

Natural Language Requirements Analysis and Class Model Generation Using UCDA

Dong Liu¹, Kalaivani Subramaniam¹, Armin Eberlein², and Behrouz H. Far¹

¹ Department of Electrical and Computer Engineering, University of Calgary, 2500, University Drive, N.W., Calgary, Alberta, Canada, T2N 1N4
{liud, subrama, far}@enel.ucalgary.ca

² Computer Engineering Department, American University of Sharjah, UAE
eberlein@enel.ucalgary.ca

Abstract. This paper presents a methodology to automate natural language requirements analysis and class model generation based on the Rational Unified Process (RUP). Use-case language schemas are proposed to reduce complexity and vagueness of natural language. Some rules are identified and used to automate class model generation from use-case specifications. A CASE tool named Use-Case driven Development Assistant (UCDA) is implemented to support the methodology. UCDA can assist the developer to generate use-case diagrams, use-case specifications, robustness diagrams, collaboration diagrams and class diagrams in IBM Rational Rose. It helps accelerate requirements analysis and class modeling, and reduce the time to market in software development.

1 Introduction

Object-Oriented Analysis and Design (OOAD) has become a very popular software development approach since the 1990's. Object elicitation and class modeling are among the central activities in OOAD. The objects are identified from the requirements, and the class model is generated based on them as well. Generally, there are two ways to specify the requirements: using formal languages or using natural languages (NL). The research community has focused on methods based on formal language requirements [1-3], while NL is widely used for requirements documentation in industry. It is hard to automate NL requirements analysis, because NL is inherently complex, vague and ambiguous [4].

Most commercial CASE (Computer Aided Software Engineering) tools do not supply the functionality of NL requirements analysis. However, there are several such tools that have been developed for research. CoGenTex Inc. developed a prototype tool named LIDA (Linguistic assistant for Domain Analysis), which provides linguistic assistance in model development [5]. The tool can process textual documents and help the user to generate a class model visualized in UML (Unified Modeling Language). NIBA (Natural Language Requirements Analysis in German) is an interdisciplinary project between computer scientists and computer linguists at the University of Klagenfurt, Austria [6]. The tool can parse requirements documents in German, interpret and transform output of the parser to conceptual pre-design

schemas, validate the schemas and finally generate a conceptual model in UML. These approaches only generate the conceptual model, but the behavior of classes still need to be identified separately.

So far, it is impossible for machines to automatically perform the whole OOAD process, but it is possible to automate some micro-activities in it. We developed a method for use-case model generation, object identification and class modeling with respect to natural language requirements based on the Rational Unified Process (RUP). The methodology is introduced in Sect. 2. Sect. 3 presents the CASE tool that was developed based on the methodology. Sect. 4 concludes the paper.

2 Methodology

Based on the Rational Unified Process (RUP), the activities and corresponding artefacts during requirements, analysis and design are specified as follows:

- Identify actors and use cases from stakeholder requests.
- Structure the use cases into use-case diagrams.
- Generate the use-case specifications.
- Review the use-case specifications.
- Analyze the use-case specifications and generate the analysis model.
- Review the analysis model.
- Generate the design model based on the analysis model.

The whole process is divided into two parts based on different concerns. The first part addresses NL requirements analysis and use-case modeling. The second part is concerned with the use-case realization and class model generation. The artifacts and activities in the process are shown in Fig. 1. The output of the requirements phase is a use-case model. Use cases are means to capture the contracts between the stakeholders of a system and its behavior [1]. A use-case model comprises diagrams in UML and specifications that record sequences of actions that a system can perform by interacting with outside actors.

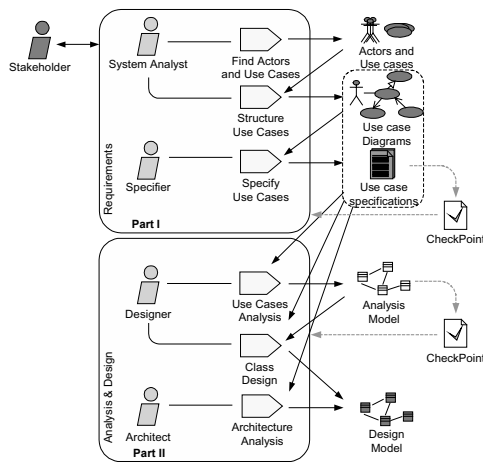


Fig. 1. Developers, activities, and artifacts in OOAD

2.1 NL Processing and Use-Case Modeling

Stakeholders’ requests are parsed by a natural language parser to identify use cases. The detailed description of each use case is parsed and analyzed to generate a use-case specification document.

2.1.1 Natural Language Parsing

Stakeholders’ requests documented in natural language are analyzed and processed by a parser. A shift-reduce parser is applied in UCDA. The typical structural elements of sentences that are recognized by the parser are listed in Table 1. The presence of coordinated structures and modifiers will increase the complexity of sentences. The tool can identify some complex sentences and break them down into simpler sentences. The rules for sentence reconstruction are listed in Table 2. The parser converts input requests into sentences, and each sentence into an abstract representation of its syntactic structure. For example, “*students request a course catalog*” can be tagged as [*students’/’N’, ‘request’/’V’, ‘a’/’ART’, ‘course’/’N’, ‘catalog’/’N’*].

Table 1. Sample tag set

Tag	Description	Tag	Description
N	Noun	P	Preposition
V	Verb	DET	Determiner
ART	Article	ADV	Adverb
ADJ	Adjective	CONJ	Conjunction
Q	Quantifier	auxV	Auxiliary Verb

Table 2. Rules for sentence reconstruction

Complex Structure	Simplified Structure
NP1 + Verb1 + NP2 + Verb2 + NP3	(i) NP1 + Verb1 + NP2 (ii) NP2 + Verb2 + NP3
{Q, ADJ, V-ing, DET} + N	N
auxV + V	V
Sentence1 {AND/OR} Sentence2	(i) Sentence1 (ii) Sentence2
NP + VP + NP1 and NP2	(i) NP + VP + NP1 (ii) NP + VP + NP2

NP – Noun Phrase; VP – Verb Phrase

2.1.2 Use Case Identification

The candidate actors are derived from nouns, especially those that are subjects of the statements, and the candidate use cases are derived from the verb phrases acting as actors’ predicates. If a candidate actor is not found in the glossary, it will be removed from the candidate actor set. We also apply heuristics to help the developer to distill the candidate sets and identify actors and use cases. Typical heuristics to distill actors are:

1. Who will supply, use, or remove information?
2. Who will use the functionality?
3. Who will support or maintain the system?
4. What are the system’s external resources?
5. What are the other systems that are needed?

The heuristics to distill use cases are as follows:

1. For each actor, what are the tasks?
2. Does the actor have to be informed about certain occurrences in the system?

When use cases are identified from the requests, detailed information is needed for each use case to generate the use-case specification. The following is part of the requirements for an ATM (automated teller machine) system.

The ATM will service one customer at a time. A customer will start a session when s/he inserts an ATM card. The customer will then be able to perform one or more transactions.

The actor identified from this paragraph is “customer”, and use cases identified are “start session” and “perform transactions”.

2.1.3 Use-Case Specification

A template is used to standardize the use-case specifications. The template contains such entries as use-case name, flow of events, special requirements, preconditions, postconditions and extension points. To enable the automated realization of use-case specifications in NL, we introduce a set of use-case schemas to normalize the NL statements. Table 3 lists the use-case schemas. The structures of simple statements are identified during parsing. They are transitive (a.k.a. monotransitive), intransitive, ditransitive, intensive, complex transitive, prepositional and non-finite [7].

Table 3. Use-case language schemas

Basic	This schema applies to all the events. The sentences in the form of this schema are simple statements.
If-then	This schema is used for alternative flows. An if-clause may consist of one or more condition statements, each of which can be described using the basic schema. A then-clause contains a flow of events.
Do-until	This schema describes a repeated event or sequence of events under a certain condition. A do-clause may contain an event or a sequence of events. An until-clause may consist of one or more condition statements, on which the iteration will stop and the flow goes to next step.
Con-Noc	This schema describes the performance of two or more activities during the same time interval, i.e., concurrency. This schema starts with a Con and ends with a Noc.

2.2 Class Model Generation

2.2.1 The Use-Case Processing Method

To perform use-case driven analysis and design, we propose a use-case processing method as follows:

For each use case,

- a. elicit the analysis classes, identify their stereotypes, and generate a robustness diagram;

- b. decompose the system’s behavior, distribute the behavior to analysis classes, and generate a collaboration diagram.

For the analysis classes,

- a. describe responsibilities;
- b. describe associations and establish them in the class diagram;
- c. identify the generalization relationships and establish them in the class diagram.

2.2.2 Rules for Class Model Generation

Domain knowledge is very important in object identification and class model generation. The glossary that defines specific terms of the domain represents part of the domain knowledge. If an entity in a use-case specification is found in the glossary, it is a candidate object that may correspond to a class in the class model.

The behavior types of the system, the associations between the stereotype objects, and the structures of the action statements in use-case specifications are associated with each other. Fig. 2 shows a model of the actions in actor-system interaction [1]. Four types of behavior are included in the model. The relationships between the behavior types and the associations between stereotype objects are listed in Table 4. The relationships between the statement structures and the behavior types are summarized, and applied to automate use-case realization. Boundary classes are used to model the system interface that handles the communication between the environment and the system. Entity classes are used to model the real world entities. Control classes are responsible for the flow of events in the use case, and they are application-dependent. Determining the control classes for a problem is very subjective.

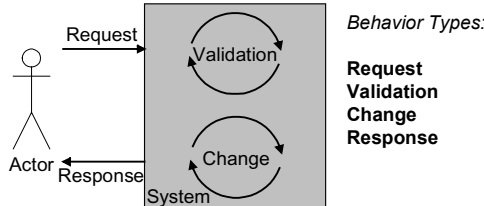


Fig. 2. An actor-system interaction model and 4 behavior types

Table 4. Relationships between behavior types and associations between stereotype objects

<i>Behavior Type</i>	<i>Association</i>
<i>Request</i>	Actor — Boundary Object
<i>Validation</i>	Boundary Object — Control Object and Control Object — Entity Object
<i>Change</i>	Control Object — Entity Object
<i>Response</i>	Entity Object — Boundary Object

Actor: actor, Boundary Object: boundary object, Control Object: control object, Entity Object: entity object

We identified the relationships between all statement structures and the behavior types, and represented them in 17 rules for object and message identification [8]. Because of the length of this paper, we only demonstrate a rule for transitive structure shown in Fig. 3, where NP represents noun phrase; VPss represents verb phrase with

the statement structure; PP represents prepositional phrase; Vgp represents verb group; and Prep represents preposition [8].

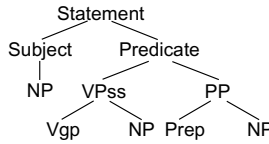


Fig. 3. The structure of a transitive statement

The rule for object identification is:

Rule: *If the structure of a statement is transitive (shown in Fig. 3), and Subject/NP//Noun(head) is an actor, then this statement is corresponding to the Request behavior type and Predicate/PP/NP//Noun(head) is a boundary object if it exists in the glossary, and Predicate/VPss/NP//Noun(head) is an entity object if it exists in the glossary.*

If there are two objects or an actor and an object in one statement, an association between them is identified. To generate the collaboration diagram, the messages contained in one scenario are identified. The corresponding rule for message identification is:

Rule: *If Subject/NP//Noun(head) is an actor, and Predicate/PP/NP//Noun(head) is a boundary object, then the action is Predication/VPss/Vgp/Verb(head) + Predication/VPss/Vgp/NP//Noun(head), the sender is Subject/NP//Noun(head) and the receiver is Predicate/PP/NP//Noun(head).*

The responsibilities of the classes can be identified from the messages in the collaboration diagrams. Each message consists of a sender, a receiver and an action. The receiver has the responsibility for the execution of the action. The messages in collaboration diagrams are transformed to the classes' responsibilities in this way.

Composition and generalization are two kinds of class relationships to be identified in the class model of the system under development. Some aggregation relationships can be derived from the use-case inclusion relationships.

Rule: *If one use case includes another use case, then a composition relationship is likely to exist between the core control classes identified from the use cases.*

Class generalizations can also be identified from use-case generalization relationships.

Rule: *If one use case has a generalization relationship with another use case, then a generalization relationship is likely to also exist between the core control classes identified from the use cases.*

2.2.3 Rules for Analysis Model Validation

We propose a method to validate the analysis model, especially the robustness diagrams. There are some constraints for objects and associations in a robustness diagram according to its semantics. The rules listed in Table 5 are derived from the constraints and used for robustness diagram validation.

Table 5. Rules for robustness diagram validation

<i>Case</i>	<i>Validation</i>	<i>Suggestion</i>
	Not allowed.	
	Allowed	
	Not allowed.	
	Not allowed.	
	Not allowed.	
	Allowed.	
	Not allowed.	
	Allowed.	
	Allowed.	
	Not allowed.	

: actor, : boundary object, : control object, : entity object

3 UCDA: Use-Case Driven Development Assistant

3.1 Overview

To implement the methodology, we develop a CASE (Computer Aided Software Engineering) tool named UCDA (Use-Case driven Development Assistant). Just like the methodology, UCDA is composed of two parts: one part for NL requirements processing and use-case modeling, and the other for use-case realization and class model generation. The architecture of UCDA is shown in Fig. 4. UCDA is integrated seamlessly with IBM Rational Rose. The user can manage UCDA with Rational Rose’s Add-in manager. Most artefacts generated by UCDA are represented in XML and visualized in Rose.

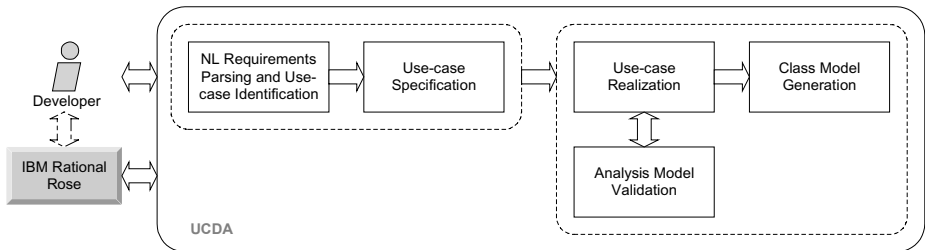


Fig. 4. UCDA architecture

The features of UCDA currently implemented are as follows:

1. Parse the NL requirements and identify actors and use cases, and then generate the use-case diagram in Rational Rose.
2. Assist the user to finish use-case specification.
3. Realize the use cases, identify the classes, and generate robustness diagrams and collaboration diagrams in Rational Rose.
4. Validate the analysis class model via robustness diagrams.
5. Generate the class model in Rational Rose.

Not all the activities can be fully automated especially the use-case specification. The user needs to interact with UCDA to supply the necessary information, and the tool will help the user to develop a model in UML for further revision.

3.2 Use-Case Modeling Environment

When the user has only the requests, s/he can start to analyze with UCDA. Fig. 5 is the environment for requirements parsing. The user needs to paste or edit the requests of a project in it. Then UCDA can help the user identify the use cases from the request and generate the use-case diagram in Rational Rose.

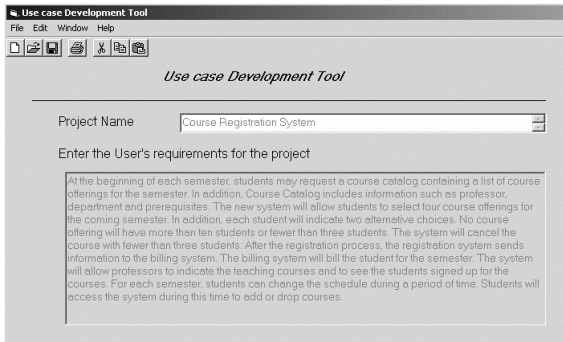


Fig. 5. The environment for NL requirements parsing and use-case identification

Then the user can specify the use cases with the assistance of UCDA. UCDA parses the user's input information and normalizes it based on use-case language schemas. The structure of each statement in the flow of events is identified, and all statements are encoded in XML. An example use-case specification with XML markups removed is as follows. Note that person and number effects on verbs are removed.

Actors: customer, bank

Flow of Events:

Basic Flow:

1. the system start withdrawal transaction;
2. the customer select the account on the customer console;
3. the system get the account from the customer console;
4. the customer select the amount on the customer console;
5. the system get the amount from the customer console;
6. the system generate the withdrawal transaction information;
7. the system send the withdrawal transaction information to the network connection;
8. the bank get the withdrawal transaction information from the network connection;

9. the bank send the withdrawal transaction approval to the network connection;
 10. the system get the withdrawal transaction approval from the network connection;
 11. the system dispense the cash in the cash dispenser;
 12. the customer get the cash from the cash dispenser;
 13. the system record the withdrawal transaction information into the log;
 14. the withdrawal transaction end;
- Alternative Flow:

- If the bank do not approve the withdrawal transaction,
- then
- i. the system display an error message on the customer console;
 - ii. the system record the withdrawal transaction information into the log;
 - iii. the withdrawal transaction end;

3.3 Use-Case Realization Environment

When the use-case model is ready, the user can use UCDA to realize the use cases and generate the class model. The functions of UCDA can be accessed via Rose’s menu. All the diagrams generated by the tool are visualized in Rational Rose. The environment for use-case realization is shown in Fig. 6. The user can set the glossary and select a use case to realize. When collaboration diagrams are generated, the tool can distribute the behavior and generate the class model in Rational Rose. The robustness diagram generated by UCDA based on the example specification in Sect. 3.2 is shown in Fig. 7 and the corresponding collaboration diagram is shown in Fig. 8.

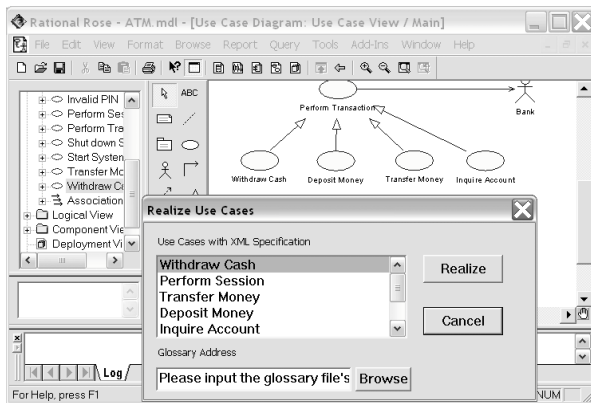


Fig. 6. The environment for use-case realization cooperating with Rational Rose

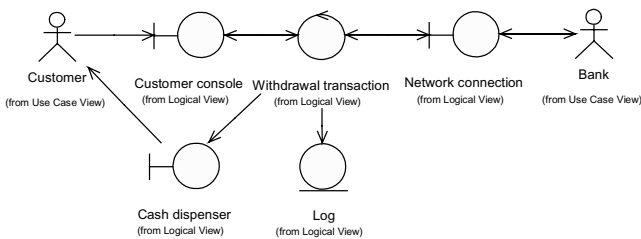


Fig. 7. A robustness diagram generated by UCDA

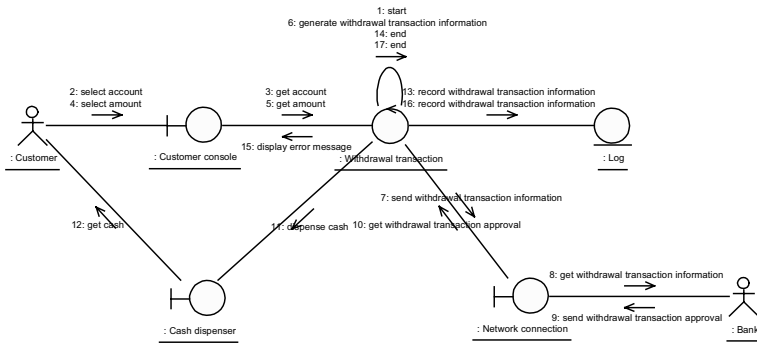


Fig. 8. A collaboration diagram generated by UCDA

4 Conclusion

A methodology for natural language requirements analysis, use-case modeling and use-case driven analysis and design is presented. The methodology comprises good practices from both natural language requirements analysis and the use-case driven analysis and design. Use-case language schemas are proposed to normalize use-case specifications, and the methods to automate object identification and class model generation based on statement structures are discussed. A CASE tool was developed to support the methodology. A future research topic is to implement features of software architecture analysis and integrate it with the UCDA and Rational Rose.

References

1. Cockburn, A.: Writing effective use cases. Addison-Wesley (2000)
2. Bois, P. D., Dubois, E., Zeippen, J.M.: On the use of a formal RE language-the generalized railroad crossing problem. Proceedings of the Third IEEE International Symposium on Requirements Engineering, Annapolis MD (1997) 128-137
3. Li, X., Liu, Z., He, J.: Formal and use-case driven requirement analysis in UML. 25th Annual International Computer Software and Applications Conference, COMPSAC2001 Chicago (2001) 215-224
4. Boyd, N.: Using natural language in software development. Journal of Object-Oriented Programming 11(9) (1999) 45-55
5. Overmyer, Scott P., Lavoie, B., Rambow, O.: Conceptual modeling through linguistic analysis using LIDA. Proceedings of the 23rd International Conference on Software Engineering, ICSE 2001, Toronto (2001) 401-410
6. Niba, L.C.: The NIBA workflow: From textual requirements specifications to UML-schemata. Proceedings of International Conference on Software & Systems Engineering and their Applications, ICSSEA 2002, Paris (2002)
7. Roberts, P.: Patterns of English. Harcourt, Brace and Company (1956)
8. Liu, D.: Automating transition from use cases to class model. MSc Thesis, University of Calgary, Calgary (2003)