

Lecture 8: Introduction to OO Analysis (by example)

Kenneth M. Anderson
Object-Oriented Analysis and Design
CSCI 6448 - Spring Semester, 2002

Credit where Credit is Due

- Some material presented in this lecture is taken from section 2.2 of Maciaszek's "Requirements Analysis and System Design". © Addison Wesley, 2000

Goals for this Lecture

- Work through the tutorial in Section 2.2 of the textbook
 - Provides overview of OO Analysis
 - Provides insight into how UML is used

Tutorial in Analysis Modeling

- Tutorial provides an example of analysis modeling with respect to the task of "on-line shopping"
 - We will develop four types of models
 - The use case model
 - The state model
 - The behavior model
 - The state change model
- Slides will provide background information; book will provide diagrams

OnLine Shopping – Order Processing

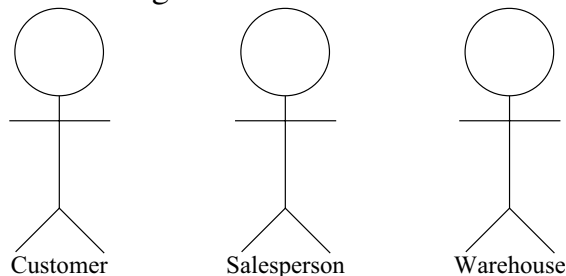
- Buying computers via Internet - page 47
- The customer can select a standard configuration or can build a desired configuration online
- To place an order, the customer must fill out the shipment and payment information
- The customer can check the order status online at any time
- The ordered configuration is shipped to the customer together with the invoice

Step 1: Find Use Cases

- The tutorial begins by talking about use cases
 - Each use case represents a complete unit of functionality that is required by an actor
 - An actor is any entity that interacts with our system; typically a human, but could also be an external software system
 - Since actors are external to the system, use cases document outwardly visible and testable system behavior

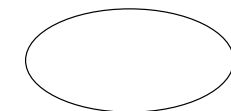
Actors

- Consider the requirement: After the **customer's** order has been entered into the system, the **salesperson** sends an electronic request to the **warehouse** with details of the ordered configuration



Use cases

- Consider the requirement
 - The customer uses the shopping web page to view the standard configuration of the chosen server, desktop or portable computer. The price is also shown



Display Standard
Computer Configuration

Use Cases, continued

- Some use cases may not interact directly with actors
 - Instead, they support other use cases
 - In particular, if several use cases each share a common task, it makes sense to encapsulate the common task in its own separate use case
- A use case diagram is a visual representation of actors and use cases
 - Note: UML diagram is synonymous with UML model

Step 1, continued

- The tutorial starts the process of modeling by transforming the problem statement (page 47) into a set of extended requirements (page 48-49)
 - These are not meant to be a complete list of requirements; simply a first set that can be derived directly from the problem statement

Step 1, continued

- Next, the requirements are placed in a table to help identify use cases and actors (pg. 50 and 51)
 - Useful for deriving an initial set of use cases;
 - more will be found later as information about the domain is discovered or as a result of working on other models
 - Some requirements lead to multiple use case

Step 1, continued

- Having identified use cases and actors, a use case diagram can be constructed
 - (page 52)
- A use case diagram is meant to show relationships between use cases and actors
 - Two types of relationships shown
 - association - a communication path
 - extend - a use case can define extension points where behavior may be customized; an extends relationships indicates that a use case is providing a customization of another use case

Narrative use case specification

- A use case diagram only show relationships between use cases
 - it does not provide any information about the details of a use case
 - thus, each use case needs to be documented textually (page 53); see next slide

Documenting use cases

- **Brief Description**
- **Actors** involved
- **Preconditions** necessary for the use case to start
- **Detailed Description** of flow of events that includes:
 - **Main Flow** of events, that can be broken down to show:
 - **Subflows** of events (subflows can be further divided into smaller subflows to improve document readability)
 - **Alternative Flows** to define exceptional situations
- **Postconditions** that define the state of the system after the use case ends

Step 2: Find Activities

- The textual information of a use case provides details about the flow of events that occur when carrying out its task
 - This information can be depicted graphically with an **activity model**

Activity Diagrams

- An **activity diagram** shows the steps of a computation
 - Each step is a **state** where the system is doing something; aka an **activity state**
 - activities take time to complete; if an activity completes quickly, it is referred to as an **action**
 - The diagram indicates the sequential and/or concurrent flow of activity states
 - The flow of control from one state to the next is called a **transition**

Step 2, continued

- To find activities, we examine the main and alternative flows of a use case
 - (page 55 shows this analysis for the `Order Configured Computer` use case; the outcome is shown on page 56)
 - Important: Use cases are written from the perspective of an **external** actor; activities however should be specified in terms of an **internal** system's viewpoint

Step 2, continued

- Once the activities have been found, we can display their interactions using an activity diagram (page 56)
 - There is one initial state but there may be multiple final states
 - Some transitions are guarded
 - The use of [timeout] indicates that the transition can only be taken if the timeout event has occurred
- To complete the activity modeling step, an activity diagram should be constructed for each use case!

Step 3: Find Classes

- The tutorial now begins to construct a **state model**, also known as a **class model**
 - State is represented via the classes of objects that the system contains, their attributes, operations, and relationships
 - These are shown in a **class diagram**
- Note: steps 2 and 3 are typically done in parallel; or via rapid iteration between the two
 - Use cases facilitate class discovery and vice versa, class models can lead to the discovery of overlooked use cases

Classes

- So far, we have used classes to define "business objects": `Order`, `Shipment`, `Customer`, etc.
 - aka entity classes (model), because they represent long-lived persistent database objects
- We also need other classes
 - those that define GUI objects, aka boundary classes (view)
 - those that control the program's logic, aka control classes
- Boundary and control classes may or may not be addressed in requirements analysis (since they are not a direct part of the application domain)
 - as such their specification may be delayed until design

Step 3, continued

- To find classes, the tutorial returns to the initial set of requirements
 - we first used them to define use cases, now we are looking for classes (pg. 58)
- This exercise may produce more classes than needed
 - these are referred to as **candidate classes**
 - we must ask questions to eliminate unnecessary classes

Classes: Asking Questions

- Is this a class?
 - Is the concept a container for data?
 - Does it have separate attributes that will take on different values?
 - Would it have many instance objects?
 - Is it in the scope of the application domain?
- See page 59

Step 3, continued

- Having found a set of initial classes, the next step is to determine the attributes for each class
 - Attributes define the structure of a class
 - “obvious” attributes are added immediately after the initial classes have been selected
 - other attributes will be discovered as analysis continues
 - See page 60

Step 3, continued

- Associations are added next (page 61)
 - Here it is critical to use the use cases to determine which classes need to communicate with each other
 - For instance, the *Order Configured Computer* use case came from a requirement that gave rise to the *Customer*, *Order*, and *ConfiguredComputer* classes;
 - this implies that these classes will need to share information

Step 3, continued

- Aggregations and Generalizations can be considered next
 - Are there classes which are “composed” of other classes
 - Are some of the candidate classes specializations or generalizations of other classes
- See page 62 (aggregations), 63 (generalizations), and 64 (completed class diagram) for the evolution of the tutorial’s class diagram

Step 4: Model Interactions

- Interaction modeling captures interactions between objects that need to be performed to satisfy a use case
 - This type of modeling occurs once the class diagram has stabilized
- Interaction modeling is similar to activity modeling but at a lower level of abstraction
 - Activity modeling shows the sequencing of events without assigning those events to objects
 - Interaction modeling shows the sequencing of events (messages) between collaborating objects

Interaction Diagrams

- Two Types
 - Sequence Diagram
 - focus is on events over time
 - Collaboration Diagram
 - focus is on object relationships
- Our textbook uses sequence diagrams in analysis and uses collaboration diagrams in design

Interactions

- An **interaction** is a set of **messages** (that define a behavior) exchanged between **objects** over **links**
- A **sequence diagram** represents an interaction with a two-dimensional graph
 - with **objects** arrayed **across the top**
 - **event sequences** shown **top to bottom**
 - events occur between object **lifelines** as a **message** from a **sender** to an **operation** in the **target** (actual parameters can be specified)

More on Sequence Diagrams

- Showing the return of control from the target to the calling object is not necessary
 - however, it is sometimes done to show return values;
- If a message needs to be sent to a collection of objects, the message name is prefixed with an asterisk, aka the iteration marker
 - this means that the indicated message is sent to each object in the collection

Step 4, continued

- The tutorial begins the iteration modeling activity by picking the first activity state (Display Current Configuration) of the activity diagram developed in step 2
 - and constructs a sequence diagram for it on page 66; a screen shot of a proposed user interface after this interaction has completed execution is also shown on page 66
- This diagram has led to the definition of operations on some of the identified classes (page 67)

Step 4, continued

- The tutorial then steps up a level of abstraction and constructs a sequence diagram for the entire activity diagram (pg 69)
 - Note how certain details from the first sequence diagram are hidden in the second diagram
 - Part of analysis is deciding how much detail is “enough” for the task at hand
 - You may start by drawing the second diagram first in a real analysis situation and then add detail to the sequence diagram as you progress through analysis (perhaps because you need more information about a particular class)

Step 4, continued

- To finish interaction modeling, develop sequence diagrams for each activity diagram
 - Activity diagrams model the details of use cases; these details help discover classes
 - Sequence diagrams further elaborate the details of activity diagrams; these new details help to discover class operations
- Interaction modeling can be incremental
 - you may only need to construct a few sequence diagrams at first; return later to construct more

Step 5: Model Object States

- A state chart model gives a detailed description of a class
 - In much the same way that an interaction model provides a detailed specification of a use case
- In particular, a statechart depicts how objects of a particular class change state over time
 - these state changes will typically describe the behavior of an object across multiple use cases
 - state, in this sense, is defined as the values of an object's attributes

Statechart Diagram

- A statechart diagram is a graph of states (rounded rectangles) and transitions (arrows) caused by events
 - These “states” and “events” are the same concepts that we know from activity diagrams, except
 - an activity diagram shows the states of executing a computation
 - while a statechart diagram documents the states of a single object

States and transitions

- Objects change values of their attributes but not all such changes cause state transitions
 - Bank account example on page 70
 - We construct state models for classes that have “interesting” state changes, not just any state changes
- The tutorial shows a possible statechart diagram for the Invoice object on page 71
 - again transitions are labeled with events

Statechart Diagram

- Normally attached to a class, but can be attached to other modeling concepts, e.g. a use case
- When attached to a class, the diagram determines how objects of that class react to events
 - Determines – for each object state – what action the object will perform when it receives an event
 - The same object may perform a different action for the same event depending on the object's state
 - The action's execution will typically cause a state change

Statechart Diagram

- The complete description of a transition consists of three parts
 - event (parameters) [guard] / action
 - event - a message, can have parameters
 - guard - transition can only occur if guard is true (otherwise the event is ignored)
 - Action – short atomic computation that executes when the transition fires
 - can also be associated with a state, e.g. the action executes when the state is entered (or exited)
 - Activity – longer computation associated with a state

Statechart Diagrams, continued

- States can be composed of other states, aka nested states
 - The composite state is abstract, it is simply a generic label for the nested states
 - A transition taken out of the composite state's boundary means that it can fire from any of the nested states
 - this helps to avoid cluttered diagrams

Step 5, continued

- The tutorial develops a state diagram for the Order class (page 72)
 - It contains nested states, guarded transitions, transitions from nested states, and one action
- Statechart modeling continues until all classes with “interesting” states have been modeled with a statechart diagram

Step 6? Iterate!

- Having performed these steps, its time to iterate
 - each new model may reveal “missing” information in the previous models
 - Analysis continues until all models have stabilized
 - At which point you are ready to move on to design
- What's Next?
 - Over the next few weeks, we will look at each of these analysis steps (and their associated diagrams) in more detail