Student Originated Software

OO Analysis & Design (OOAD) Workshop Exercise 5

Fall 2001

Due Monday, Oct. 22

Understand the collaborating objects that realize each use case

At this point, we have analyzed the static aspects of the system (as the ***core concepts***, reflected in the **Domain Model**) and the dynamics aspects of the system (as the ***required functionality***, reflected in the **Use Cases**). We are ready to begin the transition from analysis -- the 'what' -- to design -- the 'how.'

We do this by investigating the ways that the systems objects will collaborate behind the scenes ("behind the interface") to *realize* each of the use cases. We can document this work in two ways: as a simple matrix and as a Collaboration Diagram.

Major Goals

- To perform a sanity check on your use cases -- confirm that the behavior in your use case text is reasonable (and possible) given the set of objects you have to work with.
- To continue to refine the system classes -- discovering classes, attributes, and relationships that may have been missed during domain modeling.
- To transition to design -- beginning to develop the Class Model.

Refer to the background material you have for the "Surplus Stock Disposal" system (EU-Bid), as needed, when you complete the steps outlined below. As you develop these new artifacts, also continue to capture any new terms and their definitions in the system glossary -- and to improve on existing terms and definitions as you gain a better understanding of the system.

Steps

(1) Develop an initial "C-R-U-D" Matrix that depicts the relationships between the system's entity classes and use cases.
(2) Refine the "C-R-U-D" Matrix into a Collaboration Matrix that briefly describes the nature of each entity's role in the collaboration that realizes the use case.
(3) Draw this understanding as a Collaboration Diagram for each use case.
(4) Update the Domain Model, as needed.

Details

(1) Develop an initial "C-R-U-D" Matrix that depicts the relationships between the system's entity classes and use cases.

The history of systems analysis has long used this simple technique of mapping the static and dynamics models through the use of an artifact called a C-R-U-D (**C**reate, **R**ead, **U**pdate, **D**elete) matrix between the constructs of the two model types. Applying the CRUD approach, we examine each use case to discover its participating objects.

1a Draw a matrix (table) that list the domain classes across the top and the use cases down the left side.

1b *If an object of the class participates in a use case, note the type of participation in the appropriate cell (class/use case intersection) using the following codes:*

C -- for Create: does the use case create a new instance (object) of this class?
R -- for Read/Retrieve: does the use case access this object for the purpose of using any of its property knowledge, i.e., its attribute or relationship values.
U -- for Update: does the use case update any of the properties (attributes or relationships) of the object?
D -- for Delete: does the use case delete an object of this class from the active system?

Note: multiple codes may appear in a cell.

(2) Refine the "C-R-U-D" Matrix into a Collaboration Matrix that briefly describes the nature of each entity's role in the collaboration that realizes the use case.

Although a C-R-U-D matrix provides a quick mapping of the relationship between the use cases and the objects that participate in each use case, it does not reveal the nature of that participation in a form that reflects an understanding of the responsibilities of each object. In this step we expand the C-R-U-D matrix format to capture more details of the specific responsibilities, producing a Collaboration Matrix.

2a *Replace each letter (C, R, U, D) with a short description of the object's responsibility(s) in the use case.*

Write this in simple verb-noun form, using terms from the glossary/domain model, e.g., "calculate credit-score," "create a new loan account," "update loan-amount," "delete an order."

Multiple items may be called for in a cell. List in bullet format.

(3) Draw this understanding as a Collaboration Diagram for each use case.

******* Draw a Collaboration Diagram for "Launch Auction" **** Additional Collaboration Diagrams are optional.***

The UML has two diagram types to depict this kind of "interaction" graphically: sequence diagrams and collaboration diagrams. Sequence diagrams are where we are heading (as the final artifact of design, on the dynamics side). But we will first draw a simple form of the collaboration diagram which is much simpler to draw and easier to read -- meaning that this diagram type can be drawn and redrawn quickly as refinements are made.

This technique applies the popular Model-View-Controller pattern and introduces the other analysis class stereotypes we mentioned earlier: the boundary class and the controller class.

For each use case/actor pair:

3a *Carry forward the participating entity object(s).*

Use the entity objects you identified in the Collaboration Matrix.

3b *Define boundary object(s).*

Boundary objects are the objects in the system with which the actors will interact. These will become the windows, dialogs, etc. (in a GUI interface) or the system-to-system interface in a batch/backend interaction (with a non-human actor).

Name the boundary classes appropriately. (We will use a standard "_UI" suffix for boundary classes that interact with human actors and "_SI" suffix for boundary classes that interact with system actors.)

3c Define controller object(s).

Controller objects are the "glue" between the boundary objects (the user interface) and the entity objects (the system's persistent data). They embody much of the application's business logic (processing "business rules").

Name the controller classes appropriately. (We will use a standard "_Cntrl" suffix.)

3d Add a symbol for each actor.

3d Connect and label the symbols to reflect the flow of processing called for by the use case..

Apply the following rules when making connections.

1. Actors can talk only to Boundary objects.
2. Boundary objects can talk only to Controllers or Actors.
3. Entity objects can talk only to Controllers or Entities.
4. Controllers can talk to anyone except Actors.

(4) Update the Domain Model, as needed.

While you perform collaboration analysis, it is likely that you will learn things that will cause the Domain Model to be updated.

4a Add elements to the Domain Model.

In thinking about the classes' responsibilities it is very likely that you will have discovered attributes, relationships, and even entity classes that were not on your initial Domain Model. Add these into the model, also making the necessary additions/changes to the Glossary.

4b Consider removing elements from the Domain Model.

If you find that you have something on the Domain Model that was not exercised during this activity, this may suggest that you have things in your model that are outside the requested scope. Review these with your system sponsors to confirm the need for these elements and, if not needed (say, for future/other functionality), then remove them from your model.