AUTOMATING TRANSITION FROM USE-CASES TO CLASS MODEL

Dong Liu, Kalaivani Subramaniam, Behrouz H. Far, Armin Eberlein
Department of Electrical and Computer Engineering
University of Calgary
2500 University Drive, N.W.
Calgary, Alberta, Canada, T2N 1N4
{liud, subrama, far, eberlein}@enel.ucalgary.ca

Abstract

To identify objects from the requirements and to model the problem in classes are critical in Object-Oriented Analysis and Design (OOAD). Unfortunately, this is recognized as a hard task for most software engineers, because both domain experience and expertise are needed, since there is no crisp guideline. We present an approach with a set of artifacts and methodologies, and to automate the transition from requirement to detail design. Use cases are applied as the method to capture and record requirements. All the use cases are formalized by a use case template. A glossary that contains the domain vocabulary is used throughout the OOAD process to reduce the vagueness of natural language. Some language patterns are introduced to make the automatic processing of use cases possible. We apply robustness analysis to bridge the gap between a use case and its realization, i.e. between a use case and the corresponding collaboration diagram in UML. Some rules are summarized and adopted to automate the object/class identification and behavior distribution among the classes. The implementation of the tool is described.

Keywords: *Use case; Object-oriented analysis and design; Class identification and Modeling.*

1. INTRODUCTION

Object Oriented (OO) development is currently the most popular software development methodology. Objects and classes are core concepts for Object Oriented Analysis and Design (OOAD). Finding the objects and classes, and then building the class model for a problem are among the central decisions in OO software development. However, it seems that there are no theoretically or even pragmatically well-developed guidelines to help the software engineers tackle this problem successfully. Someone once said that "It's a holy Grail. There is no panacea" [1]. Obviously, this task has puzzled the developers since OO was first introduced. The proposal of our project is to construct a methodology and develop a CASE (Computer Aided Software Engineering) tool to address this problem, and to automate the transition from requirements to detailed design in OO software development.

The structure of this paper is as follows: Section 2 introduces object elicitation and class modeling methodology including Object Model Creation Process (OMCP), use cases and robustness analysis. Section 3 talks about the details about the methodology and how to automate the transition from use cases to class model. In Section 4 the implementation of the tool is presented. Section 5 contains conclusions.

2. OBJECT ELICITATION AND CLASS MODELING METHODOLOGY

2.1 Object Model Creation Process (OMCP)

The Object Model Creation Process (OMCP) is a widely applied methodology to elicit objects and build a class model from requirements, and a few systems that implement the OMCP are already available [2]. An overview of OMCP is shown in Fig. 1. The requirements are acquired from the stakeholders through interviews or other methods, and are documented in a requirement document. Then the engineers responsible for analysis and design try to find the objects based on the problem defined by the requirements and their own understanding of the domain. Attributes, behaviors, relationships and other essentials of the objects are identified. The next step is to set up and refine the class model based on the object model for the problem using generalization. When the class model is developed, the programmer can begin implementing the classes. As shown in Fig. 1, domain knowledge, and public or private experiences with OOAD are the resources supporting the decisions. And this is why the task is hard to accomplish.

CCECE 2003 - CCGEI 2003, Montréal, May/mai 2003
0-7803-7781-8/03/$17.00 © 2003 IEEE

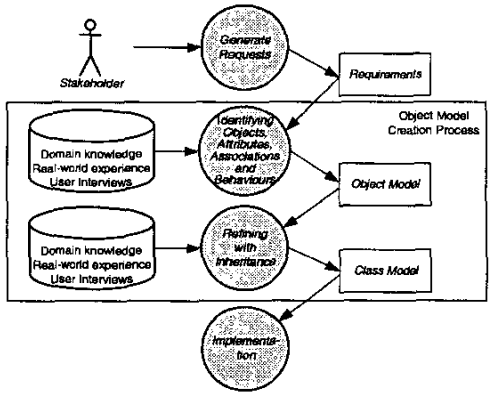

Fig. 1. **Object Model Creation Process (OMCP).**

OMCP is only a framework, the artifacts and the methods must be specified when it is implemented. Most of the input information of OMCP comes from requirements, and use cases are applied as the means to capture and record requirements in our approach.

2.2 Use Case

Use cases are means to capture the contracts between the stakeholders of a system and its behavior [3]. It was introduced and applied to the object oriented methodology very early [4]. Although there is some discredit for using use cases in OO software development process [3][5], e.g. sets of use cases form functional decomposition of the system, which is incompatible with object-based decomposition in OO technology, it is still widely used to acquire and record requirement.

UML specifies the notation for use cases. The detailed information of use cases is contained in the specification documents. These documents can be formal or informal. Some use case templates are defined and used to formalize use cases. Table 1 is one such template. In our approach, all the use cases are documented using such a template and encoded in XML.

Table 1. **Use case template.**

1. Use Case Name
1.1 Brief Description
2. Flow of Events
2.1 Basic Flow
2.2 Alternative Flows
3. Special Requirements
4. Preconditions
5. Postconditions
6. Extension Points

2.3 Robustness Analysis

After the use case model is developed, the next step of analysis is use case realization. In UML, collaboration is used to describe how the use cases are realized and interact with the society of objects. Two kinds of diagrams can be adopted: collaboration diagrams and sequence diagrams. However, the objects need to be identified before either diagram can be created. Without objects, it is hard to transit from use case to collaboration. To avoid such analysis paralysis, robustness diagram is introduced as the halfway point bridging the gap.

The idea of robustness analysis initially came from Ivar Jacobson's early paper [6]. In [4], he formally introduces it, although the name of the methodology was not specified and even the notations of stereotypes changed later. It is also known as ICONIX in Doug Rosenberg's work [7]. The behavior of the objects can be mapped to an orthogonal, three-dimensional space with interface objects, entity objects and control objects as the axes. The three stereotype objects and the interaction between them comprise the robustness diagram. With robustness diagram included, the process from use cases to class diagrams can be illustrated as shown in Fig. 2. In the stage of robustness diagram, stereotype objects are identified and attributes of objects are found as well. In the stage of collaboration, interactions among objects are identified. Then the behavior is allocated to different objects.

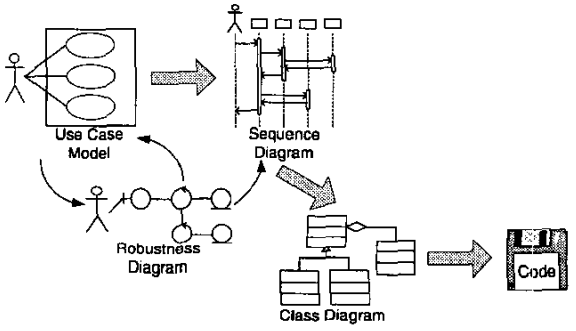

Fig. 2. **Transition from use case to class model.**

3. AUTOMATING THE TRANSITION

To develop a tool that helps automate the transition from use cases to class diagrams, some good practices and methodologies can be applied in the process [8]. Furthermore, there are some rules summarized for the creation of artifacts and transition between artifacts. The biggest problem for automation is the vagueness of natural language. Another problem is that use cases focus on the system's behaviors, while we want to create a model of objects and classes. The methods and mechanisms used in our approach are introduced in the following sub-sections.

3.1 Glossary

When requirements are documented, and the artifacts created during development are named and specified, it is common that different engineers use different terms, and the terms have diverse meanings depending on the domain and personal background. This makes communication inside the development team difficult, and leads to deviations or even mistakes. In the community of eXtreme Programming (XP), System Metaphor is one of the twelve practices of XP [9]. Now they have revised it to Common Vocabulary [10]. A shared vocabulary makes requirements unambiguous so that they can be easier understood by the engineers for system analysis and design. We also introduce a glossary for OO analysis to remove the vagueness of the use cases in natural language. Each term in the glossary represents a static concept model for an object in the real world.

3.2 Use Case Language Patterns

If all the use cases were written freely without any constraints, the abundance of syntactic phenomena will give rise to much semantic vagueness, which causes great difficulties when automating the processing of use cases. We create some language patterns that partly formalize the use case language. Table 2 lists the 3 basic language patterns.

Table 2. **Basic language patterns for use case.**

Simple Statement	Most of the simple English sentences are of this pattern. The structure of this basic pattern is Subject + Predicate.
While-Do Statement	This pattern is used to represent the repeated event flows when a condition is fulfilled. Both While-part and Do-part are of Simple Statement. or set of Simple Statements
If-Then Statement	This pattern is used to represent the specific event flow under a certain condition. If-part is of Simple Statement and Then-part is of Simple Statement or set of Simple Statements.

For most of the statements in use cases, namely, Simple Statements, there are more sub patterns, which are listed in Table 3.

Table 3. **Sub patterns for Simple Statement.**

I	Subject + Verb
II	Subject + (Verb + *Object*)
III	Subject + (Be + Predicative)
IV	Subject + (Have + Past Participle + *Object*)
V	Subject + (Verb + Direct Object + to/for + Indirect Object)
	Subject + (Verb + Indirect Object + Direct Object)
VI	Subject + (Verb1 + Object + to + Verb2)
VII	Subject + (Verb1 + Object1 + to + Verb2 + Object2)
VIII	Subject + (Verb + Object + Present Participle)
	Subject + (Verb + Object + Adjective)
IX	Subject + (Verb + Object + Past Participle)

Italic font denotes the part is optional.

When the requirements are documented in use cases and formalized, we encourage such practices as: Write the use case specification using of the 3 statement patterns, namely simple statement, while-do statement and if-then statement; Speak in active voice rather than passive voice; Set up a glossary for the domain; use the same term from the beginning to the end; Replace all the pronouns with specific names of the entities; Use the same verb for the same service or action in different statements; Keep the form of complex predicate unique.

Using these language patterns and practices, we can greatly reduce the vagueness of use cases written in natural language, with only a few constraints on the use case writing. The behavior information inside the statements can be fetched by machine with the syntactic pattern matched. And the patterns are important factors of preconditions for the rules in if-then form. The following segment is an example of use case written using these patterns, which is the Basic Flow part of a use case named Withdraw Cash for an ATM system.

1. *The system starts withdrawal transaction;*
2. *The customer selects account from the customer console; the system gets the account;*
3. *The customer selects the amount from the customer console; the system gets the amount;*
4. *The system generates the transaction information; the bank gets the transaction information;*
5. *The bank approves the withdrawal transaction; the system dispenses the cash to cash dispenser;*
6. *The withdrawal transaction ends.*

3.3 Process Pattern

For every use case with detailed specification, there is a corresponding use case realization. For every use case realization, two artifacts are generated, the robustness diagram and the collaboration/sequence diagram. The robustness diagram finds analysis classes from the system's behavior, and the collaboration/sequence diagram distribute the behavior to the analysis classes found. Fig. 3 and Fig. 4 are respectively the robustness diagram and collaboration diagram for the realization of the use case example in section 3.2.

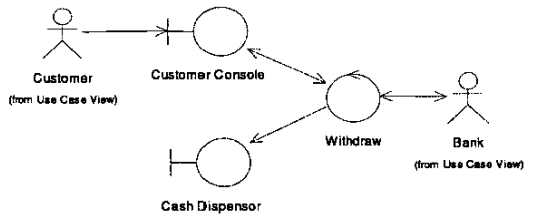

Fig. 3. **Robustness diagram for withdraw cash.**

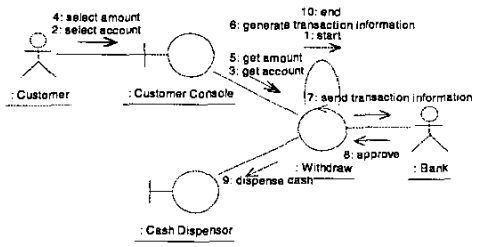

Fig. 4. **Collaboration diagram for withdraw cash.**

Because use cases are clustered by system behavior, the generated robustness diagrams and collaboration /sequence diagrams are also clustered by behavior. When we cluster the artifacts based on analysis class, we get the analysis class model of the system. For every analysis class, identify responsibilities and associations, establish association between the classes, and add the attributes into the classes. Finally, the class diagram is generated. Fig. 5 shows the class `Withdraw` generated with the information of this use case and the artifacts. After all the use cases are analyzed, we can find that there is a generalization between `Transaction` and `Withdraw`, i.e. `Withdraw` is a subclass of `Transaction`.

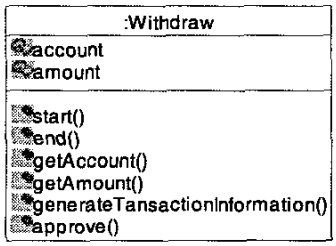

Fig. 5. **Withdraw class.**

4. IMPLEMENTATION

To cooperate with Rational Rose [11], the CASE tool is designed and developed as an add-in of Rose. The functionality of the tool is divided into 2 parts: one is use case realization, and the other class diagram generation. This arrangement follows the process pattern, and at the same time offers convenient service to the user. The user can start from use cases or the half-way artifacts: robustness diagram and collaboration/sequence diagram. The user can modify the robustness diagrams or collaboration/sequence diagrams generated from the use cases specifications, then use the function of class diagram generation to generate or update the class diagram.

The use case specifications are processed and encoded in XML (eXtensible Makeup Language) according to the XML schema. The schema is constructed based on the use case template and language patterns. The path and file name of the XML file is stored in the corresponding use case's external document information slot. The parsing and processing of the XML file are implemented using VC++ with the support of Xerces project [12]. The glossary can be loaded to the tool according to different domains. It is encoded in XML as well. The information retrieval of Rose elements and creation and visualization of UML artifacts are performed using the Rational Rose Extensibility Interface (REI). The tool can be installed on Microsoft Windows 2000 and later systems with Rational Rose installed.

5. CONCLUSION

The methodology to automate the transition from requirements to detail design is discussed in this paper. Use case, robustness analysis, the patterns and practices to enable the automatic transition are introduced in detail. The paper also reports the implementation of the CASE tool. How to capture the requirements and write them into effective use cases and how to make use of domain knowledge by a CASE tool are still problems to tackle.

References

[1] G. Booch, *Object Oriented Design with Applications*. The Benjamin Cummings Publishing Company, 1991.

[2] R.S. Wahono, B.H. Far, "*A framework of object identification and refinement process in object-oriented analysis and design*," Proceedings of The 1st IEEE International Conference on Cognitive Informatics, ICCI2002, Calgary, Canada, 2002.

[3] A. Cockburn, *Writing Effective Use Cases*. Addison-Wesley, 2000.

[4] I. Jacobson, M. Christerson, P. Jonsson and G. Övergaard, *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, Wokingham (England), 1992.

[5] B. Meyer, *Object-Oriented Software Construction*. Prentice Hall, 2002.

[6] Ivar Jacobson, "*Object oriented development in an industrial environment*," OOPSLA '87, pp183-191, 1987.

[7] D. Rosenberg, K. Scott, *Applying Use Case Driven Object Modeling with UML: An Annotated e-Commerce Example*. Addison-Wesley Professional, 2001.

[8] Liwu Li, "*A semi-automatic approach to translating use cases to sequence diagrams*," Proceedings of Technology of Object-Oriented Languages and Systems, 1999.

[9] Roy W. Miller, Christopher T. Collins, "*XP distilled*," IBM developerWorks, March 2001. Available: http://www-106.ibm.com/developerworks/java/library/j-xp/#author1

[10] Roy W. Miller, "*Demystifying extreme programming: "XP distilled" revisited*," IBM developerWorks, August 2002. Available: http://www-106.ibm.com/developerworks/java/library/j-xp0813/

[11] IBM, Rational Rose, http://www.rational.com/products/rose/index.jsp

[12] The Apache XML Project, http://xml.apache.org/index.html