Student Originated Software

OO Analysis & Design (OOAD) Workshop Exercise 6

Fall 2001

Due Monday, Oct. 29

Defining Sequence Diagrams and the initial design Class Model

Completing the Interaction diagrams (i.e., developing Sequence Diagrams from your Collaboration Diagrams) allows you to do two important things: (1) define the detailed behavior of your objects, and (2) find appropriate homes for this behavior in terms of operations in the classes, along with the attribute detail for those operations. The result of this activity is a first cut at the design of the system's Class Model -- i.e., the artifact from which system code will be produced for a variety of implementation languages (in our case, Java).

Major Goals

- To complete the design of the system's dynamic behavior, in the form of a set of Sequence Diagrams.
- To design an initial Class Model, allocating and refining messages from the Sequence Diagram into operations of appropriate classes.
- To add necessary design detail to the attributes of the classes in the Class Model.

Refer to the background material you have for the "Surplus Stock Disposal" system (EU-Bid) -- in particular, the Collaboration Diagram for "**Launch Auction**" -- when you complete the steps outlined below.

Steps

(1) Define Sequence Diagrams.
(2) Distribute operations among classes and define methods.
(3) Complete the attribute detail of the entity classes of the Class model.

Details

(1) Define Sequence Diagrams.

In our development process, Sequence Diagrams represent the major work product of the *dynamics* side of design. A Sequence Diagram shows the detailed interactions that occur over time among the objects associated with each of the use cases. Objects interact by sending messages to each other. These messages serve as what Ivar Jacobson calls 'stimuli' -- that is, a message *stimulates* an object to perform some desired action. For each 'unit of behavior' within a use case, you will identify the necessary messages and responses (i.e., the operations and, eventually, the method underlying each operation).

In this workshop we will develop the Sequence Diagram for "**Launch Auction**."

1a Define a sequence diagram (from the collaboration diagram)

Give the sequence diagram the same name as the collaboration diagram. (In other words, this will be "Launch Auction.")

1b Add in the actors and objects.

The actor(s) and collaborating objects (typically depicted as "anonymous classes") are drawn across the top of the diagram, from left to right, in generally this order: the

 ©ARCorp, 2001

initiating actor, the boundary class the actor 'talks' to, the controller class(es) the boundary class interacts with; the collaborating entity classes; and (if needed): the outbound-side controller(s), boundary class(es), and actor(s).

1c Name the actors and objects.

Give each actor and object a name, following the guidelines explained in the lecture.

1d (optional) Add the text from the scenario.

Copy the text from the scenario down the left side of the diagram. Since this depicts only the interactions between the actor and the boundary object(s), leave sufficient white space for the interactions that happen behind the interface.

1e Draw the messages between the 'client' (requesting) object and the 'server' (responding) object. Consider what the client needs to provide to the server object for it to do what it is being requested to do.

Each of the entity objects is an instance of a class that appears in the Domain Model. As such, these objects should already have most of their attributes identified. Many of these objects will, in turn, be serving data to other objects as the interactions flow, so you can expect to discover some missing attributes as you work through a Sequence Diagram. If you do, be sure to add them into your Domain Model (and Glossary), as you discover them.

1f Name the messages.

Since each message will become an operation in a class, name the message in a verb+noun form that reflects the perspective of the server (not the client). (It can be helpful to think in terms of the object and say "I know how to _____" or "I can be asked to _____" when you develop these names.)

(2) Distribute operations among classes and define methods.

In our development process, the Class Model represents the major work product of the *statics* side of design. We will focus, in this lesson, only on the entity objects of the Sequence Diagrams.

You should have had all of your domain entities and about 90 percent of your attributes defined from the Domain Model at the end of your analysis of the collaborations. In this step, the Domain Model entities are transformed into design classes of the Class Model, with the operations supplied from the Sequence Diagrams.

2a Draw an initial design Class Model from the Domain Model.

Apply the following transforms to the constructs of the Domain Model to produce a Class Model. For each entity:

1. In the first compartment of the Class Model class:
   - Entity name: any 'white space' characters are removed and the class name is written in *TitleCase*, beginning with an upper-case letter.
2. In the second compartment of the Class Model class:
   - Attribute names: any 'white space' characters are removed and the attribute name is written in *TitleCase*, beginning with a lower-case letter.

©ARCorp, 2001

- Relationship names: the name of the related-to class (or its rolename, if assigned) is written in *TitleCase*, beginning with a prefix of "my" followed by the class (or role) name. For a "many" relationship, add "s" to indicate a Collection.

2b *Write each method from the Sequence Diagram as an operation in the third compartment of the Class Model.*

Each message to an entity object on the Sequence Diagram becomes an operation of the class of that object. If you are using an automated tool (such as Rose), this step is already done (as a result of Step-1).

2c *Complete the operation signature.*

If the operation uses arguments, list each and assign its type. If the operation returns a value, specify this.

2d *Describe the operation.*

Write a brief description of the operation.

(3) Complete the attribute detail of the entity classes of the Class model.

In this step, the remaining design-level detail is filled-in for the attributes.

For each attribute in the Class Model:

3a *Define its visibility (typically "private").*
3b *Define its type.*
3c *State its initial value (if any).*
3d *Specify its containment property -- typically, "by value" rather than "by reference."*
3e *Is it static? (default is "no")*
3f *Is it derived? (default is "no")*

©ARCorp, 2001