Systems Analysis and Design

Alan Dennis, Barbara Haley Wixom, and Roberta Roth
John Wiley & Sons, Inc.

Slides by Candace S. Garrod
Red Rocks Community College
Unified Modeling Language

Chapter 14
Key Definitions

- **Object-oriented techniques** view a system as a collection of self-contained objects which include both data and processes.

- The **Unified Modeling Language (UML)**
  - the object modeling standard
  - adds a variety of techniques to the field of system development.
BASIC CHARACTERISTICS OF OBJECT-ORIENTED SYSTEMS
Object Concepts

- An object is a person, place, event, or thing about which we want to capture information.
- Each object has properties (or attributes).
- The state of an object is defined by the value of its properties and relations with other objects at a point in time.
- Objects have behaviors -- things that they can do -- which are described by methods (or operations).
- Objects do not use primary or foreign keys, instead each instance is assigned a unique identifier (UID) when it is created.
Classes and Objects

Classes
- Patient
  - Name
  - Birthdate
  - Address
  - Phone Number
+ Insert ()
+ Delete ()

Appointment
- Patient name
- Doctor name
- Date
- Time
+ Insert ()
+ Delete ()

Objects
An instance of the Patient class

aPatient : Patient
Name = Theresa Marks
Birthdate = March 26, 1965
Address = 50 Winds Way, Ocean City, NJ 09009
Phone Number = (604) 555-7889

An instance of the Appointment class

anAppointment : Appointment
Patient name = John Smith
Doctor name = Dr. David Broussasou
Date = September 17, 2006
Time = 9:30 A.M.
A class is a general template we use to define and create specific instances or objects.
Object

An object is an instantiation of a class.

An object is a person, place, event, or thing about which we want to capture information.
Messages and Methods

Messages are information sent to objects to trigger methods.

Insert new instance.

A message is sent to the application.

The object's insert method will respond to the message and insert a new patient instance.
Encapsulation and Information Hiding

Encapsulation is simply the combination of process and data into a single entity.

The principle of information hiding suggests that only the information required to use a software module be published to the user of the module.
Inheritance

- Classes are arranged in a hierarchy
  - Superclasses or general classes are at the top
  - Subclasses or specific classes are at the bottom
    - Subclasses inherit attributes and methods from the superclasses above them
- Classes with instances are concrete classes
- Abstract classes only produce templates for more specific classes
Class Hierarchy

- Person (Abstract class)
  - Doctor (Abstract class)
    - General practitioner
    - Specialist (Concrete class)
  - Patient (Concrete class)
Inheritance

**Without Inheritance**

- **Patient**
  - Last name
  - First name
  - Birthdate
  - Insurance carrier
  + Update carrier()
  + Update birthdate()

- **Doctor**
  - Last name
  - First name
  - Birthdate
  - Medical school specialty
  + Update specialty()
  + Update birthdate()

**With Inheritance**

- **Person**
  - Last name
  - First name
  - Birthdate
  + Update birthdate()

- **Patient**
  - Insurance carrier
  + Update carrier()

- **Doctor**
  - Medical school specialty
  + Update specialty()
Polymorphism and Dynamic Binding

1. An insert message is sent to the patient object.
2. The object’s method responds to the message.
3. The application responds appropriately.

1. An insert message is sent to the appointment object.
2. The object’s method responds to the message.
3. The application responds appropriately.
## Benefits of an Object Approach

<table>
<thead>
<tr>
<th>Concept</th>
<th>Supports...</th>
<th>Leads to...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes, objects, methods, and messages</td>
<td>• A more realistic way for people think about their business</td>
<td>• Better communication between user and analyst or developer</td>
</tr>
<tr>
<td></td>
<td>• Highly cohesive units that contain both data and processes</td>
<td>• Reusable objects</td>
</tr>
<tr>
<td></td>
<td>• Benefits from having a highly cohesive system (see cohesion in Chapter 11)</td>
<td>• Benefits from having a loosely coupled system design (see coupling in Chapter 11)</td>
</tr>
<tr>
<td>Encapsulation and information hiding</td>
<td>• Loosely coupled units</td>
<td>• Reusable objects</td>
</tr>
<tr>
<td></td>
<td>• Fewer ripple effects from changes within an object or in the system itself</td>
<td>• Fewer ripple effects from changes within an object or in the system itself</td>
</tr>
<tr>
<td>Inheritance</td>
<td>• Allows us to use classes as standard templates from which other classes can be built</td>
<td>• Less redundancy</td>
</tr>
<tr>
<td></td>
<td>• Standards and consistency within and across development efforts</td>
<td>• Faster creation of new classes</td>
</tr>
<tr>
<td></td>
<td>• Ease in supporting exceptions</td>
<td>• Standards and consistency within and across development efforts</td>
</tr>
<tr>
<td>Polymorphism</td>
<td>• Minimal messaging that is interpreted by objects themselves</td>
<td>• Ease in supporting exceptions</td>
</tr>
<tr>
<td></td>
<td>• Simpler programming of events</td>
<td>• Simpler programming of events</td>
</tr>
<tr>
<td></td>
<td>• Ease in replacing or changing objects in a system</td>
<td>• Ease in replacing or changing objects in a system</td>
</tr>
<tr>
<td></td>
<td>• Fewer ripple effects from changes within an object or in the system itself</td>
<td>• Fewer ripple effects from changes within an object or in the system itself</td>
</tr>
<tr>
<td>Use case driven</td>
<td>• Allows users and analysts to focus on how a user will interact with the system to perform a single activity</td>
<td>• Better understanding and gathering of user needs</td>
</tr>
<tr>
<td></td>
<td>• Better communication between user and analyst</td>
<td>• Better communication between user and analyst</td>
</tr>
<tr>
<td>Architecture centric and functional, static, and dynamic views</td>
<td>• Viewing the evolving system from multiple points of view</td>
<td>• Better understanding and modeling of user needs</td>
</tr>
<tr>
<td></td>
<td>• More complete depiction of information system</td>
<td>• More complete depiction of information system</td>
</tr>
<tr>
<td>Iterative and incremental development</td>
<td>• Continuous testing and refinement of the evolving system</td>
<td>• Meeting real needs of users</td>
</tr>
<tr>
<td></td>
<td>• Higher quality systems</td>
<td>• Higher quality systems</td>
</tr>
</tbody>
</table>
Unified Modeling Language – UML (Version 2)

- Defines a set of fourteen object diagramming techniques
- The key building block is the use case
- Diagrams are tightly integrated syntactically and conceptually to represent an integrated whole
- Application of UML can vary among organizations
# UML 2.0 Diagram Summary

<table>
<thead>
<tr>
<th>Diagram Name</th>
<th>Used to</th>
<th>Primary Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structure Diagrams</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class</td>
<td>Illustrate the relationships between classes modeled in the system.</td>
<td>Analysis, Design</td>
</tr>
<tr>
<td>Object</td>
<td>Illustrate the relationships between objects modeled in the system.</td>
<td>Analysis, Design</td>
</tr>
<tr>
<td></td>
<td>Used when actual instances of the classes will better communicate the model.</td>
<td></td>
</tr>
<tr>
<td>Package</td>
<td>Group other UML elements together to form higher level constructs.</td>
<td>Analysis, Design, Implementation</td>
</tr>
<tr>
<td>Deployment</td>
<td>Show the physical architecture of the system. Can also be used to show software components being deployed onto the physical architecture.</td>
<td>Physical Design, Implementation</td>
</tr>
<tr>
<td>Component</td>
<td>Illustrate the physical relationships among the software components.</td>
<td>Physical Design, Implementation</td>
</tr>
<tr>
<td>Composite Structure</td>
<td>Illustrate the internal structure of a class, i.e., the relationships among the parts of a class.</td>
<td>Analysis, Design</td>
</tr>
<tr>
<td><strong>Behavioral Diagrams</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Activity</td>
<td>Illustrate business workflows independent of classes, the flow of activities in a use case, or detailed design of a method.</td>
<td>Analysis, Design</td>
</tr>
<tr>
<td>Sequence</td>
<td>Model the behavior of objects within a use case. Focuses on the time-based ordering of an activity.</td>
<td>Analysis, Design</td>
</tr>
<tr>
<td>Communication</td>
<td>Model the behavior of objects within a use case. Focuses on the communication among a set of collaborating objects of an activity.</td>
<td>Analysis, Design</td>
</tr>
<tr>
<td>Interaction Overview</td>
<td>Illustrate an overview of the flow of control of a process.</td>
<td>Analysis, Design</td>
</tr>
<tr>
<td>Timing</td>
<td>Illustrate the interaction that takes place among a set of objects and the state changes which they go through along a time axis.</td>
<td>Analysis, Design</td>
</tr>
<tr>
<td>Behavioral State Machine</td>
<td>Examine the behavior of one class.</td>
<td>Analysis, Design</td>
</tr>
<tr>
<td>Protocol State Machine</td>
<td>Illustrates the dependencies among the different interfaces of a class.</td>
<td>Analysis, Design</td>
</tr>
<tr>
<td>Use Case</td>
<td>Capture business requirements for the system and to illustrate the interaction between the system and its environment.</td>
<td>Analysis, Design</td>
</tr>
</tbody>
</table>
Integration of four UML Diagrams

A Sequence Diagram is created for every use case.

A Class Diagram is created for the system.

A Behavioral State Machine Diagram is created for every complex class on the Class Diagram.
Adaptation of the Unified Process

Phased Development Methodology
Use Case Diagram Concepts

- Summarizes all use cases (for the part of the system being modeled) together in one picture
- Typically drawn early in the SDLC
- Shows the associations between actors and use cases
Use Case Diagram for Appointment System
## Syntax for Use Case Diagram

<table>
<thead>
<tr>
<th>Term and Definition</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>An actor</td>
<td>Actor role name</td>
</tr>
<tr>
<td>- Is a person or system that derives benefit from and is external to the system</td>
<td></td>
</tr>
<tr>
<td>- Is labeled with its role</td>
<td></td>
</tr>
<tr>
<td>- Can be associated with other actors using a specialization/superclass association, denoted by an arrow with a hollow arrowhead</td>
<td></td>
</tr>
<tr>
<td>- Are placed outside the system boundary</td>
<td></td>
</tr>
<tr>
<td>A use case</td>
<td>Use case name</td>
</tr>
<tr>
<td>- Represents a major piece of system functionality</td>
<td></td>
</tr>
<tr>
<td>- Can extend another use case</td>
<td></td>
</tr>
<tr>
<td>- Can use another use case</td>
<td></td>
</tr>
<tr>
<td>- Is placed inside the system boundary</td>
<td></td>
</tr>
<tr>
<td>- Is labeled with a descriptive verb–noun phrase</td>
<td></td>
</tr>
<tr>
<td>A system boundary</td>
<td>System name</td>
</tr>
<tr>
<td>- Includes the name of the system inside or on top</td>
<td></td>
</tr>
<tr>
<td>- Represents the scope of the system</td>
<td></td>
</tr>
<tr>
<td>An association relationship</td>
<td></td>
</tr>
<tr>
<td>- Links an actor with the use case(s) with which it interacts</td>
<td></td>
</tr>
</tbody>
</table>
Use Case Diagram for Specialized Actor

Appointment System

- Make appointment
  - Management
  - Patient

- Produce schedule information
  - Management
  - Patient

- Record availability
  - Doctor
  - New patient
Extends and Includes Associations

Diagram of an appointment system showing various actions and relationships.
Steps in Creating the Use Case Diagram

1. Identify Use Cases
2. Draw the system boundary
3. Place Use Cases on the diagram
   - Group Use Cases into packages
   - Add special Use Case associations
4. Identify the actors
5. Add associations
Elements of a Class Diagram

- A static model that shows the classes and relationships among classes that remain constant in the system over time.
- Resembles the ERD, but depicts classes which include both behaviors and states, while entities in the ERD include only attributes.
- Scope not system wide, but pertaining to a single Use Case.
Class Diagram for Manage Appointment
## Class Diagram Syntax

<table>
<thead>
<tr>
<th>Term and Definition</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A class</strong></td>
<td>![Class name symbol]</td>
</tr>
<tr>
<td>- Represents a kind of person, place, or thing about which the system must capture and store information</td>
<td></td>
</tr>
<tr>
<td>- Has a name typed in bold and centered in its top compartment</td>
<td></td>
</tr>
<tr>
<td>- Has a list of attributes in its middle compartment</td>
<td></td>
</tr>
<tr>
<td>- Has a list of operations in its bottom compartment</td>
<td></td>
</tr>
<tr>
<td>- Does not explicitly show operations that are available to all classes</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>An attribute</strong></th>
<th>![Attribute name symbol]</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Represents properties that describe the state of an object</td>
<td></td>
</tr>
<tr>
<td>- Can be derived from other attributes, shown by placing a slash before the attribute’s name</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>A method</strong></th>
<th>![Operation name symbol]</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Represents the actions or functions that a class can perform</td>
<td></td>
</tr>
<tr>
<td>- Can be classified as a constructor, query, or update operation</td>
<td></td>
</tr>
<tr>
<td>- Includes parentheses that may contain special parameters or information needed to perform the operation</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>An association</strong></th>
<th>![Multiplicity symbols]</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Represents a relationship between multiple classes, or a class and itself</td>
<td></td>
</tr>
<tr>
<td>- Is labeled using a verb phrase or a role name, whichever better represents the relationship</td>
<td></td>
</tr>
<tr>
<td>- Can exist between one or more classes</td>
<td></td>
</tr>
<tr>
<td>- Contains multiplicity symbols, which represent the minimum and maximum times a class instance can be associated with the related class instance</td>
<td></td>
</tr>
</tbody>
</table>
Operation Types

- Constructor operation: create new instances of a class
- Similar to relationships in ERDs
- Multiplicity shows how an instance of an object can be associated with other instances
## Multiplicity

<table>
<thead>
<tr>
<th>Instance(s)</th>
<th>Representation of Instance(s)</th>
<th>Diagram Involving Instance(s)</th>
<th>Explanation of Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exactly one</td>
<td>1</td>
<td><img src="image" alt="Department-Boss" /></td>
<td>A department has one and only one boss.</td>
</tr>
<tr>
<td>Zero or more</td>
<td>0..*</td>
<td><img src="image" alt="Employee-Child" /></td>
<td>An employee has zero to many children.</td>
</tr>
<tr>
<td>One or more</td>
<td>1..*</td>
<td><img src="image" alt="Boss-Employee" /></td>
<td>A boss is responsible for one or more employees.</td>
</tr>
<tr>
<td>Zero or one</td>
<td>0..1</td>
<td><img src="image" alt="Employee-Spouse" /></td>
<td>An employee can be married to zero or one spouse.</td>
</tr>
<tr>
<td>Specified range</td>
<td>2..4</td>
<td><img src="image" alt="Employee-Vacation" /></td>
<td>An employee can take between two to four vacations each year.</td>
</tr>
<tr>
<td>Multiple, disjoint ranges</td>
<td>1..3, 5</td>
<td><img src="image" alt="Employee-Committee" /></td>
<td>An employee is a member of one to three or five committees.</td>
</tr>
</tbody>
</table>
Steps in Creating a Class Diagram

1. Identify classes
2. Identify attributes and operations
3. Draw associations between classes
Initial Attributes for Class Diagrams

Marketing Material
- type
- description
- email
- content

CD
- sku
- title
- artist
- category
- salestatus

Inventory
- store
- zip code

Vendor
- name
- street address
- city
- state
- zipcode
- contact
- phone

Customer
- email
- lastname
- firstname
- address
- city
- state
- zipcode
- daytime phone

In-Store Hold
- date
Revised Attributes and Associations

- **Marketing Material**
  - type
  - description
  - email
  - content
  - provides -> 1

- **CD**
  - sku
  - title
  - artist
  - category
  - salestatus
  - describes -> 0..*

- **Inventory**
  - store
  - zip code
  - is stored in -> 0..*

- **Vendor**
  - name
  - street address
  - city
  - state
  - zipcode
  - contact
  - phone
  - provides -> 1

- **Customer**
  - email
  - lastname
  - firstname
  - address
  - city
  - state
  - zipcode
  - daytime phone
  - places -> 1

- **In-Store Hold**
  - date
  - is reserved by -> 0..*
Final Class Diagram

- Vendor
  - name
  - street address
  - city
  - state
  - zipcode
  - contact
  - phone

- Marketing Material
  - type
  - description
  - email
  - content

- Product
  - Number
  - Description

- CD
  - sku
  - title
  - artist
  - category
  - sale status

- Inventory
  - store
  - zip code

- Customer
  - email
  - lastname
  - firstname
  - address
  - city
  - state
  - zipcode
  - daytime phone

- In-Store Hold
  - date

- Relations:
  - Vendor provides Marketing Material
  - Marketing Material describes CD
  - Artist Info
  - Sample Clips
  - Reviews
  - Customer places In-Store Hold
SEQUENCE DIAGRAM
Sequence Diagram Concepts

- Illustrates the classes that participate in a use case
- Shows the messages that pass between classes over time for one Use Case
- Can be a generic sequence diagram, but more frequently one is drawn for a single scenario within the use case
- Design diagrams are implementation specific -- database objects or specific GUI components serve as classes
Sequence Diagram

- aPatient: Patient
- aReceptionist: Receptionist
- Patients: List
- UnpaidBills: List
- Appointments: List

Sequence Diagram Details:
- Request Appt()
- LookUpPatient()
- [aPatient Exists] LookupBills()
- MatchAppts()
- CreateAppt()
- anAppt: Appointment

Copyright 2009 © John Wiley & Sons, Inc. All rights reserved.
Steps in Creating a Sequence Diagram

1. Identify classes
2. Add messages
3. Place lifeline and focus of control
# Syntax for Sequence Diagram

<table>
<thead>
<tr>
<th>Term and Definition</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>An actor:</strong></td>
<td><img src="anActor.png" alt="Symbol" /></td>
</tr>
<tr>
<td>- Is a person or system that derives benefit from and is external to the system</td>
<td></td>
</tr>
<tr>
<td>- Participates in a sequence by sending and/or receiving messages</td>
<td></td>
</tr>
<tr>
<td>- Are placed across the top of the diagram</td>
<td></td>
</tr>
<tr>
<td><strong>An object:</strong></td>
<td><img src="" alt="Symbol" /></td>
</tr>
<tr>
<td>- Participates in a sequence by sending and/or receiving messages</td>
<td></td>
</tr>
<tr>
<td>- Is placed across the top of the diagram</td>
<td></td>
</tr>
<tr>
<td><strong>A lifeline:</strong></td>
<td><img src="lifeline.png" alt="Symbol" /></td>
</tr>
<tr>
<td>- Denotes the life of an object during a sequence</td>
<td></td>
</tr>
<tr>
<td>- Contains an X at the point at which the class no longer interacts</td>
<td></td>
</tr>
<tr>
<td><strong>A focus of control:</strong></td>
<td><img src="focus.png" alt="Symbol" /></td>
</tr>
<tr>
<td>- Is a long narrow rectangle placed atop a lifeline</td>
<td></td>
</tr>
<tr>
<td>- Denotes when an object is sending or receiving messages</td>
<td></td>
</tr>
<tr>
<td><strong>A message:</strong></td>
<td><img src="message.png" alt="Symbol" /></td>
</tr>
<tr>
<td>- Conveys information from one object to another one</td>
<td></td>
</tr>
<tr>
<td><strong>Object destruction:</strong></td>
<td><img src="destruction.png" alt="Symbol" /></td>
</tr>
<tr>
<td>- An X is placed at the end of an object's lifeline to show that it is going out of existence</td>
<td></td>
</tr>
</tbody>
</table>
Steps of the Customer Places
Order Scenario

1. User requests CD information
2. User inserts CD(s) into shopping cart
3. User confirms order and provides payment and shipping information
4. Order is sent to clearance center for approval
5. Customer is notified that order is accepted
6. Order is placed with the distribution system
Sequence Diagram for Customer Places
Order Scenario

1. aCustomer
2. aCD: CD
3. aMktMat: Marketing Material
4. anInv: Inventory
5. aSpOrder: Special Order

- Request CD (title, artist, or category)
- Request availability (CD ID)
- Add to Cart (CD ID)
- Check out (name and contact information)
- Order Accepted

Shopping Cart
BEHAVIORAL STATE MACHINE DIAGRAM
Behavioral State Machine Concepts

A dynamic model showing changes of state of a single class over time in response to events along with its responses and actions.

Typically not used for all classes, but just to help simplify the design of algorithms for methods of complex classes.
Behavioral State Machine
Diagram for a Hospital Patient

- Enters hospital
- Entering
- Checks in
- Admitted
- Under observation
- Released
- [Diagnosis = healthy]
- [Diagnosis = unhealthy]
- [Diagnosis = healthy]
- [> 2 weeks]
# Behavioral State Machine Syntax

<table>
<thead>
<tr>
<th>Term and Definition</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>A state</td>
<td></td>
</tr>
<tr>
<td>- Is shown as a rectangle with rounded corners</td>
<td></td>
</tr>
<tr>
<td>- Has a name that represents the state of an object</td>
<td></td>
</tr>
<tr>
<td>An initial state</td>
<td></td>
</tr>
<tr>
<td>- Is shown as a small filled-in circle</td>
<td></td>
</tr>
<tr>
<td>- Represents the point at which an object begins to exist</td>
<td></td>
</tr>
<tr>
<td>A final state</td>
<td></td>
</tr>
<tr>
<td>- Is shown as a circle surrounding a small solid filled-in circle (bull’s-eye)</td>
<td></td>
</tr>
<tr>
<td>- Represents the completion of activity</td>
<td></td>
</tr>
<tr>
<td>An event</td>
<td>Event name</td>
</tr>
<tr>
<td>- Is a noteworthy occurrence that triggers a change in state</td>
<td></td>
</tr>
<tr>
<td>- Can be a designated condition becoming true, the receipt of an explicit signal</td>
<td></td>
</tr>
<tr>
<td>from one object to another, or the passage of a designated period of time</td>
<td></td>
</tr>
<tr>
<td>- Is used to label a transition</td>
<td></td>
</tr>
<tr>
<td>A transition</td>
<td></td>
</tr>
<tr>
<td>- Indicates that an object in the first state will enter the second state</td>
<td></td>
</tr>
<tr>
<td>- Is triggered by the occurrence of the event labeling the transition</td>
<td></td>
</tr>
<tr>
<td>- Is shown as a solid arrow from one state to another, labeled by the event name</td>
<td></td>
</tr>
</tbody>
</table>
The Life of an Order

- The customer creates an order on the Web.
- The customer submits the order once he or she is finished.
- The credit authorization must be approved for the order to be accepted.
- If denied, the order is returned to the customer for changes.
- If accepted, the order is sent to the distribution system so that it can be shipped.
- The customer receives the order.
Steps for Creating a Behavioral State Machine Diagram

1. Identify the states
2. Identify the transitions
Behavioral State Machine Diagram for a Special Order

Customer checks out, creating Special Order

Submitted

[In inventory = yes]

Transfer Requested

Special Order shipped to store

Shipped

Special Order received by store and held for pickup

On-Hold

Customer purchases Special Order

On-Order

[In inventory = no]

Special Order shipped to store

Sold

Special Order closed
Many organizations are moving to the use of object-oriented techniques.

Objects are grouped into classes that share common properties and methods and arranged in a hierarchy.

Objects communicate by sending messages which trigger methods.
Summary

Major object-oriented modeling techniques include:
- Use Case diagrams
- Class diagrams
- Sequence diagrams
- Statechart diagrams