Robust Object Oriented System Analysis

Dr Jie Zhao, Dunstan Thomas Consulting

Summary

Uses cases are widely accepted as the best approach to capturing system requirements, in particular, functional requirements. However, an extremely important step towards a final implementation is how use an object oriented analysis method to translate the customers' needs into system software.

This article introduces a robust, systematic, use case driven object-oriented system analysis method that encompasses three components: Process, Modelling Language and a CASE Tool. In particular, we will look at finding classes (boundary, control and entity), allocating system behaviours to classes and producing class and sequence diagram to document the results of the analysis using the Unified Modelling Language or UML.

The example used in this article is an On-line Shopping Cart web application.

OO System Analysis Method

The OO system analysis method consists of three parts: Process, Modelling Language and a CASE tool. The process serves as the project roadmap that defines who (roles) is doing what (activities performed and artifacts produced), when (time order of activities) and how (guidelines, templates). Rational Unified Process (RUP) implements software development best practices and is a use case driven, architecture centric and iterative development process. It provides a framework that can be (and more often should be) customised to suit the size, complexity and other characteristics of the project.

Due to its iterative nature, activities performed in different phases and iterations vary. Figure 1 shows a customised system analysis process UML activity diagram during early iterations of the elaboration phase.
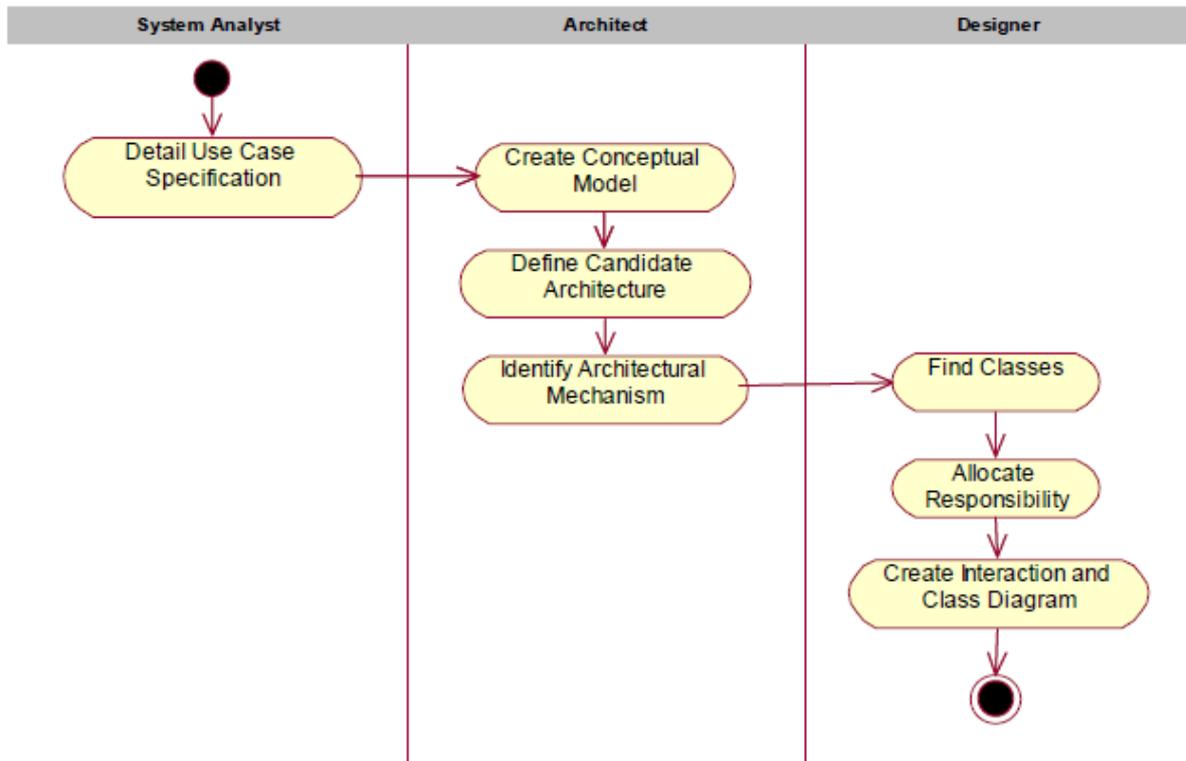
[Figure 1. Early Elaboration Phase Process Activity Diagram](#)

UML allows us to document, visualize and specify system and guide the developers to implement the solution. UML, as the only industrial standard visual modelling language, has been adopted widely in software industry since 1997. It unifies the previous modelling languages (such as Rumbaugh, Jacobson and Booch) and makes it possible for everyone in the project to use the same modelling language.

CASE (Computer Aided Software Engineering) tools are used to implement the UML modelling language. We can use them to create various diagrams, generate reports, perform cross reference checking, generate source code from model (forward engineering), update model from source code (reverse engineering) and so on. In this article we choose Rational Rose as the tool for performing the example system analysis.

Detail Use Case Specification

Before we start to explain this activity, let's have a look at what has already been produced during the inception phase: (1) use case model survey report that contains the use case diagram (Figure 2) and brief descriptions of the actors and use cases to capture the overall system functional requirements. (2) supplementary specification document for the non-functional requirements.
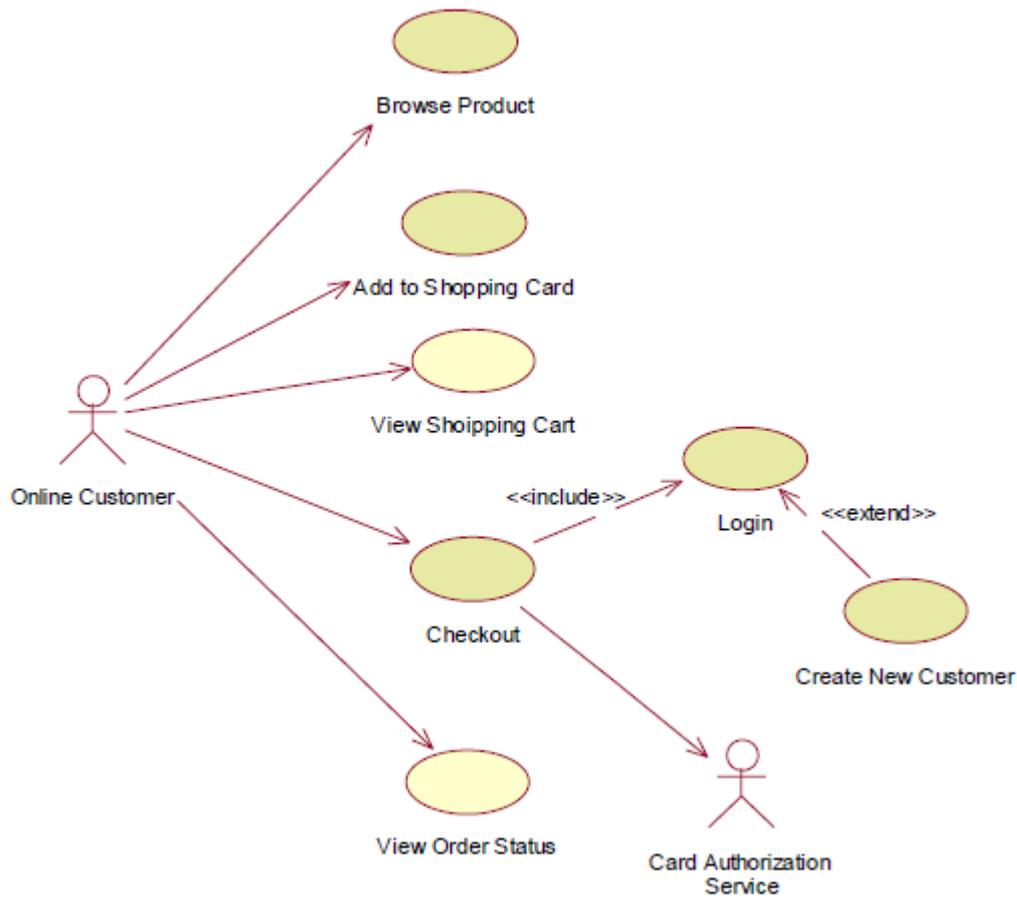
Figure 2. On-line Shopping Cart Use Case Diagram

Figure 3. Checkout Realization Traceability Diagram

Let's assume that in this iteration the *Checkout* use case will be realized according to the project plan as shown in the traceability diagram Figure 3. The system analyst needs to specify the details of the *Checkout* use case, i.e., flow of events.

The following is the basic flow of *Checkout* use case specification:

Checkout Use Case Specification

Basic Flow

This use case starts when the User indicates that he/she wishes to check out.

1. Include "Login" use case.
2. The system examines the contents of the shopping cart and displays an itemized list of all of the items in the shopping cart.
3. The system calculates the subtotal, VAT, shipping cost and the total value, then displays.
4. The customer confirms to continue the checkout process; the system asks the customer to enter payment details and shipping address.
5. The customer selects credit card payment method and enters credit card type, card number, cardholder's name and expiry date.
6. The customer enters shipping address if it is different from the address the customer entered when creating new customer.
7. The system generates a credit payment request and sends it to an external card authorization service.
8. The card authorization service authorizes the payment; the system receives a credit payment approval reply from card authorization system and indicates authorization success.
9. The system saves the customer's order into database and displays the order summary and the use case ends.

Create Conceptual Model

Conceptual model is a class diagram to capture the domain knowledge. Based on the understanding and experience the architects we can identify some domain concepts and their relationship. This provides a good starting point for the individual use case analysis. Figure 4 is the conceptual model of the example online shopping cart system.
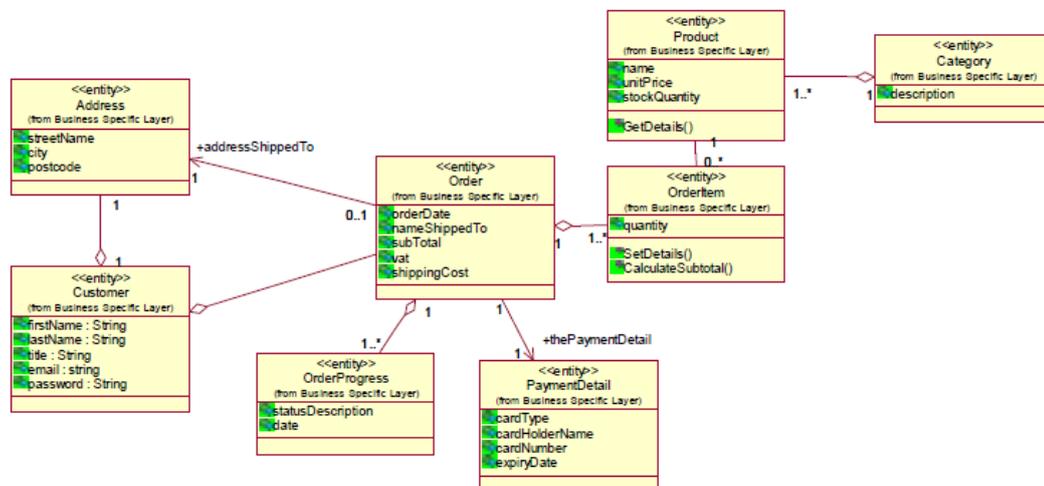

Figure 4. Online Shopping Cart Conceptual Model

Define Candidate Architecture

To avoid the use of vertical components and promote reuse, we separate different logical parts of the system into what is sometimes referred to as a layered architectural pattern. This consists of a presentation layer, a business layer and a data service layer (Figure 5). The user interface components exist in presentation layer, business logic classes will be in business layer and the classes providing data creation, access, update, delete functionalities will reside in data service layer. Since only the upper layers depend on the lower layers the components inside the lower layers, such as Business Specific Layer and Data Service Layer can be reused.
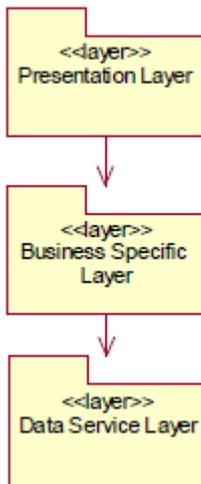
Figure 5. On-line Shopping Cart System Architecture

Identify Architecture Mechanism

Architecture mechanisms are system-wide strategies to deal with the common problems encountered by designers. They are project or organisational specific design patterns. By studying the requirements documents, we have identified following architectural mechanisms for the on-line shopping card system:

- Persistence: handles database related issues
- Security: user authentication and authorization, secure communication

Find Classes

How to find a group of classes that realise the behaviour specified in the use case flow of events is one of the most difficult tasks in object oriented analysis and design. There are three types of analysis classes:

- Boundary class
- Control class and
- Entity classes

These provide an easy to learn and consistent approach to identify the analysis classes from the use case model. In terms of UML notation, they are stereotyped classes with their own specific icons as shown in Figure 5. It is also possible to use UML class notation with stereotype label.
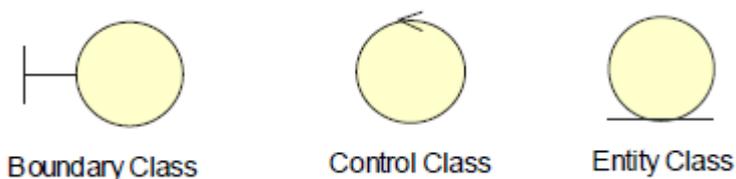

Figure 6. Three types of analysis classes

Boundary Class

Represents the interfaces between the actors and the system. Depending upon the type of the actor, a boundary class is required to provide a user interface, external system (legacy system) interface or device interface. In a web application the user interface boundary classes represent the web pages such as in ASP.NET, *.aspx pages.

Control Class

Represents the use case logic and coordinates the other classes. It separates the interface classes from business logic classes. In ASP.NET, the code-behind classes can be regarded as Control classes.

Entity Class

Manages the information the system needs to provide the required functionality. Usually entity classes are business specific. They are information experts and encapsulate the business knowledge. Most of the time, entity classes are persistent classes that can be used to generate database schema directly. Now the question is, how we can identify boundary, control and entity classes from use case specification flow of events?

The guidelines are:

- One Boundary Class per actor/use case pair
- One Control Class per use case
- Noun filtering technique for Entity Classes

Therefore, for the example on-line shopping cart web application use case *Checkout*, we can identify the following analysis classes:
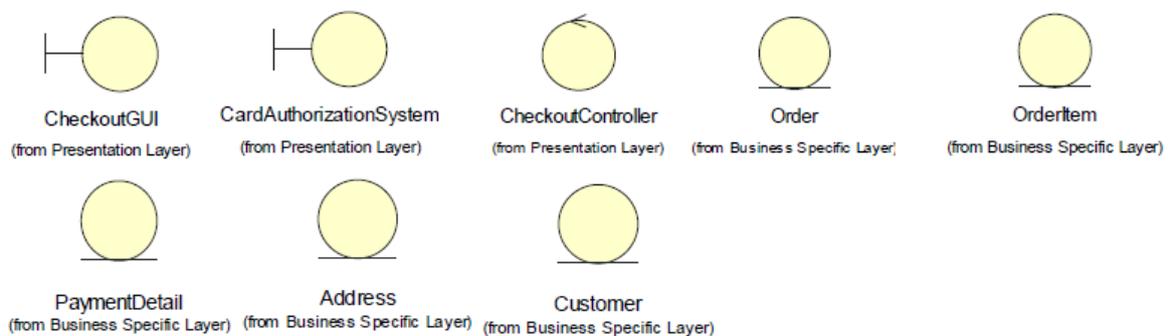

Figure 7. Checkout Use Case Realization Analysis Classes

Allocate Responsibilities to Classes

By studying the *Checkout* use case basic flow, and bear in mind the kind of responsibilities that different analysis classes should have, also applying the fundamental design patterns such as Expert, low coupling and high cohesion, the responsibilities are distributed among the objects who are taking part in the use case realization. The results are documented in both the sequence diagram and the class diagram (also called View Of Participating Classes or VOPC class diagram) as shown in Figure 8 and 9 in next section.
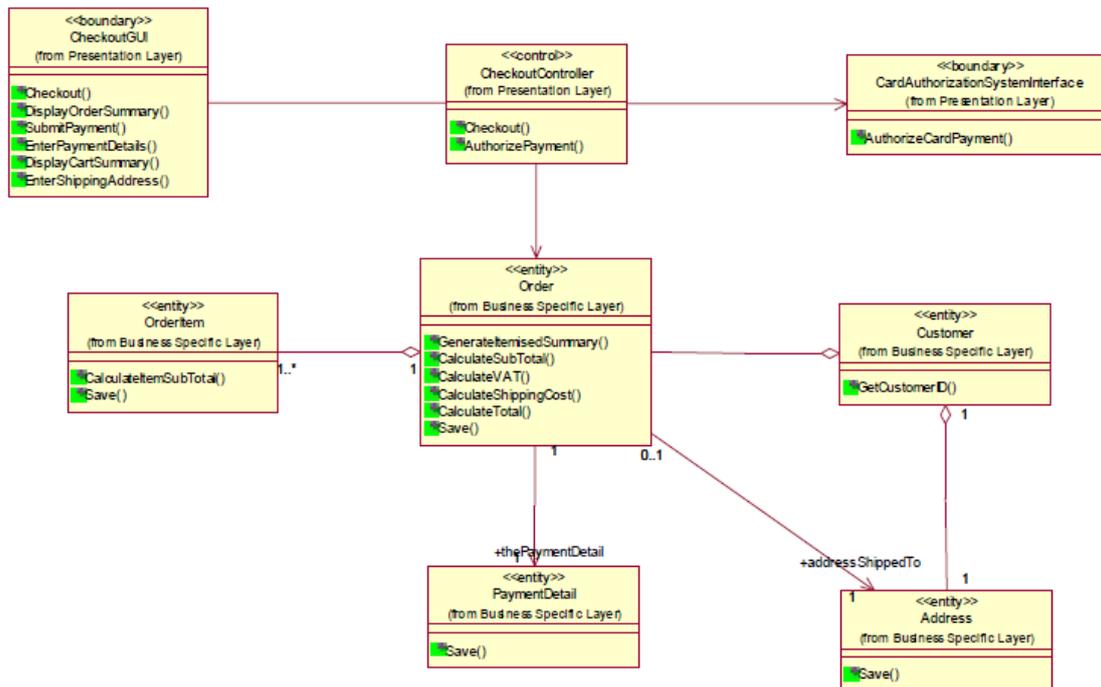
Create Interaction and Class Diagram

The use case analysis results are represented using both an interaction diagram (either sequence diagram or collaboration diagram) and a class diagram to document the dynamic (behaviour) and static (structural) aspects of the system analysis. Figure 8 and 9 are the sequence and class diagram respectively.

Figure 8. Checkout Realization Sequence Diagram

Figure 9. Checkout Realization VOPC Class Diagram

Conclusion

This article presents a systematic, robust object oriented system analysis method that is composed of three components: a customized RUP process, UML modelling language and Rational Rose CASE tool. By using an online shopping cart application as an example, starting from the use case specification, the complete system analysis process has been described and the resultant UML models produced in Rose are provided.