Developing Sequence Diagrams in UML

Conference Paper · September 2001

DOI: 10.1007/3-540-45581-7_28 · Source: CiteSeer

1 author:

Il-Yeol Song
Drexel University
295 PUBLICATIONS **3,321** CITATIONS

SEE PROFILE

ER2001

Developing Sequence Diagrams in UML

Il-Yeol Song
College of Information Science and Technology
Drexel University
Philadelphia, PA 19104
song@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 ten-step heuristic for developing sequence diagrams and illustrate the technique with a case study. In this technique, we show a proper use of control objects and boundary objects when developing sequence diagrams. In our heuristic the relationships among multiple sequence diagrams in a single use case are elegantly represented using control objects. We found that developers effectively developed sequence diagrams using this heuristic method.

1 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. The UML includes nine inter-related diagrams which are used to model different aspects of the system being modeled. The relationships among the diagrams are shown in Fig. 1. 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 [2]. 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 [8]. The purpose of interaction diagrams is to [6, 2; 14; 5, 8; 19, 3, 10, 15]:

- 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,
- Model synchronous/asynchronous message passing in real-time systems

While seemingly intuitive, methods for constructing an OID have not been described much in literature. Our experience shows that novice developers have significant trouble in understanding and developing OIDs. Most UML books simply explain the notations and semantics of OIDs and present pre-built sequence diagrams.

Some authors provide simple guidelines for developing sequence diagrams. We found that those simple guidelines are not sufficient for many novice developers. Based on the author's several years of experience in teaching object-oriented analysis and design, we show an effective heuristic for developing interaction diagrams and illustrate the technique with a case study.

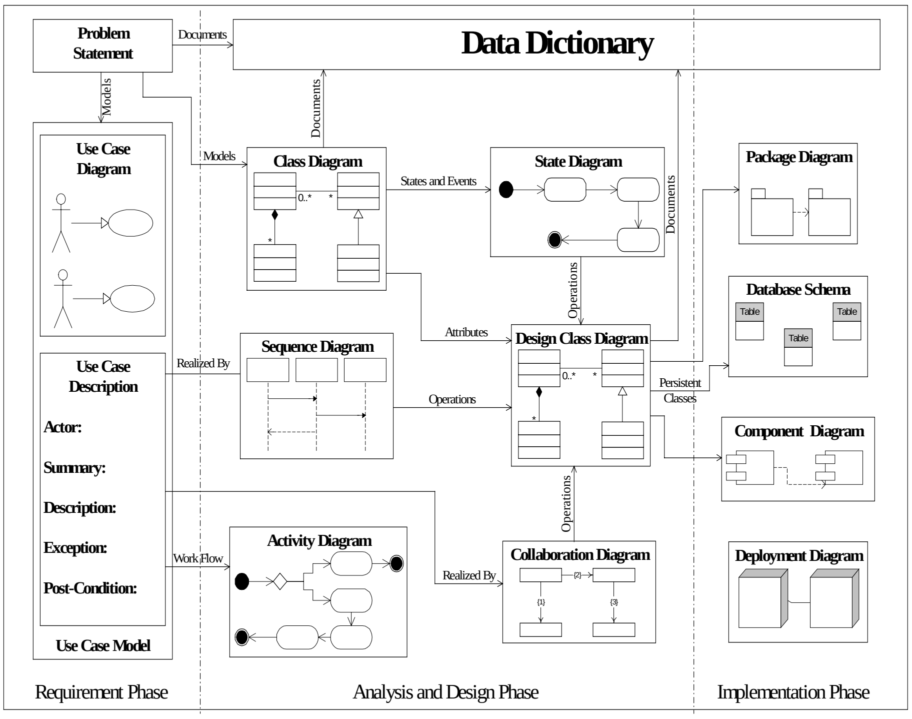

Fig. 1. Relationships among UML diagrams

Ideally, 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. Our heuristic takes advantage of properties of stereotype objects – entity, boundary and control objects. We assume that an actor only directly communicates with a boundary object, but not with a control or an entity object. We create a control object for each primary, included or extended use case. This convention allows us to elegantly represent the relationships among multiple sequence diagrams in a single 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 our paper, we use the terms *interaction diagram* and *sequence diagram* interchangeably.

The rest of the paper is organized as follows: Section 2 summarizes the notation and concepts of sequence diagrams. Section 3 reviews research activities in the areas of sequence diagrams and its development methods suggested by various authors. Section 4 describes the use of stereotype objects and their relationships. Section 5 presents the heuristics, and Section 6 illustrates the heuristics with a case study in a video rental system. Section 7 concludes our paper.

2 Sequence Diagrams

In this section, we define terminology and notation used in the sequence diagram in UML.

The popularity of the sequence diagram, originally called an object interaction diagram, is attributed to Jacobson et al. [6]. 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. The schematic structure and notation of a typical sequence diagram is shown in Fig. 2.

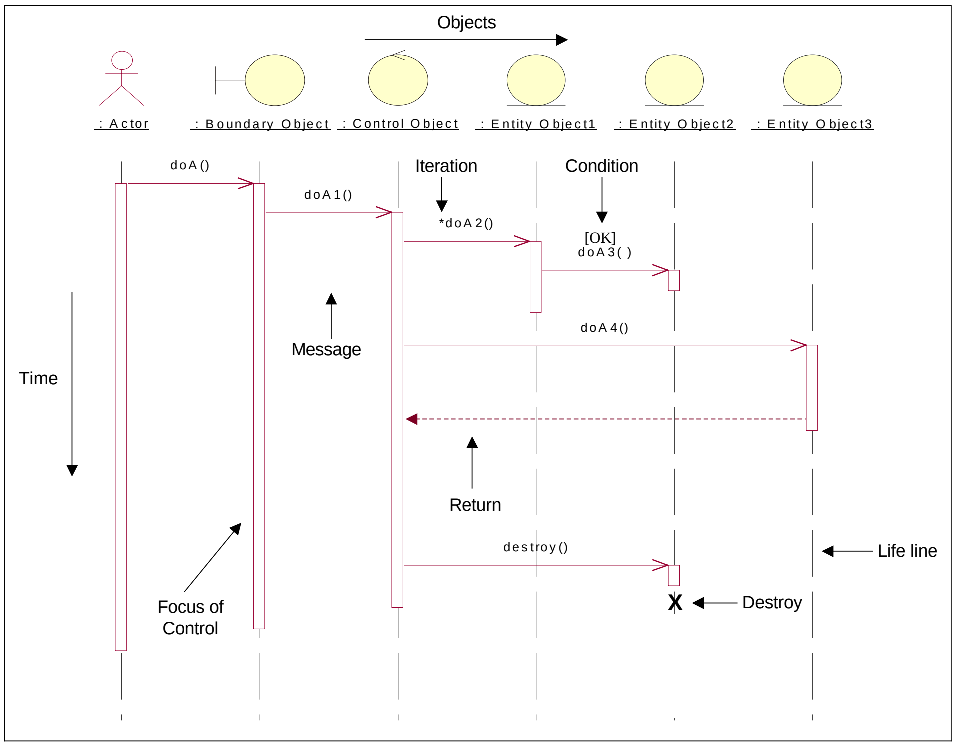

Fig. 2. A structure and notation of the sequence diagram

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 focus of control [2]. 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 scenarios. For example, an 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 [2]. 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. While developing the first draft of a sequence diagram, there are a lot of important design decisions to make. The sequence diagram may also be difficult to develop because the UML does not provide a process or specific steps that can be followed. It is up to the designer to choose or develop a method that will assist him or her in creating an effective diagram.

Booch et al [2] indicates that the sequence diagrams are a means to model some aspect of the dynamic behavior of the system and can be used in context of the whole system, a sub-system or they can be attached to a use case. Some indicate that a sequence diagram should be drawn, at least per use-case [6, 19, 5, 18]. When a sequence diagram is developed for a use case, the use case descriptions can be used to develop at least the initial draft of a sequence diagram. Throughout the design process the use case diagram can be revised based on the results of the sequence diagram, and vise versa, until both models are tuned appropriately [8, 10]. We identify operations from the use case description and represent them in sequence diagram (see Fig. 1).

Since a sequence diagram may be developed for each primary use case, *included* use case and *extended* use case, there will be a number of sequence diagrams developed. However, the UML does not define notations that connect these diagrams. Song et al proposes an idea for such a notation [17]. The suggestion is to insert a connector symbol on a sequence diagram that 'drills down' to a subordinate sequence diagram or 'rolls up' to a calling one. In this paper, we propose the use of control objects to connect multiple sequence diagrams.

3 Survey on the Sequence Diagram and Its Developments

In this section, we survey research activities on the sequence diagram and guidelines for developing the diagram suggested by various authors.

We found that most research activities on sequence diagrams have focused on real time systems [16, 9, 13, 4], simulation [7] or behavior-driven analysis and design [3]. Very few authors even mentioned possible methods, processes or steps that could be used to develop effective sequence diagrams.

Amber [1] outlines his suggestions for the steps for constructing a sequence diagram, as summarized in Table 1. 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."

Table 1. Ambler's Steps for Constructing a Sequence Diagram [1, p.62]

1.	Identify the class in which the use-case scenario starts.
2.	Walk through the process logic of the scenario, identifying each message that needs to be sent and the object to which it is sent.
3.	Once an object is identified, draw its lifeline.
4.	Indicate the activity box on the object's lifeline where it sends a message or is expecting a return.

Rosenberg [14] discusses the development of sequence diagrams using robustness analysis, which was introduced by Jacobson et al. [6]. However, Rosenberg, does not include step-by-step guidelines for a developer to follow.

Booch et al [2] outlines separate steps for constructing the sequence diagram (See Table 2), 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 [2] also outlines a number of characteristics for a well-structured interaction diagram (Table 3) and a number of tips for developing an interaction diagram (Table 4). While not specific, these are very helpful suggestions, especially for the novice and attention should be given to them.

Table 2. Steps for Constructing a Sequence diagram by Booch et al. [2, p. 251]

1.	Set the Context of the Interaction Diagram (System, Sub-system, Use Case).
2.	Identify the objects which play a role in the interaction and place the most important ones on the left.
3.	Set the lifeline of each of the objects.
4.	Starting with the message that initiates the interaction, layout each message from top to bottom between lifelines, showing each message's properties.
5.	Adorn to each lifeline the objects 'focus of control' or activation period, if this visualization is necessary.
6.	Adorn each message with time and space constraints, if needed.
7.	Adorn each message with pre- and post-conditions, if needed.

Table 3. A well-structured Interaction Diagram by Booch et al. [2, p. 256]

1.	Is focused on communicating one aspect of a system's dynamics.
2.	Contains only those elements that are essential to understanding that aspect.
3.	Provides detail consistent with its level of abstraction and should expose only those adornments that are essential to understanding.
4.	Is not so minimalist that it misinforms the reader about semantics that are important.

Table 4. Tips for drawing an Interaction Diagram by Booch et al. [2, p. 256]

1.	Give it a name that communicates its purpose.
2.	Lay out its elements to minimize lines that cross.
3.	Use notes and color as visual cues to draw attention to important features of your diagram.

Jacobson et al [6] categorizes sequence diagrams by structure. The *fork* or centralized sequence diagram contains one primary client operation that controls the flow of signals to multiple server operations. In contrast, the *stair* or decentralized sequence diagram uses delegation as the primary means for structuring communications among multiple objects. More specifically, Jacobson et al. describes a decentralized OID as one where “each object only knows a few of the other objects and knows which objects can help with a specific behavior. Here we have no ‘central’ object.”

Jacobson recommends the more encapsulated stair configuration for situations where the communicating objects are connected to each other in some hierarchical fashion. On the other hand, a centralized fork structure is recommended for times when operations can change a sequence of operations and when new operations can be inserted into the sequence diagram. See Fig. 8.15 in [6] for the two structures.

When developing sequence diagrams, we found that these guidelines were not sufficient for many novice developers. Our experience shows that novice developers have significant trouble in understanding and developing sequence diagrams. Based on the author’s experience in teaching object-oriented analysis and design, we have developed an effective ten-step heuristic for developing sequence diagrams. We illustrate the technique with a case study in Section 5.

4 Boundary Objects and Control Objects

Since our heuristic uses the notions of boundary objects and control objects, we briefly present their definitions and properties.

It was Jacobson et al [6] who first introduced entity, control and boundary objects for developing a robust system architecture. The idea was to distribute the behaviors specified in a use case description into those three object types. See Fig. 2 for the notations of the three object types. We adopt the definition and properties of these objects from [6, 19, 14].

Entity object classes represent real-life domain objects or concepts that are internal to the system. Examples are Customer, Rental and Video classes. External actors usually have no direct contact with entity objects. Instead they are accessed through boundary objects. *Boundary* object classes handle the communication with external actors, and they encapsulate environmental-dependent behavior, thereby protecting the integrity of the entity object. Examples are windows, screens and menus that are used for input and output. *Control* object classes are transaction classes that capture a sequence of operations. These objects capture business rules and policies. Jacobson et al [6] and Larman [8] use the control object to handle a use case. A control object is also frequently called a Handler or a Controller [8, 14].

As in [19, 14], the relationships among those three objects can be summarized as in Fig. 3. An actor communicates only with a boundary object. A control object bridges between a boundary object and an entity object. A boundary object is not recommended directly to communicate with an entity object. An entity object is not recommended to send a message directly to a boundary object as it increases coupling with specific interface of a system [8].

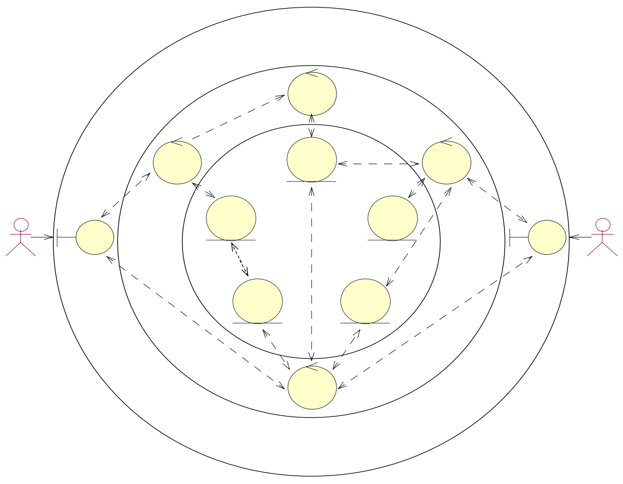

Fig. 3.. Relationships Among Boundary Objects, Control Objects, and Entity Objects.

The UML refers to these categories as *stereotypes* [2]. We note that entity object classes are eventually become persistent whereas controls and boundary objects classes do not usually become persistent. Entity, boundary, and control stereotypes will play a major role in Section 5 when the heuristic is described.

5 The Ten-Step Heuristic for Developing Sequence Diagrams

In this section, we present our heuristic for developing sequence diagrams.

Our methodology is based on the following ideas:

- An actor only directly communicates with a boundary object, but not with a control or entity object.
- We create a control object for each primary, included or extended use case.
- We create a boundary object for each pop-up screen in GUI.

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.

We assume that post-conditions are developed in the form of contracts. We use Larman's categories to specify post-conditions as follows [8]:

- Instance creation and deletion.
- Attribute modification.
- Association formed and broken.

The ten-step heuristic for developing sequence diagrams is shown in Table 5. We note that association forming in Step 8.2 is done when an object is created. Thus, an association forming does not require a separate message passing. Steps 8.3.1-8.3.4 provide developers with tips for identifying various possible operations.

6. Case Study

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

6.1 The 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 an item is checked out. 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 ten rentals, items by status, overdue videos by customers and outstanding balances by customers.

Table 5. The Ten-Step Heuristic on Sequence Diagram Development

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

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

3 Create a *use-case controller* (*primary control object*) to handle communication between the primary boundary object and domain objects.

4 If the use case involves any included or extended use case, create one *secondary control object* for each of them.

5 Identify the number of major screens necessary to implement the use case. Create one *secondary boundary object* for each of the major screens and create one *secondary control object* for each of them.

6 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.

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

- The primary boundary stereotype
- The primary use case controller
- Domain classes (list in the order of access)
- Secondary control objects and secondary boundary objects in the order of access

8 Identify all problem-solving operations based on the following classifications:

8.1 Instance creation and destruction
8.2 Association forming
8.3 Attribute modification:
8.3.1 Calculation
8.3.2 Change States
8.3.3 Display or reporting requirements
8.3.4 Interface with external objects or systems

These problem-solving operations can be identified by:

- Identify verbs from the use case description
- Remove verbs used to *describe* the problem; select verbs used to solve the problem. We call these verbs *problem-solving verbs(PSVs).*
- From the problem-solving verbs, select verbs that represent an automatic operation. We call these PSVs *problem-solving operations(PSOs)* and use them in the sequence diagram.

9 Rearrange the sequence of messages among the object classes based on any pre-existing design patterns, when possible.

10 Name each message and supply it with optional parameters. This can be done at design stage as well.

6.2 Use Case Diagram

A system-level use case diagram for the above VRS is shown in Fig. 4. In this use case diagram, we have identified four actors and eight primary use cases and two included use cases. Among them, *Customer* is an indirect actor who does not use the system. Only actors *Staff* and *Manager* use the system. Actor *Credit card system* is a supporting actor that receives a message and responds to the message.

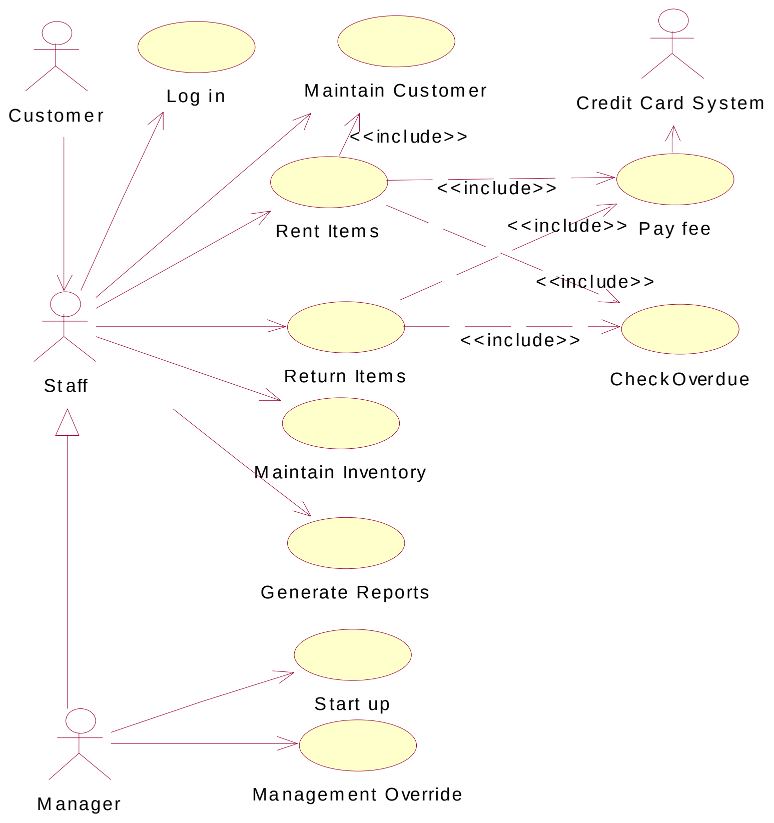

Fig. 4. Use case diagram for VRS

6.3 Class Diagram

A class diagram for the above VRS is shown in Fig. 5. Note that we assumed that one *Item* can be associated with zero or more *RentalItem*s. This is because we wanted to keep all the rental data for six months before archival.

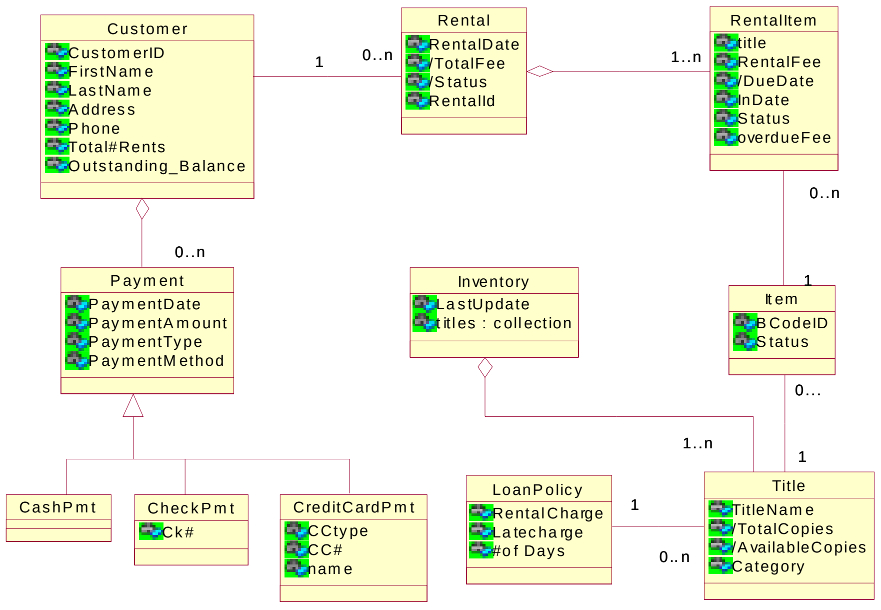

Fig. 5. The class diagram for VRS

6.4 Application of the Heuristic and the System Sequence Diagram

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 [8]. A system sequence diagram shows all the system events between the system actor and the system as a black box. 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. Fig. 6 shows the system sequence diagram of VRS having four system events.

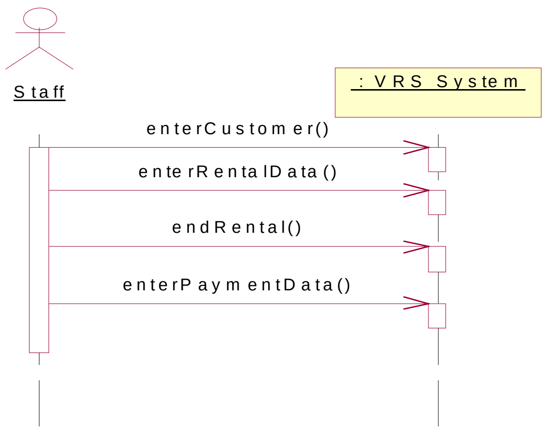

Fig. 6. The system sequence diagram for *Rent Items* use case

For lack of space, we only apply our heuristics for the second, third, and fourth system events: *enterRentalData()*, *endRental()* and *enterPaymentData()*. Note that customer data will be handled in the system event *enterCustomer()*, and payment object will be created and actual payment is processed in the included use case P*ayfee*. Thus, they will not be shown in our sequence diagram below.

1. Actor	Staff
2. *Primary boundary* object.	Rental window
3. *Use-case controller*	Rental handler
4. Secondary control object	Payfee
5. Secondary boundary object	None
6. Participating domain classes	Rental, Rental item, Item, Title, Payment, LoanPolicy
7. Block labels	Put Rental window object first, then Rental handler followed by domain classes
8. Major operations	
8.1 Instance creation:	Rental handler, Rental, RentalItem
8.2 Associate forming:	Connect RentalItem to Item Connect RentalItem to Rental Connect Rental to Customer
8.3 Attribute forming:	
8.3.1 Calculation:	calculateDueDate(), calculateTax() calculateTotalRental(),
8.3.2 Change States:	updateItemStatus(), decreaseAvailableCopies(), setDueDate(), setRentaldate(),
8.3.3 Display/reporting requirements:	getItem(), getTitle(), getRentalData(), getlDuration(), getFee(), getRentalFee() displayTotalFee(),
8.3.4 Interface with external objects or systems: None	
9. Rearrange the sequence: None in this case	
10. Name each message properly with obvious parameters: can add later during design stage	

The constructed sequence diagram is shown in Fig. 7. Note that we develop a separate sequence diagram for each included or extended use case and represent each of included/extended use case by a secondary control object. Thus, we represented the included use case P*ayfee* (Fig. 4) by a secondary control object *PmtControlle*r. The primary control object *RentalHandler* calls the secondary control object *PmtController*. In this way, connections among multiple included/extended use cases within a primary use case can be elegantly represented by message-passings between control objects. The sequence diagram for an included use case will begin with a secondary control object. For example, the sequence diagram for P*ayfee* will begin with the *PmtController*.

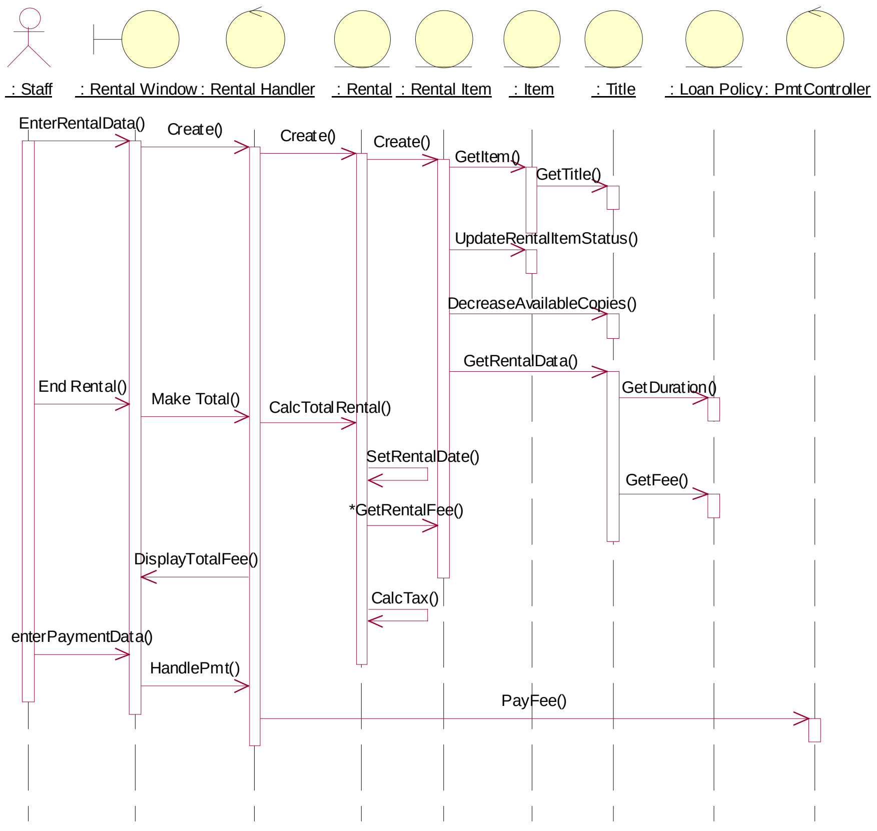

Fig. 7. The sequence diagram for System *Events enterRentalData(), endrental()* and *enterpaymentData()*

7 Conclusion

We notice that novice developers have significant trouble in understanding and developing sequence diagrams. Most UML books simply explain the notations and semantics of sequence diagrams and present pre-built sequence diagrams. Some authors provide simple guidelines for developing sequence diagrams. We found that those simple guidelines are not sufficient for many novice developers. In this paper, we have presented the ten-step heuristic for developing sequence diagrams. We do not claim that the ten-step heuristic presented in this paper always produces the best sequence diagram for all the situations. However, we believe that the technique is highly applicable regardless of problem domains. We also believe that the draft diagram produced by our heuristic can be easily customized and fine-tuned to the specific application. Our experience shows that students who used this method developed sequence diagrams easily and quickly. Even though we assumed that the developer had already developed use case modeling and descriptions, our heuristic can be easily applied without detailed use case modeling. We are currently developing fine-tuning techniques that can be applied after developing an initial sequence diagram using our heuristic.

References

[1] Ambler, S. *Building Object Applications That Work*, SIGS Books, 1998

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

[3] Delcambre, L.M.L. and Eckland, E., A Behaviorally driven Approach to Object-Oriented Analysis and Design with Object-Oriented Data Modeling, In *Advances in Object-Oriented Data Modeling*, MIT Press, 2000, pp. 21-40.

[4] Engels, G., Groenewegen, L., and Kappel, G. Coordinated Collaboration of Objects, In *Advances in Object-Oriented Data Modeling*, MIT Press, 2000, pp. 308-331.

[5] Eriksson, H. and Penker, M. (1998). *UML Toolkit*. New York: John Wiley, Inc.

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

[7] Kabajunga, C and Pooley, R.. Simulating UML sequence diagrams. *UK Performance Engineering Workshop*, UK PEW 1998, pages 198—207, July 1998.

[8] Larman, C., *Applying UML and Patterns,* Prentice Hall, 1998.

[9] Li, X. and Lilius, J. Timing Analysis of UML Sequence Diagrams. *UML'99, The Unified Modeling Language. Beyond the Standard*. The Second International Conference, Fort Collins, CO, USA, October 28-30, 1999.

[10] Maciaszek, L. A., *Requirement Analysis and System Design: Developing Information Systems with UML*. Addison Wesley, 2001.

[11] Pooley, R. and Stevens, P., *Using UML: Software Engineering with Objects and Components*. Addison Wesley, 2000.

[12] Quantrani, T. *Visual Modeling with Rational Rose and UML*. Addison Wesley, 1998.

[13] Rudolph, E., Grabowski, J., and Graubmann, P. Towards a Harmonization of UML-Sequence Diagrams and MSC. *SDL'99 - The next Millenium*, Elsevier, June 1999.

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

[15] Siau, Keng (2001). *Unified Modeling Language: Systems Analysis, Design and Development Issues*. Idea Publishing Group.

[16] Seemann, J. and Wvg, J. (1998). Extension of UML Sequence Diagrams for Real-Time Systems, In Proc. *International UML Workshop*, Mulhouse, June 1988.

[17] Song, I.-Y., Watts, P., Hassell, L., and Wong, C. "Modeling Dynamic Behavior with Object Interaction Diagrams," *Proc. of 4th International Conference on Computer Science and Informatics (CSI '98)*, Oct. 23-28, 1998, pp. 408-412.

[18] Texel, P. and Williams C. *Use Cases Combined with Booch/OMT/UML: Process and Products.* Upper Saddle River: Prentice Hall PTR, 1997.

[19] 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.