

## **Theme 10: Methodologies for automating transition from stakeholders' requests to skeleton code**

### **Abstract**

*The goal of this research is to develop 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 the complexity and vagueness of natural language. Some rules are identified and used to automate the generation of the class model from use-case specifications. A CASE tool named 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 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 Development (OOAD) has become a very popular software development approach since 1990's. Object elicitation and class modeling are among the central activities in OOAD. The objects and the class model are identified and generated from the requirements. Generally, there are two ways of specifying requirements: the requirements specified in formal languages and those in natural languages (NL). The research community has focused on the methods based on formal language requirements [1-3], while NL is widely used for requirements documentation in industry. However, it is hard to automate NL requirements analysis, because NL is often complex, vague and ambiguous [4].

There are some CASE (Computer Aided Software Engineering) tools that have been developed to support OOAD based on natural language requirements. CoGenTex Inc. ([www.cogentex.com](http://www.cogentex.com)) developed a methodology and the corresponding prototype tool named LIDA (Linguistic assistant for Domain Analysis), which provides linguistic assistance in the model development process [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 [6]. The tool can parse the requirements documents in German, interpret and transform the output of the parser, validate the result and finally generate the concept model in UML. However, most of the approaches just generate the concept class model, and the behavior of the classes still need to be identified.

So far, it has been impossible for the machine to perform the whole process successfully, 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 Rational Unified Process (RUP). The method and the tool are developed to automate as many the micro-activities during requirements analysis and class model generation as possible.

## 2. Methodology

Based on RUP, the activities and corresponding artefacts in requirements, analysis and design phases are refined 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 about use-case realization and class model generation. The artifacts and activities in the process are shown in Fig. 1.

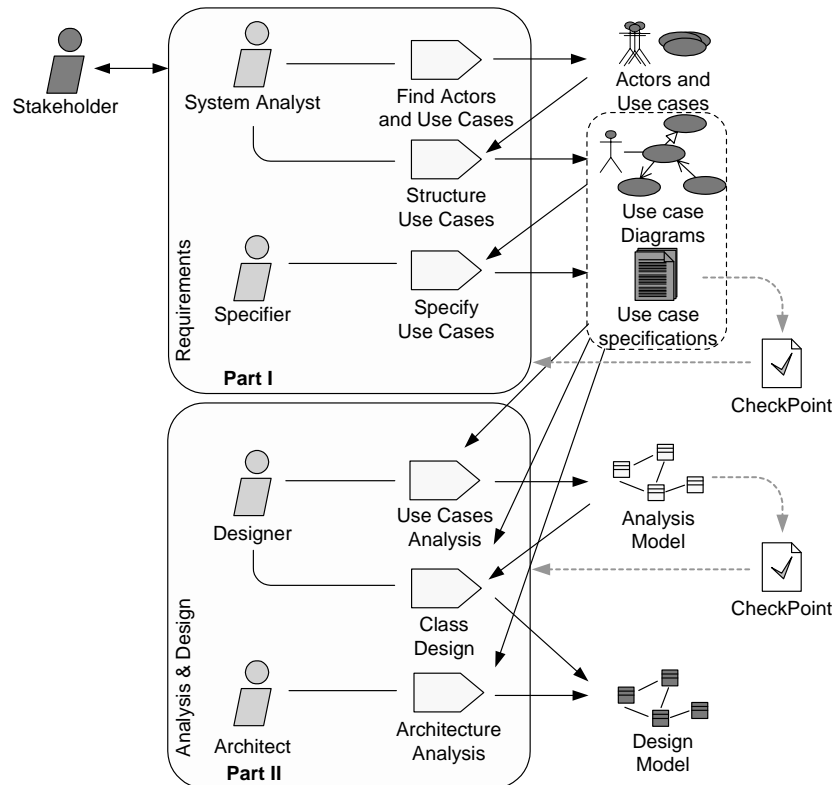


Fig. 1. The structure of the project

## 2.1 NL Processing and Use-case Modeling

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

### 2.1.1 Natural language parsing

Stakeholders' requests are documented in natural language and are analyzed and processed by a parser. A shift-reduce parser (<http://nltk.sourceforge.net/>) is applied in UCDA. 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'*]. The complete tag set is shown in Table 1.

Table 1. Tag set

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

The complex sentences are reconstructed according to the rules listed in Table 2. The use cases are identified from the reconstructed sentences.

Table 2. Rules for sentence reconstruction

Sentence structure	Structure after reconstruction
Sentence has no verb	Discard the sentence
NP1 + Verb1 + NP2 + Verb2 + NP3	NP1 + Verb1 + NP2 NP2 + Verb2 + NP3
NP1 + be + past tense of verb	Prompt the user to change from passive voice to active voice
NP preceded by Q or ADJ or Verb-ing or DET	Discard Q, ADJ, Verb-ing, DET
VP preceded by auxiliary verb	Discard auxiliary verb
`No` + NP + VP	NP + `not` + VP
Sentence1 AND/OR Sentence2	Sentence1 Sentence2
NP VP NP1, NP2, NP3 and NP4	NP VP NP1 NP VP NP2 NP VP NP3 NP VP NP4

NP – Noun Phrase; Q – Quantifier; ADJ – Adjective; DET – Determiner; VP – Verb Phrase

### 2.1.2 Use Case Identification

The candidate actors are derived from the sets of nouns, especially those that are the subjects of the statements, and the candidate use cases are derived from the verb phrases acting as the 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 set 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 the use cases are identified from the requests, detailed information is needed for each use case to generate the use-case specification.

### 2.1.3 Use-case Specification

The use-case specifications are generated based on a template. The template contains the entries like 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 them. Table 3 lists the use-case schemas. The structures of simple statements are identified to make the latter processing easy. The structure types are transitive (a.k.a. mono-transitive), intransitive, ditransitive, intensive, complex transitive, prepositional and non-finite [7].

**Table 3. Use-case schemas**

<b>Basic Statement</b>	This schema applies to all the events. It includes all the simple statement structures.
<b>If-then</b>	This schema is used for the alternative flow. An if-clause may consist of one or more condition statements, each of which can be described using the basic statement style. A then-clause contains a flow of events.
<b>Do-until</b>	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.
<b>Con-Noc</b>	This schema describes the performance of two or more activities during the same time interval, i.e., the 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,

1. elicit the analysis classes, identify their stereotypes, and generate robustness diagram;
2. decompose the system's behavior, distribute the behavior to analysis classes, and generate the collaboration diagram.

For the analysis classes,

1. describe the responsibilities;
2. describe the associations and establish them in the class diagram;
3. identify the generalization relationships and establish them in the class diagram.

### 2.2.2 Rules for Class Model Generation

For a certain system under design, the 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. A model for actor-system interaction, shown in Fig. 2., has been developed to show the actions in an interaction [1]. Four types of behaviors are included in the model. The relationships between the behavior types and the associations between stereotype objects are listed in Table 3. The relationships between the statement structures and the behavior types are summarized to support the automated processing of use-case realization.

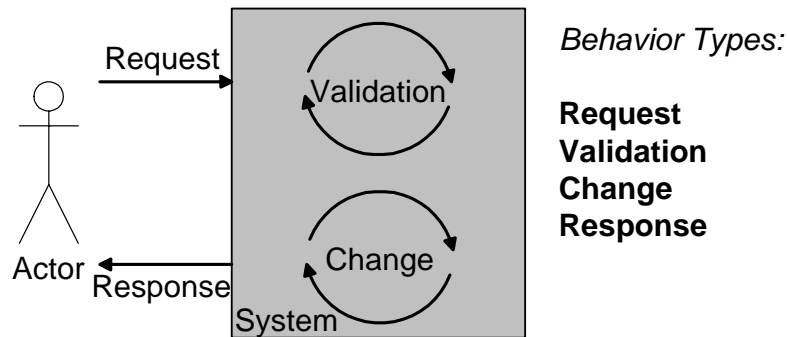


Fig. 2. An actor-system interaction model and 4 behaviour types

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

<i>Behavior Type</i>	<i>Association</i>
<i>Request</i>	
<i>Validation</i>	
<i>Change</i>	
<i>Response</i>	

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

We identify the relationships between all the statement structures and the behavior types, and represent them in 17 rules for object and message identification. Here 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.

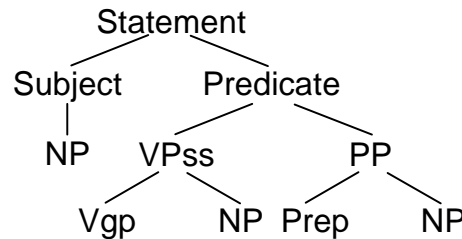


Fig. 3. The structure of a transitive statement

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 is found in the glossary, and Predicate/VPss/NP/Noun(head) is an entity object if it is found in the glossary.

If there are two objects or an actor and an object existing in one statement, an association between them is identified. To generate the collaboration diagram, the messages existing 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.*


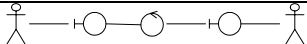


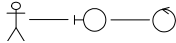

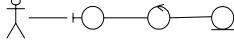

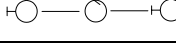


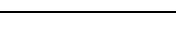




Class generalizations can also be identified from use-case generalization relationships.





**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. These constraints are based on the semantics of the robustness diagram. The rules listed in Table 4 are derived from the constraints and used for robustness diagram validation.

**Table 5. Rules for robustness diagram validation**

Case	Validation	Suggestion
	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 Developer Assistant

#### 3.1 Overview

To implement the methodology, we develop a CASE (Computer Aided Software Engineering) tool named UCDA (Use-Case driven Developer Assistant). Just like the methodology, UCDA is mainly composed of two parts: the 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 showed in Fig. 4. UCDA is integrated with Rational Rose seamlessly. The user can manage UCDA with Rational Rose's Add-in manager shown in Fig. 5. Most of the artefacts that are generated by UCDA and represented in XML are also created in Rational Rose. This enable the developer can easily access the model in UML and update is in Rose.

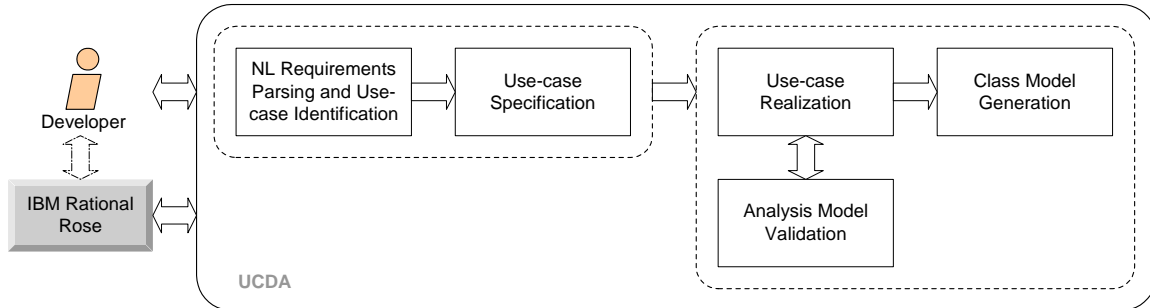


Fig. 4. UCDA architecture

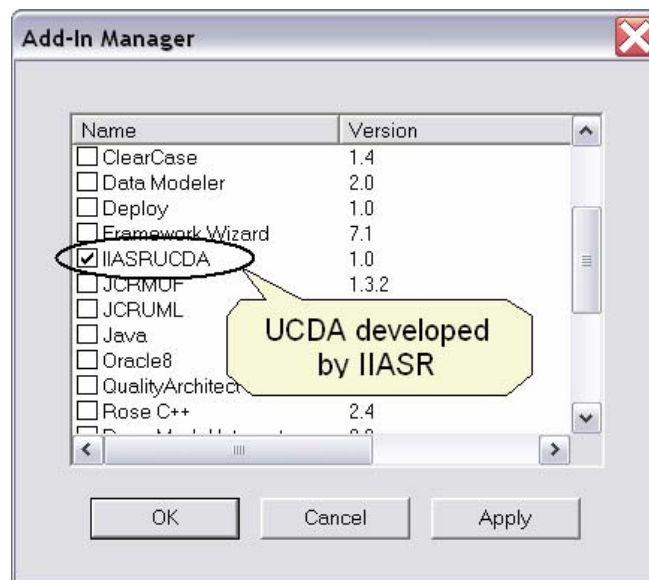


Fig. 5. Managing UCDA in Rational Rose

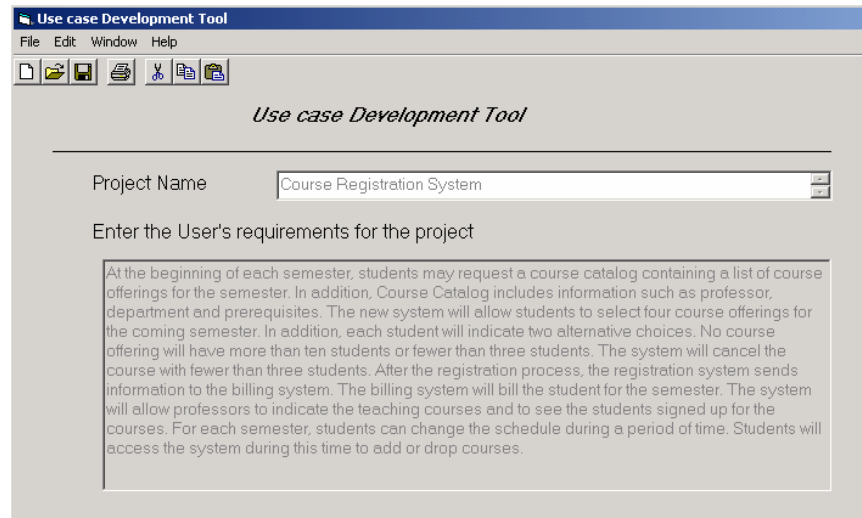
The implemented features of UCDA are as follows:

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

Not all the activities can be automated especially the use-case specification. The user needs to interactive with UCDA to supply the proper information, and the tool will help the user to finish the work easily and get a model in UML for refinement.

### 3.2 Use-case Modeling Environment

When the user has just very brief requirements, say the requests, she/he can start to work with UCDA to finish the analysis and design work. Fig. 6 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. Then the user can specify the use cases with the assistance of UCDA. The environment of use-case specification is shown in Fig. 7. UCDA parse the user's input information and normalize it by use-case language schemas. The structure of each statement in the flow of event is identified, and the statement is encoded in XML.



**Fig. 6. The environment for NL requirements parsing and use case identification**

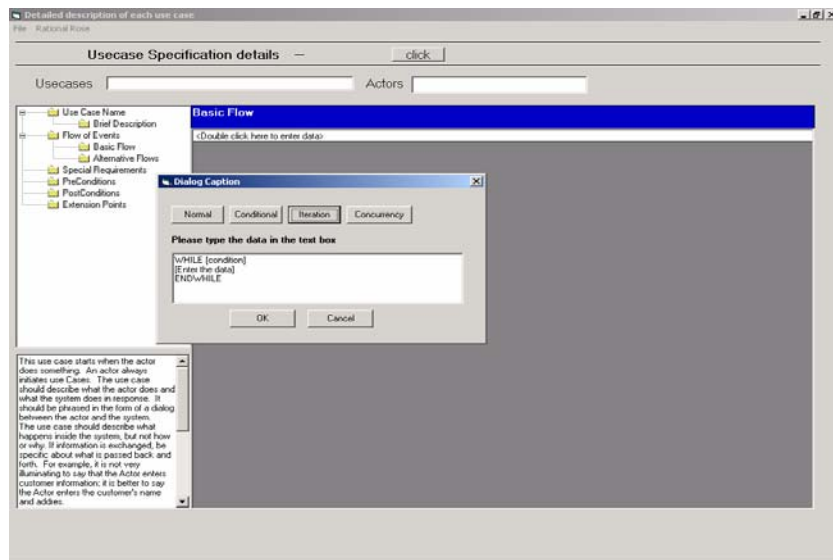


Fig. 7. The environment for use-case specification

### 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. 8. The user can select the use case to realize with the glossary chosen. When the collaboration diagrams are generated, the tool can distribute the behavior and generate the class model in Rational Rose.

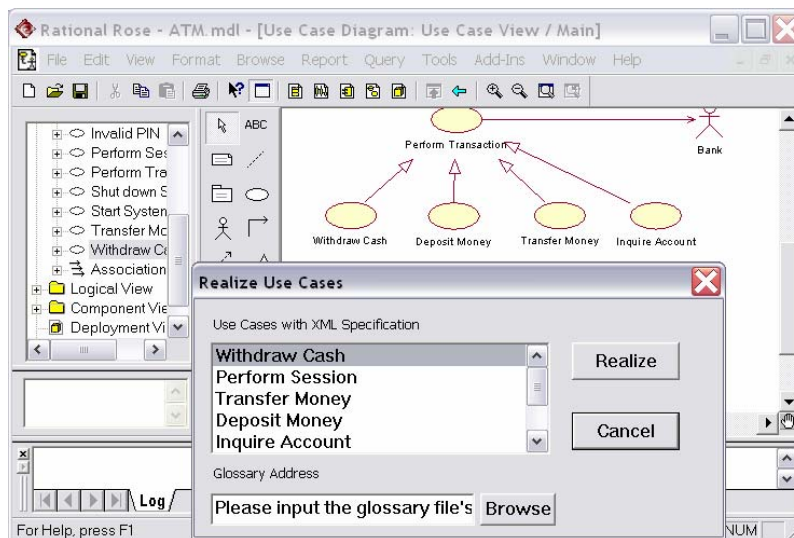


Fig. 8. The environment for use-case realization cooperated with Rational Rose

## 4. Conclusion

A methodology for natural language requirements elicitation, use-case modeling and use-case driven analysis and design is presented. The methodology comprises the good practices from both the natural language requirements analysis and the use-case driven analysis and design. Use-case language schemas are proposed to normalize the use-case specifications, and the methods to automate the 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 may be to implement features of software architecture analysis and integrated it with UCDA and Rational Rose.

## References

- [1] Alistair Cockburn, *Writing Effective Use Cases*. Addison-Wesley, 2000.
- [2] P. Du Bois, E. Dubois, J.M. Zeippen, "On the use of a formal RE language-the generalized railroad crossing problem". *Proceedings of the Third IEEE International Symposium on Requirements Engineering*, 1997.
- [3] X. Li, Z. Liu, J. He, "Formal and use-case driven requirement analysis in UML". *25th Annual International Computer Software and Applications Conference, COMPSAC 2001*.
- [4] Nik Boyd, "Using Natural Language in Software Development". *Journal of Object-Oriented Programming* 11(9), 45-55, 1999.
- [5] Scott P. Overmyer, Benoit Lavoie, Owen Rambow, "Conceptual Modeling through Linguistic Analysis Using LIDA". *Proceedings of the 23rd International Conference on Software Engineering, ICSE 2001*, 401-410, May 2001, Toronto, Ontario, Canada.
- [6] L.C. Niba, "The NIBA workflow: From textual requirements specifications to UML-schemata". *Proceedings of International Conference on Software & Systems Engineering and their Applications, ICSSEA 2002*, Paris, December 2002.
- [7] Paul Roberts, *Patterns of English*. Harcourt, Brace and Company, 1956.
- [8] Russell Abbott, "Program Design by Informal English Descriptions". *Communications of the ACM* 26(11):882-894, Nov. 1983.
- [9] Motoshi Saeki, Hisayuki Horai, Hajime Enomoto, "Software Development Process from Natural Language Specification". *Proceedings of the 11<sup>th</sup> International Conference on Software Engineering, ICSE-11*, 1989.