A Heuristic for Developing Object Interaction Diagrams

Il-Yeol Song
Drexel University, USA
Songiy@Drexel.edu

Abstract

The UML (Unified Modeling Language) has been widely accepted as a standard language for object-oriented analysis and design. Among the UML diagrams, one of the most difficult and time-consuming diagrams to develop is the object interaction diagram (OID), which is rendered as either a sequence diagram or a collaboration diagram. Our experience shows that developers have significant trouble in understanding and developing OIDs. In this paper, we present an effective heuristic for developing interaction diagrams and illustrate the technique with a case study. We found that students effectively developed OIDs using this heuristic method.

Keywords: UML, object-oriented design, interaction diagram, sequence diagram, collaboration diagram

Introduction

The UML (Unified Modeling Language) claims to be a language, rather than a method. The UML provides a set of notations and concepts that are necessary for developing object-oriented software or systems. Among the UML diagrams, one of the most difficult and time-consuming diagrams to develop is the object interaction diagram (OID), which is rendered as either a sequence diagram or a collaboration diagram. OIDs model dynamic behavior by showing how system components interact to complete core tasks defined in use case design (Booch, Rumbaugh, Jacobson, 1999). While many novice designers put emphasis upon static models, they often fail to emphasize the use of dynamic models, which are very important for properly allocating responsibility among objects (Larman 1998). The purposes of interaction diagrams are (Booch et al., 1999; Rosenberg, 1999; Eriksson & Penker. 1998, Larman, 1998; Yourdon, Whitehead, Thomann, Oppel, and Nevermann, 1995):

- Use to model interactions between objects.
- Assist in understanding how a system (a use case) actually works.
- Verify that a use case description can be supported by the existing object classes
- Identify responsibilities/operations and assign them to classes

While seemingly intuitive, methods for constructing an OID have not been described in literature. Our teaching experience shows that students have significant trouble in understanding and developing OIDs. Based on the author's many years' teaching object-oriented analysis and design, we show an effective heuristic for developing interaction diagrams and illustrate the technique with a case study.

Our heuristic assumes that the developer has already developed use case diagrams; use case descriptions, and the class diagram. We develop interaction diagrams based on each primary use case. For the rest of the paper, we show how to develop sequence diagrams for each use case. Since a sequence diagram can be easily converted into a collaboration diagram, our heuristic can be equally applied to developing collaboration diagrams.

For the rest of the paper, we briefly summarize the notation of sequence diagrams in Section 2. The heuristics are presented in Section 3, while the case study is presented in Section 4. Section 5 concludes our paper.

Sequence Diagrams and Their Development

The popularity of the sequence diagram, originally called an object interaction diagram, is attributed to Jacobson et al. (1992). A sequence diagram focuses on time sequencing or time ordering of messages or the order in which messages are sent. The emphasis in these diagrams is what happens first, second, and so on. They represent the passage of time graphically. These diagrams have two axes: the horizontal axis displays the objects and the vertical axis shows time. In addition, sequence diagrams have two features not present in collaboration diagrams: an object's lifeline and the period it is active (Booch et al., 1999).

Object lifelines are used in the sequence diagram to represent the existence of the object during a scenario. While most objects will be in existence during the entire scenario, at times objects are created or deleted during the scenario (Booch et al, 1999). For example a transaction or order could be created and a reservation could be deleted.

The limitation of the sequence diagram is that it does not explicitly show the relationships or links between objects. These relationships are the primary emphasis of the collaboration diagram. One of the goals behind the development of the UML was to keep it as simple as possible while still being able to model the spectrum of systems that needed to be built (Booch et al., 1999). However it is more complicated than previously developed object-oriented methods, because it is intended to be more comprehensive. As a result, the UML diagrams are often difficult to develop. In the case of developing the first draft of an interaction diagram, there are a lot of important design decisions to make. The interaction diagram may also be difficult to develop because the UML does not provide a process or specific steps that can be followed to produce an effective diagram. As previously discussed, it is up to the designer to choose or develop a method that will assist him or her in creating an effective diagram. We found very few authors who even mentioned possible methods, processes or steps that could be used to develop effective UML diagrams.

Amber (1998) outlines his suggestions for the steps for constructing a sequence diagram. His steps describe, in a very brief form, what needs to be done to put together the main parts of the diagram. However these steps do not give any type of detailed instruction on "how." Rosenberg (1999) discusses the development of sequence diagrams using robustness analysis, which was introduced by Jacobson et al. (1992). However, Rosenberg, does not include step-by step guidelines for a developer to follow.

Booch et al (1999) outline separate steps for constructing the sequence diagram, as well as the collaboration diagram. While some of the steps overlap, there are different steps that represent the differences in the two diagrams. These steps are more detailed than Ambler's but still concentrate on what needs to be done rather than addressing 'how' it can be done. Booch et al (1999) also outline a number of characteristics for a well-structured interaction diagram and a number of tips for developing an interaction diagram. While not specific, these are very helpful suggestions, especially for the novice and should be given some attention to.

Heuristics for Developing Sequence Diagrams

For easy presentation of our paper, we have divided our method into two steps: pre-step and application step. The pre-step summarizes the necessary work that needs to be done to apply our heuristic. The application step shows the details of the heuristic.

Pre-Steps:

- Develop a problem statement
- Develop a use case diagram
- Develop use case descriptions for major primary use cases
- Develop a class diagram for the problem
- Develop pre-conditions and post-conditions for each primary use case

Heuristic on Sequence Diagram Development

1. Select the initiating actor and initiating external event from the use case description.

2. Identify the primary display screen needed for implementing the use case. Call it the Primary boundary stereotype object.

3. Introduce a use-case controller (control stereotype) to handle communication between boundary stereotype object and domain objects.

4. Identify the number of major screens necessary to implement the use case. Create one helper boundary stereotype object for each of the major screens.

5. From the class diagram, list all domain classes participating in the use case by reviewing the use case description. If any class identified from the use case description does not exist in the Class Diagram, add it to the class diagram.

6. Use those classes just identified as block labels (Column names) in the sequence diagram. List classes in the following order:

   6.1. The primary boundary stereotype

   6.2. Use case controller

   6.3. Domain classes (list in the order of access)

7. Identify all major operations based on the following classifications:

   7.1. Instance creation and destruction

   7.2. Association forming

   7.3. Attribute modification:

      7.3.1. Calculation

      7.3.2. Change States

      7.3.3. Any reporting requirements

      7.3.4. Interface with external objects or systems

8. Order/rearrange the sequence of messages among the object classes for implementing the use case based on design patterns

9. Name each message and supply it with optional parameters.

Case Study

In this section, we illustrate our heuristic. We apply the heuristics to a video rental system and develop sequence diagrams for the rent items use case.

Problem Statement

This is about a small, local video rental store (VRS). The problem will be limited to rental, return, management of inventory (add/delete new tapes, change rental prices, etc.), and producing reports summarizing various business activities. The rental items of the store are limited to video tapes. Customer ID number (arbitrary number), phone number or the combination of first name and last name are entered to identify customer data and create an order. The bar code ID for each item is entered and video information from inventory is displayed. The video inventory file is decreased by one. When all tape IDs are entered, the system computes the total rental fee and payments are processed. The rental form is created, printed, and stored. The customer signs the rental form, takes the tape(s), and leaves. A return is processed by reading the bar code of returned tapes.

Any outstanding video rentals are displayed with the amount due on each tape and a total amount due. The past-due amount must be reduced to zero when new tapes are taken out. For new customers, the unique customer ID is generated and the customer information is entered into the system. Videos are stacked by their category such as Drama, Comedy, Action, etc. Any conflict between a customer and computer data is resolved by the store manager. Rental fees can be paid by either cash, check, or a major credit card. Reporting requirements include viewing customer rental history, video rental history, titles by category, top 10 rentals, items by status, overdue videos by customers, and outstanding balances by customers.

Use Case Diagram

The use case diagram for the above VRS is shown in Figure 1.

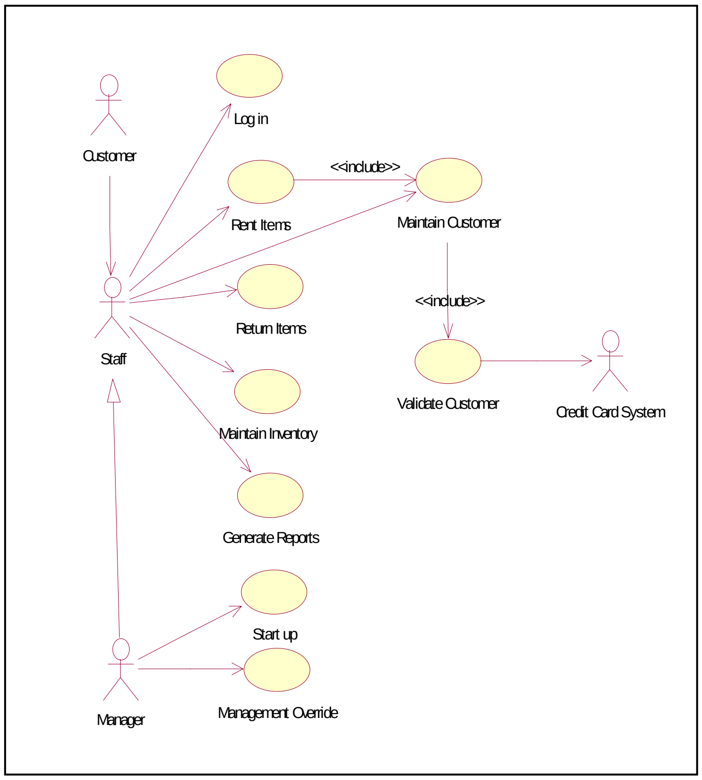

Figure 1. Use case diagram for VRS

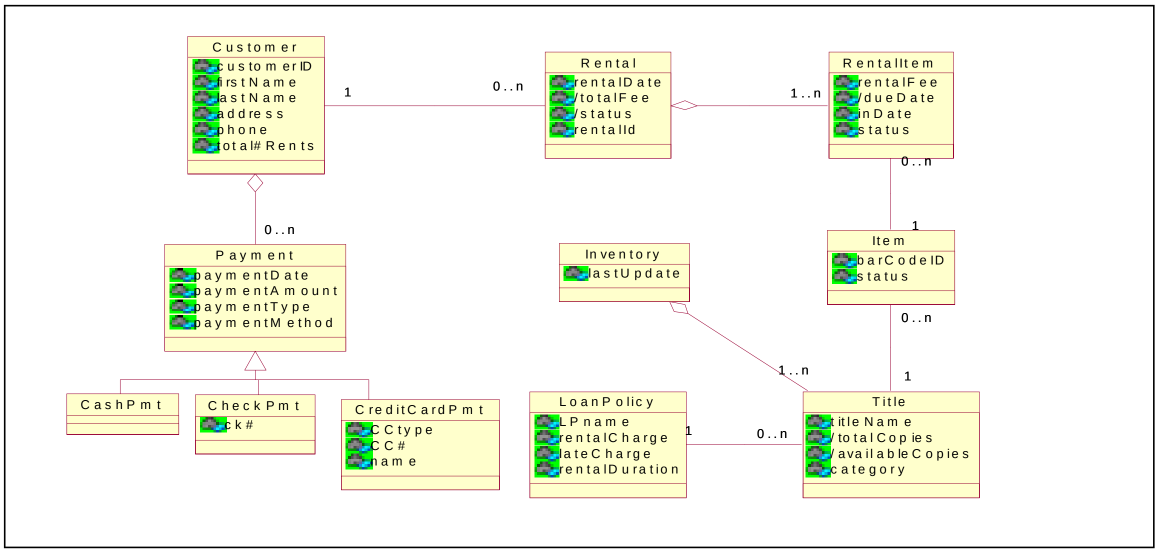

Figure 2. The class diagram for VRS

Class Diagram

The class diagram for the above VRS is shown in Figure 2. Note that we assumed that one item can be associated with zero or more RentalItems. This is because we wanted to keep all the rental data for six months.

Application of Heuristics

A sequence diagram can easily become complicated as the complexity of the problem domain increases. In order to reduce the complexity of the sequence diagram, we use the idea of a system sequence diagram used by Larman (1998).

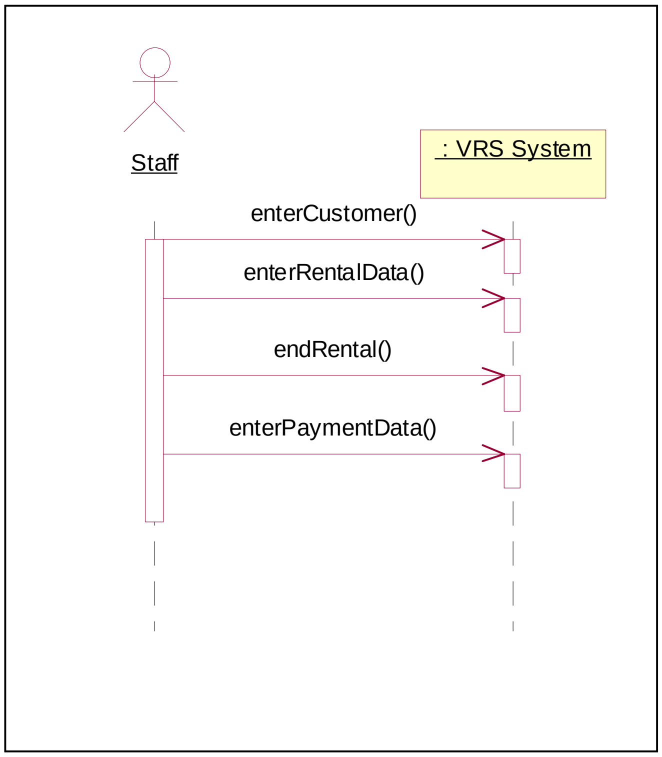

Figure 3. The system sequence diagram for *Rent Items* use case

Actor:			Staff
Primary boundary stereotype object.			Rental window
Use-case controller:			Rental handler
Other major display screens			None
Participating domain classes			Rental, Rental item, Item, Title, Payment, LoanPolicy
Block labels			Put Rental window object first, then Rental Handler followed by domain classes in the Diagram
Major operation			
	Instance creation:		Rental handler Rental, RentalItem
	Associate forming:		Connect RentalItem to Item Connect RentalItem to Rental Connect Rental to Customer Connect Payment to Customer
	Attribute forming:		
		Calculation:	CalculateDueDate(), calculateTax() calculateTotalRental(),
		Change States:	updateItemStatus(), decreaseAvailableCopies(), setDueDate(), setRentaldate(),
		Display/reporting requirements:	getItem(), getTitle(), getRentalData(), getIDuration(), getFee(), getRentalFee(), displayTotalFee()
Interface with external objects or systems:			None
Order the sequence:			can be done at the diagram level
Name each message properly with obvious parameters:			can add later

Table 1. The application of the heuristic

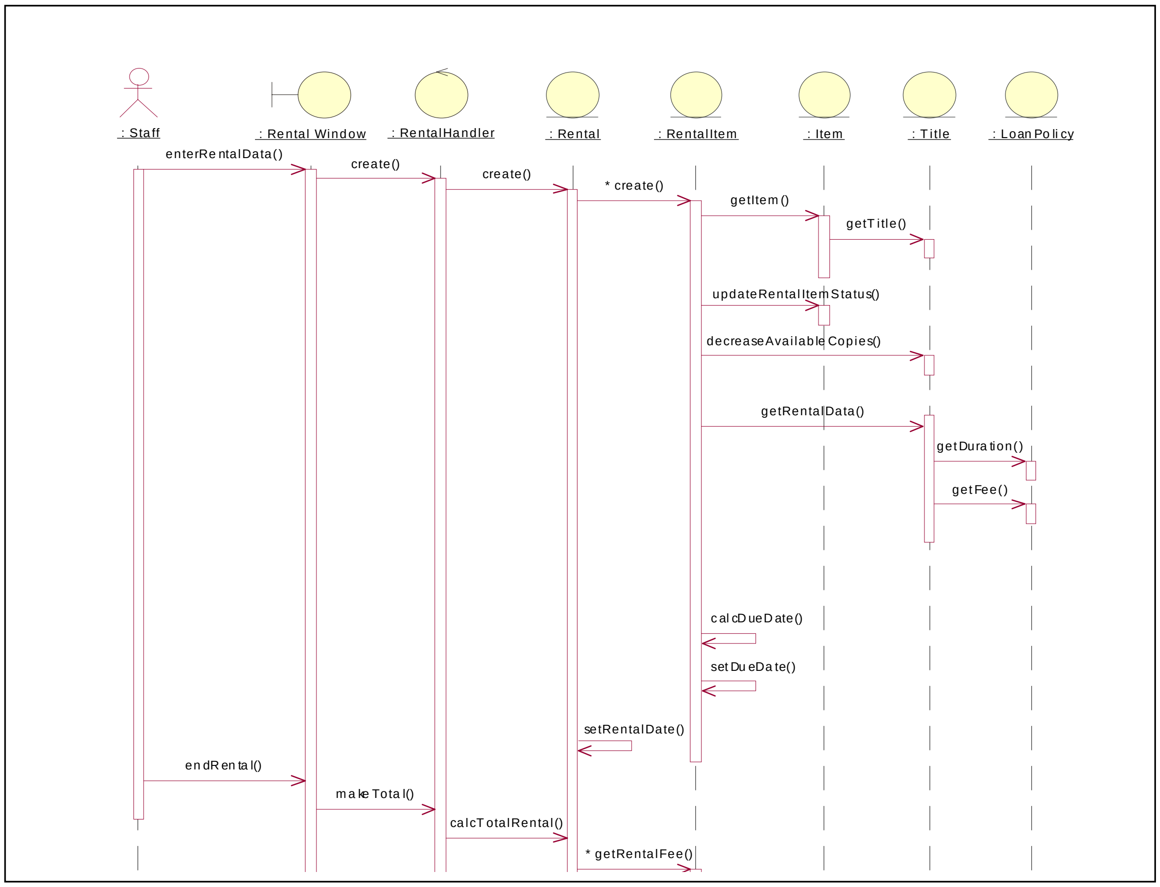

Figure 4. Sequence diagram for *Enter Rental Data* system event

A system sequence diagram shows all the system events between the system actor and the system as a black box (Larman 1988). A system event is an input that is generated by an actor to the system. Using this approach, an entire sequence diagram for a use case is decomposed into a set of system events allowing us to develop one sequence diagram for one or more system events. Figure 3 shows the system sequence diagram of VRS having four system events.

For lack of space, we only apply our heuristics for the second and third system events, enterRentalData() and endRental(), in developing our sequence diagram. Note that customer data will be handled in the enterCustomer() event, and payment will be created and processed in the enterPaymentData() event. Thus, they will not be shown in our sequence diagram in Figure 4.

Conclusion

In this paper, we have presented a 9-step heuristic for developing sequence diagrams. We believe that the technique presented in this paper is highly applicable regardless of problem domains and can be easily customized to the specific application. Our experience shows that students who used this method developed sequence diagrams easily and quickly.

Reference

Ambler, S. (1998) Focus on UML: How the UML Models Fit Together Software Development. Available at http://www.sdmagazine.com/uml/focus.ambler.htm

Booch, G., Rumbaugh, J., and Jacobson, I (1999). The Unified Modeling Language: User Guide. Addison Wesley,

Eriksson, H. and Magnus P. (1998). UML Toolkit. New York: John Wiley & Sons, Inc.

Jacobson, I., Christerson, M., Jonsson, P., and Overgaard, G.. Object-Oriented Software Engineering: A Use Case Driven Approach. Harlow, England: Addison-Wesley, 1992.

Larman, C., Applying UML and Patterns, Prentice Hall, 1998.

Rosenberg, D. (1999). Use Case Driven Object Modeling with UML: A Practical Approach, Addison Wesley.

Yourdon, E., Whitehead, K., Thomann, J., Oppel, K., and Nevermann, P. Mainstream Objects: An Analysis and Design Approach for Business. Upper Saddle River, NJ: Yourdon Press, 1995.

Biographies

Il-Yeol Song is a professor of Drexel University since 1988. Prof. Song has authored over 75 papers on the subject of database design, object-oriented analysis & design, and data warehousing. Prof. Song co-chaired ACM CIKM '99, DOLAP '98 and DOLAP '99.