oriented Software Engineering

Object-Oriented Analysis - a method of analysis that examines software requirements from the perspective of the classes and objects found in the vocabulary of the problem domain

Object-Oriented Design - a method of design encompassing the processes of object-oriented analysis and object-oriented decomposition

Object-Oriented Programming - a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships

Phases of System Development

- Requirements analysis - actors and use cases
- Analysis - classes, objects and their collaboration
- Design - technical infrastructure
- Programming - code
- Testing - validation with requirements

Object-Oriented Design

- Technology for producing models that reflect an application domain in a natural way
- Facilitates communication, change, expansion, validation and verification
- Provides opportunity to create and implement reusable components
- Convenient implementation in object-oriented programming language
- System requirements are traceable to code
- Integrated method for design and implementation

UML

- Unified Modeling Language of Booch, Rumbaugh and Jacobsen
- Models systems and software using object-oriented approach
- Couples conceptual and executable artifacts
- Addresses large-scale systems
- Usable by both humans and machines
- Accepted as standard by OMG (Object Management Group)
- Supported by tools such as Rational Rose and Popkin System Architect

UML Modeling Views

- Use-case - functionality of system as perceived by external actors
- Logical - system's static structure and dynamic behavior
- Component - organization of code components
- Concurrency - communication and synchronization between concurrent components
- Deployment - implementation on physical architecture

UML Tool Functions

- Draw diagrams
- Act as repository
- Support navigation
- Provide multiuser support
- Generate code
- Reverse engineer
- Integrate with other tools
- Cover all abstraction levels
- Interchange models

Requirements Specification

A library support system with requirements:

- The library lends books and magazines to borrowers.
- All books, magazines, and borrowers are registered.
- New titles may be purchased, sometimes as multiple copies. Old titles may be removed.
- The librarian interacts with the borrowers.
- Borrowers can reserve books/magazines not currently available. This reservation system will notify the borrower upon availability and is subject to cancellation.
- For all entities of the system, information may be created, updated and deleted.

Use-Case Diagram

H. Eriksson and M. Pinder, *UML Toolkit*, John Wiley and Sons, 1998,
Figure 12-1.

Lending Item Use Case

1. If the borrower has no reservation:

- (a) Identify title
- (b) Identify available item
- (c) Identify borrower
- (d) Library lends item
- (e) Register new loan

2. If the borrower has a reservation:

- (a) Identify borrower
- (b) Identify title
- (c) Identify available item
- (d) Library lends item
- (e) Register new loan
- (f) Remove reservation

Domain Class Structure

H. Eriksson and M. Pinder, *UML Toolkit*, John Wiley and Sons, 1998,
Figure 12-2.

State Diagram for Title Class

H. Eriksson and M. Pinker, *UML Toolkit*, John Wiley and Sons, 1998,
Figure 12-3.

Sequence Diagram for Use-Case Lend Item

H. Eriksson and M. Pinker, *UML Toolkit*, John Wiley and Sons, 1998,
Figure 12-4.

System Architecture

H. Eriksson and M. Pinder, *UML Toolkit*, John Wiley and Sons, 1998,
Figure 12-5.

Business Objects

H. Eriksson and M. Pinder, *UML Toolkit*, John Wiley and Sons, 1998,
Figure 12-6.

Business Objects (continued)

H. Eriksson and M. Pinter, *UML Toolkit*, John Wiley and Sons, 1998, Figure 12-6 (zoom).

State Diagram for Title with Implementation

H. Eriksson and M. Pinker, *UML Toolkit*, John Wiley and Sons, 1998,
Figure 12-7.

Sequence Diagram for Add Title Use Case

H. Eriksson and M. Pinker, *UML Toolkit*, John Wiley and Sons, 1998,
Figure 12-8.

Collaboration Diagram for Add Title Use Case

H. Eriksson and M. Pinner, *UML Toolkit*, John Wiley and Sons, 1998, Figure 12-9.

User Interface Classes

H. Eriksson and M. Pinder, *UML Toolkit*, John Wiley and Sons, 1998,
Figure 12-10.

Component Diagram

H. Eriksson and M. Pinter, *UML Toolkit*, John Wiley and Sons, 1998,
Figure 12-12.

Sample Java Code

```
// Loan.java: represents a loan. The loan  
// refers to one title and one borrower.
```

```
package bo;  
import util.ObjId;  
import db.*;  
import java.io.*;  
import java.util.*;
```

```
public class Loan extends Persistent {  
    private ObjId item;  
    private ObjId borrower;  
    public Loan() { }  
    public Loan(ObjId it, ObjId b) {  
        item = it;  
        borrower = b;  
    }  
    public BorrowerInformation getBorrower() {  
        BorrowerInformation ret =  
            (BorrowerInformation) Persistent .  
                getObject(borrower);  
        return ret;  
    }  
}
```

```
public String getTitleName() {
    Item it = (Item) Persistent.getObject(item);
    return it.getTitleName();
}
public Item getItem() {
    Item it =
        (Item) Persistent.getObject(item);
    return it;
}
public int getItemId() {
    Item it = (Item) Persistent.getObject(item);
    return it.getId();
}
public void write(RandomAccessFile out)
    throws IOException {
    item.write(out);
    borrower.write(out);
}
public void read(RandomAccessFile in)
    throws IOException {
    item = new ObjId();
    item.read(in);
    borrower = new ObjId();
    borrower.read(in);
}
}
```