Framework for using a Natural Language Approach to Object Identification

Mosa Emhamed Elbendak
Northumbria University.
Camden Str.
Newcastle upon Tyne, NE2 1XE
mosa.elbendak@unn.ac.uk

Abstract

Object-oriented analysis and design has now become a major approach in the design of software system. This paper presents a method to automate natural language requirements analysis for object identification and generation based on the Parsed Use Case Descriptions (PUCDs) for capturing the output of the parsing stage. We employ Use-Case Descriptions (UCDs) as input into the whole framework of identification of classes and relationship. PUCD is used to extract nouns, verbs, adjectives and adverbs from use case descriptions as part of an identification process to identify objects/classes and relationships. We refine classes by human expert to produce a class model as output.

Keywords

Requirements specifications, object-oriented, parsing, analysis, Natural Language Processing.

1 Introduction

The process of requirements identification is considered one of the most critical and difficult tasks in database design because most of the input to this process is in natural languages, such as English, which are inherently ambiguous. Developers need to interact with users in their language. Also they need to review and analyse documents written in natural language. This paper presents the work we have achieved so far to find a solution to the problem of automatic identification of objects/classes and relationships from a Requirements Specifications (RS) written in a natural language. Firstly, we review existing literature on the subjects of requirements specifications, object identification, and conceptual database design. The different techniques adopted in the natural language processing systems that attempt to transform natural language to conceptual models have been reviewed. Moreover, we have reviewed the rules to convert English sentences into ER and EER diagrams to determine entity types, attribute types and relationship types.

We intend to employ use-case descriptions as input into the whole framework of identification of classes and relationships, using an existing parsing tool to identify noun phrases, verb phrases, adjectives and adverbs. We propose an intermediate representation called Parsed Use Case Descriptions (PUCD) for capturing the output of the parsing stage, which is then used in subsequent steps. A PUCD is a set of original sentences, parsed sentences, nouns, verbs, adjectives and adverbs, which we use to extract nouns, verbs, adjectives and adverbs from use case descriptions. The next step is the identification process to identify objects/classes, attributes, operations, associations, aggregations and inheritance so as to produce a class model. We refine classes by human expert. In addition to the literature review, the work achieved to date includes an outline of the proposed method for object identification, which is based on existing work on how to map English sentences into conceptual models. The next step identify objects/classes, attributes, operations, and the association, aggregation and inheritance abstractions to produce a class model by applying a set of rules.

2 Motivation

Our motivation is therefore to identify automatically objects/classes and relationships from requirements specifications written in a natural language, e.g. English, so as to increase efficiency in the use of scarce resources and to reduce errors in dealing with complex requests. Therefore we investigate how natural language processing tools and techniques can be used to support the Object-Oriented Analysis (OOA) process. We assume that an English description of the software problem to be solved has already been written. This can be an initial description of the problem or a more detailed list of requirements. We employ Use Case Descriptions (UCDs) as input for identifying classes and relationships as these are well-structured texts . Automation of the Systems Development Life Cycle (SDLC) can alleviate the critical problems of ambiguity, inconsistencies and conflicts in functional requirements [11]. Use case descriptions are very effective in analyzing and capturing functional requirements. They play a major role in defining the processes and actors who can be part of the shareholders of the system. They can be extended, automated and implemented to achieve complete, consistent, and conflict free requirements specification.

The contribution of this paper is to develop a PUCD automatically from use case descriptions as input to extract nouns, verbs, adjectives and adverbs.

Student Research Workshop, RANLP 2009 - Borovets, Bulgaria, pages 23–28

3 Challenges

Handling of Cycles. Software engineering tools nearly always involve a number of cycles with discussion at the end of each iteration on each solution. Thus we have the well-known first- and second-cut approaches. This style of working facilitates the refinement of the models developed. In the work described here we anticipate that two or three cycles will be needed to optimise convergence on an agreed outcome. A framework has to be constructed for handling the cycles.

Use of Natural Language (NL). There are many difficulties often associated with the use of NL which can be summarised as following:

- The ambiguity and complexity of NL are major problems in requirements specifications, as they may lead to misunderstanding between the different users, which most likely will badly affect customer satisfaction with the implementation produced. Furthermore any errors, mistakes or inconsistencies incurred at this stage can be very costly later especially when a system has already been implemented. It has been reported that the cost difference to correct an error in the early stage compared with leaving it till the end is 1:100. See [18, 2, 13].
- The analysis process is considered to be one of the most critical and difficult tasks because most of the input to this process is in natural language such as English.
- Automatic identification of objects/classes and relationships is potentially faster than manual identifications but may be less accurate.
- There is no standard method for automatically identifying objects and classes from English sentences.
- With NLP it is now possible to distinguish nouns and verbs but it is not so easy to classify the verbs as particular types of relationships such as association, aggregation or inheritance, in the identification process.
- There is little or no adaption of standards like UML for expressing requirements specification (RS) for the purpose of object identification.

4 Background and Related Work

Previous studies provide some rules for mapping natural language elements to object-oriented concepts. However, it appears that the coverage is incomplete. For example Abbott [1] first suggested that nouns indicate classes and objects, while verbs can denote behaviours. Researchers and software designers such as Booch et al., and Liang et al., [3, 12] have come to the conclusion that object identification and the refinement process are an ill-defined task, because of the difficulty of heuristics and the lack of a unified methodology for analysis and design. This is mainly due to the lack of a formalism for object-oriented analysis and design.

Although there are many projects focusing on Computer Aided Software Engineering (CASE) tools for object-oriented analysis and design, there are only a few focusing on the formalisation and implementation of the methodology for the object model creation process. Also they are not developed well for the software design that requires collaborative working among members of a software design project team. Wahono and Far [21, 20] examine the issues associated with the methodology for collaborative object-oriented analysis and design. This system is called OOExpert.

Data Model Generator (DMG) is a rule-based design tool by Tjoa and Berger [19] which maintains rules and heuristics in several knowledge bases and employs a parsing algorithm to access information on a grammar using a lexicon designed to meet the requirements of the tool. During the parsing phase, the sentence is parsed by retrieving necessary information using the rules and heuristics to set up a relationship between linguistic and design knowledge. The DMG has to interact with the user if a word does not exist in the lexicon or the input of the mapping rules is ambiguous. The linguistic structures are then transformed by heuristics into EER concepts. Though there is a conversion from natural language to EER models, the tool has not yet been developed into a practical system.

ER generator by Gomez et al. [8] is another rule-based system that generates E-R models from natural language specifications. The E-R generator consists of two kinds of rules: specific rules linked to semantics of some words in sentences, and generic rules that identify entities and relationships on the basis of the logical form of the sentence and of the entities and relationships under construction. The knowledge representation structures are constructed by a Natural Language Understanding (NLU) system which uses a semantic interpretation approach.

CM-Builder by Harmain and Gaizauskas [9] is a natural language based CASE tool which aims at supporting the analysis stage of software development in an object-oriented framework. The tool documents and produces initial conceptual models represented in the Unified Modelling Language. The system uses discourse interpretation and frequency analysis in linguistic analysis. For example, attachment of postmodifiers such as prepositional phrases and relative clauses is limited. Other shortcomings include the state of the knowledge bases which are static and not easily updateable nor adaptive. Meziane and Vadera [15] and Farid [7] implemented a system for the identification of VDM data types and simple operations from natural language software requirements. The system first generates an Entity-Relationship Model (ERM) from the input text followed by VDM data types from the ERM.

Mich and Garigliano [17] and Mich [16] described an NL-based prototype system, NL-OOPS, that is aimed at the generation of object-oriented analysis models from natural language specifications. This system demonstrated how a large scale NLP system called LOLITA can be used to support the OO analysis stage.

Some researchers, also advocating NL-based systems, have tried to use a controlled subset of a natu-

ral language to write software specifications and build tools that can analyse these specifications to produce useful results. Controlled natural languages are developed to limit the vocabulary, syntax and semantics of the input language. Macias and Pulman [14] discuss some possible applications of NLP techniques, using the CORE Language Engine, to support the activity of writing unambiguous formal specifications in English.

The research work described above has provided valuable insights into how NLP can be used to support the analysis and design stages of software development. However, each of these approaches has weaknesses, which means that as yet NL-based CASE tools have not emerged into common use for OO analysis and design. Abbott and Booch's work describes a methodology, but they have not produced a working system which implements their ideas. Meziane produced workable systems but these required an unacceptable level of user interaction such as accepting or rejecting noun phrases to be represented in the final model on a sentence by sentence basis as the requirements document is processed.

Mich and Garigliano's approach, which is closest to our own, is reliant on the coverage of a very large scale knowledge base and the impact of (inevitable) gaps in this knowledge base on the ability of the system to generate usable class models is unclear. It is also worth noting that none of these systems, so far as we are aware, has been evaluated on a set of previously unseen software requirements documents from a range of domains. This ought to become a mandatory methodological component of any research work in this area, as it has in other areas of language processing technology, such as the DARPA-sponsored Message Understanding Conferences (see, e.g. [10]).

5 Research Method

Figure 1 outlines the research activities involved in the research method as follows. The method begins with the Requirements Specification (RS) that describes the structure and behaviour of the system. RS are usually written in Natural Language (NL) e.g., English. NL enables non-technical users to understand the requirements. NL needs to be analyzed, transformed and restructured into a form used as a notation for software requirements specifications. NL includes text from different linguistic levels such as words, sentence and meaning.

The object identification process uses RS to identify fundamental elements (e.g., classes, attributes, operations and relationships) of a conceptual design. The process to generate a conceptual model uses a diagrammatic notation (e.g., UML class diagram or ERM). Figure 1 appears to be recursive but in practice three cycles are likely to be sufficient. The purpose of the first cycle is to check if there is any error in the requirements specifications; the purpose of the second cycle is to correct the errors by refinement to the requirements specification; the purpose of the third cycle completes the whole process by a verification and validation process which checks whether the class model conforms to the original RS. This model therefore includes object identification, conceptual design generation, refinement and verification and validation.

The work described here will make a greater use of automation than earlier approaches, by considering a generated structured output PUCD automatically. The automation is assisted by the sentence-based nature of the parser and the use of a complex set of rules to assist with object identification.

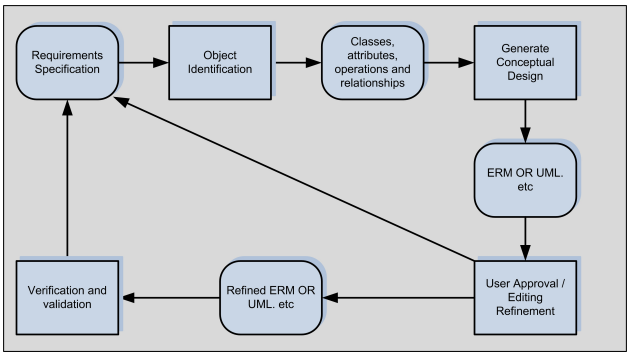

Fig. 1: *Outline of research method*

6 Overview of the Proposed Method

Figure 2 gives an overview of the identification of classes and relationship. Use Case Descriptions (UCDs) represent input to the process as a whole. As explained in more detail later, the parsing process as a whole involves as preliminaries a tokenizer, sentence splitter, part-of-speech tagger and chunking, followed by the parser itself. The PUCD generated is a set of original sentences, parsed sentences, nouns, verbs, adjectives and adverbs. A preliminary class model is generated from the PUCD, which is then refined by a human expert. The steps to design the class diagram from NL are listed below:

Step 1: Parse the use-case description(s) using Memory-Based Shallow Parser (MBSP) to generate noun phrases and verb phrases.

Step 2: Generate PUCD from the output of step 1.

Step 3: Identify classes, and association, aggregation and inheritance abstractions from PUCD objects/classes, using the rules to produce a class model.

Step 4: Refine the output of step 3 using a human expert.

6.1 Use Case Descriptions (UCDs)

UCDs written in a natural language are usually employed for specifying of functional requirements. The format of a use case is not standardized. UCD takes a function requirements text file containing the software requirements. We impose no limitations on the form of the requirements document provided that it is written in English. It can be in the structure of a general problem statement describing the software problem or a list of more detailed functional requirements.

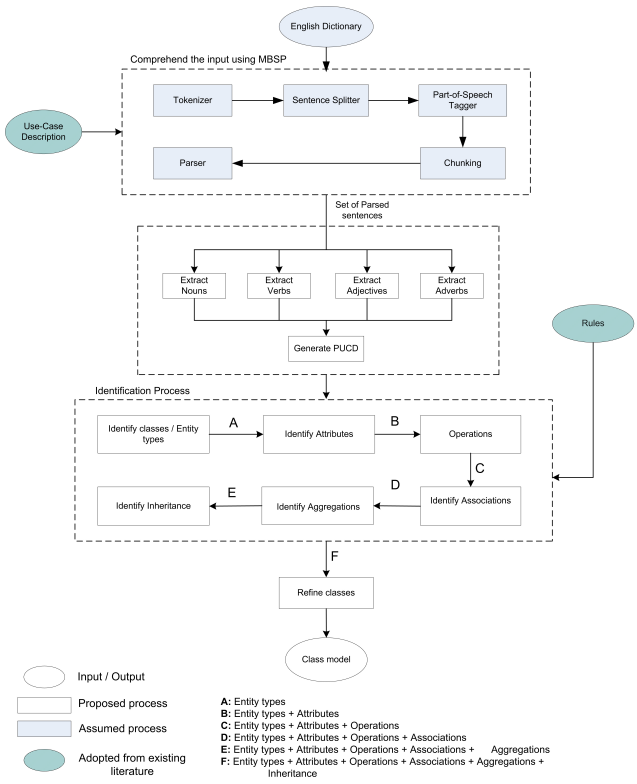

Fig. 2: *Overview of Identification of classes and relationships*

6.2 Comprehension of the Input using MBSP

MBSP is an essential component in text analysis systems for text mining applications such as information extraction and question answering. See [22]. Shallow parsing gives only a partial analysis of the syntactic structure of sentences as opposed to full-sentence parsing. The parsing includes detecting the main constituents of sentences (for example noun phrases (NPs) and verb phrases (VPs)). The MBSP for English consists of the following modules:

(a) Tokenizing: The tokenizer splits a plain text file into tokens. This includes, e.g., separating words and punctuation, identifying numbers, and so on.

(b) Sentence Splitting: The sentence splitter identifies sentence boundaries.

(c) Part-of-Speech (POS) Tagging: The POS tagger assigns to each word in an input sentence its proper part of speech such as noun, verb and determiner to reflect the words syntactic category. See [5, 4].

(d) Chunker: The chunking involves the process of detecting the boundaries between phrases (for example noun phrases) in sentences. See [6]. Chunking can be regarded as light parsing. In MBSP, NL chunking and bracket prediction is applied for the chunking purposes.

(e) Parsing: The parsing here means the process of determining the syntactic structure of a sentence given a formal description of the allowed structures in the language called a *Grammar*.

6.3 Parsed Use-Case Description (PUCD)

The inputs for the PUCD are parsed and tagged text. The main purpose of PUCD is to extract nouns, verbs, adjective and adverbs so as to collect the class/entity type, attribute and relationship from the tagged input.

In some cases the use of one use case description may not be enough to provide all of the information that we need. Therefore, it is recommended to employ more than one use case description to cover all the information needed on properties such as attributes and relationships to produce a class model.

The parsed and tagged text PUCD is defined as a set of tuples as follows:

PUCD = {< original sentence, parsed sentence, Ns, Vs, ADJs, ADVs >}.

Ns = {< N, tag>}, where N is any noun and tag ∈ {NN, NNP, NNPS, NNS}

Vs = {< V, tag>}, where V is any verb and tag ∈ {VB, VBD, VBG, VBN, VBZ}

ADJs = {< ADJ, tag>}, where ADJ is any adjective and tag ∈ {JJ, JJR, JJS}

ADVs = {< ADV, tag>}, where ADV is any adverb and tag ∈ {RB, RBR, RBS}

OS is an original sentence.

PS is a parsed sentence

6.4 Object Identification Process

Details of the object identification process are given in Figure 2. After we extract nouns, verbs, adjectives and adverbs from the generated PUCD, we can then identify classes/entities, attributes and relationships using identification process rules.

Below we illustrate how classes, attributes and relationships are identified:

6.4.1 Identifying Classes/Entities

Figure 2 shows the identification process for identifying classes/entity types, attributes, operations and relationships. We describe more details for each one in the identification process.

The first step in identifying classes is to produce a list of candidate classes. Using PUCD and applying the rules, this can be done by considering all the basic nouns in the PUCD.

A class is defined as follows:

$$C := \{< C_n, \text{ATT}, B, R >\}$$

where C_n is a class name, ATT is a set of attributes, B is a set of behaviour or operations and R is a set of relationships.

We identify a list of candidate classes and attributes as follows:

1. Determiners (such as: a, an, the, each, and, with, etc.) do not play a crucial role at this stage of identification, so they are ignored.

2. Plural noun phrases are converted to their singular form because class names in UML are given in the singular. For example, *customers* is changed to *customer*, and *order items* to *order item*.

3. Redundant candidates are removed from the list of the output PUCD as they are not needed. An exact string matching technique can be used to compare the candidates in the list with each other. For example, customer in our example sentences is a redundant customer which appears many times in the text. The same is done for all other candidates.

Example 1: Identifying objects/classes and relationship with use case description as input.

This example is a very simple one to demonstrate the technique. The inputs for the Parsed Use-Case Description (PUCD) are parsed and tagged text. The main purpose of PUCD is to extract nouns, verbs, adjectives and adverbs that indicate the class/entity type, attribute and relationship from the tagged input. In this example we show the effect of PUCD on one original sentence. This example is simple but we also used many other use case descriptions of varying complexity as input to show the identification of objects/classes and relationships in the production of a class model.

PUCD = {{< OS: Some customers will search for specific CDs or CDs by specific artists, while other customers will want to browse for interesting CDs in certain categories (e.g., rock, jazz, classical), PS:(TOP (S (NP (NNS Customers)) (VP (MD will) (NP (NN access)) (NP (NP (DT the) (NNP Internet) (NNS sales) (NN system)) (SBAR (S (VP (TO to) (VP (VB look) (PP (IN for) (NP (NP (NNS CDs)) (PP (IN of) (NP (NN interest))))))))))) (. .))), Ns:{< NNS Customers >, < NN Access >, < NNP Internet >, < NNS Sales >, < NN System >, < NN Interest>}, Vs: {< Look >} >,

6.4.2 Identify Attributes

A class A has a set of attributes $ATTs$ that describe information about each object:

$$ATT := \{A | A :=< A_n, T >\}$$

where each attribute A has an attribute name A_n and a type T.

The first step in the attribute identification process is to extract ADJ and ADV written in the PUCD and apply the attribute rules; this can be done by considering all the basic adjectives and adverbs in the PUCD.

6.4.3 Identifying Relationships

There are four basic kinds of relationships: Association, Aggregation, Composition and Inheritance. Each class C has a set of relationships R. Relationship is represented by relationship type, related class and cardinality. A relationship R is defined as follows:

$R := \{\text{rel} | \text{rel} :=< \text{RelType}, \text{relC}, \text{Cr} >\}$

where RelType is a relationship type (e.g., associated with, aggregation, composition and Inherits), relC is a related class and Cr is a cardinality.

7 Result of Building an Initial Class Model using semi-automatic Means

Figure 3 shows a class diagram of the CD Selections Internet System (Place Order Use-Case View) written as functional requirements into the UCDs. This model shows 17 classes drawn as solid rectangles. These classes are linked to each other with associations represented by lines between the class boxes.

We review the production of the refined list of candidate classes and attributes and a list of candidate relationships. Figure 3 shows the result of building an Initial Class Model using semi-automatic means from UCDs as input. The original sentence goes through certain stages: parsed by Memory-based Shallow Parser (MBSP) into tokens, split into sentences, tagged with part-of-speech flag, and identification of noun phrases, verb phrases, adjective phrases and adverb phrases. The next step shows how to extract nouns, verbs, adjectives and adverbs by applying the rules we have identified for classes, attributes, operations and relationships. The candidate classes identified are *Customer, Search Request, CD, CD List, Review*. Three different types of search requests were revealed: *Title Search, Artist Search, Category Search*. By applying the rules to the brief description an additional candidate class identified was *Order*. By reviewing the verbs, contained in this use case, we saw that a *Customer* places an *Order* and that a *Customer* makes a *Search Request*.

Using verbs for identifying relationships is not always straight-forward: a verb may indicate an association or an aggregation or inheritance. The use of NLP to distinguish between these types of relationship is a non-trivial problem, which still needs to be addressed but it is hoped that the use of the whole structure of the PUCD will provide an advance in this area.

We identified a set of attributes for the *Customer* (name, address, e-mail and credit card) and for the *Order* (CDs to purchase and quantity) classes and uncovered additional candidate classes *CD Categories* and *Credit Card Center*. Finally, we realized that the *Category Search* class used the *CD Categories* class, and also identified three subclasses of *CD Categories*, namely *Rock, Jazz, Classical*.

Acknowledgments

I would like to express my sincere gratitude to my supervisor Dr. Nick Rossiter and Dr. Paul Vickers for agreeing to supervise my research and providing much of his valuable time to that, for all the good discussions. Many thanks to the Cultural Affairs Department, Libyan People's Bureau, Paris, for the help and financial support.

I would be grateful to Dr. Akhtar Ali for his help in the initial formulation of the work described here.

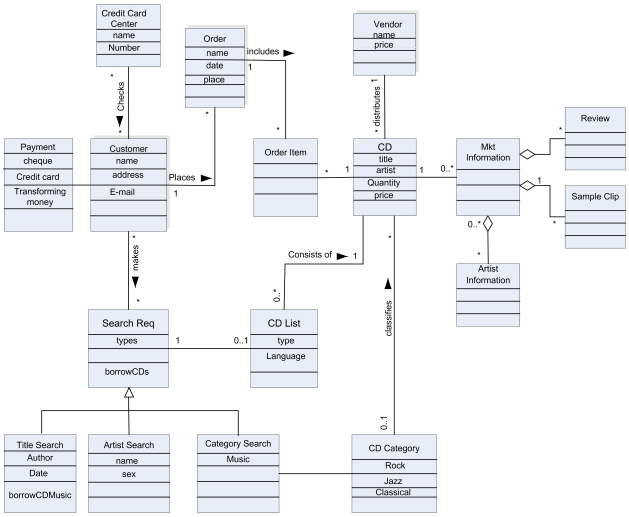

Fig. 3: *Class Model from Use Case Descriptions*

8 Conclusion and Further Work

In this paper we outline an approach that we believe may help to strengthen the process of object identification. We have developed a method called Parse Use-Case Descriptions (PUCDs) to extract nouns, verbs, adjectives and adverbs from use case description. This model is then used for the identification of objects/classes, their attributes, and the static relationships among them to produce a class model. We presented a refinement that generates the class model by using a human expert.

In further work we will focus in with more realistic examples on developing the full method for automatically identifying objects/classes and relationships from RS. We will investigate available technologies and tools to be used, design a system architecture and implement a prototype to realise the identification of entities, attributes and relationships. We will also assess the differences between manually and automatically identifying classes and relationships and evaluate the prototype both in its own right and in a comparison with existing work.

References

[1] R. J. Abbott. Program design by informal English descriptions. *Commun. ACM*, 26(11):882–894, 1983.

[2] B. W. Boehm. *Software Engineering Economics (Prentice-Hall Advances in Computing Science and Technology Series)*. Prentice Hall PTR, October 1983.

[3] G. Booch, J. Rumbaugh, and I. Jacobson. *Object-oriented Analysis and Design With Application*. Addison-Wesley Longman Publishing, Inc., United States of America, 1991.

[4] E. Brill. A simple rule-based part of speech tagger. In *ANLP*, pages 152–155, 1992.

[5] E. Brill. Some advances in transformation-based part of speech tagging. In *AAAI*, pages 722–727, 1994.

[6] W. Daelemans, A. van den Bosch, J. Zavrel, J. Veenstra, S. Buchholz, and B. Busser. Rapid development of nlp modules with memory-based learning. In *In Proceedings of ELSNET in Wonderland*, pages 105–113, 1998.

[7] M. Farid. From English to Formal Specifications, Department of Mathematics and Computer Science. 2000.

[8] F. Gomez, C. Segami, and C. Delaune. A System for the Semi-automatic Generation of E-R Models from Natural Language Specifications. *Data Knowl. Eng.*, 29(1):57–81, 1999.

[9] H. M. Harmain and R. Gaizauskas. Cm-builder: A natural language-based case tool for object-oriented analysis. *Automated Software Engg.*, 10(2):157–181, 2003.

[10] L. Hirschman. The evolution of evaluation: Lessons from the message understanding conferences. *Computer Speech and Language*, 12(4):281–305, 1998.

[11] A. M. Langer. *Analysis and Design of Information Systems*. SpringerLink, United Kingdom, London, 2008.

[12] Y. Liang, D. West, and F. A. Stowel. An approach to object identification, selection and specification in object-oriented analysis. *Inf. Syst. J.*, 8(2):163–, 1998.

[13] D. Liu, K. Subramaniam, A. Eberlein, and B. H. Far. Natural language requirements analysis and class model generation using ucda. In *IEA/AIE'2004: Proceedings of the 17th international conference on Innovations in applied artificial intelligence*, pages 295–304. Springer Springer Verlag Inc, 2004.

[14] B. Macias and S. G. Pulman. A method for controlling the production of specifications in natural language. *Comput. J.*, 38(4):310–318, 1995.

[15] F. Meziane and S. Vadera. Obtaining e-r diagrams semi-automatically from natural language specifications. In *ICEIS (1)*, pages 638–642, 2004.

[16] L. Mich. Nl-oops: from natural language to object oriented requirements using the natural language processing system lolita. *Nat. Lang. Eng.*, 2(2):161–187, 1996.

[17] L. Mich and R. Garigliano. A linguistic approach to the development of object oriented systems using the nl system lolita. In *ISOOMS '94: Proceedings of the International Symposium on Object-Oriented Methodologies and Systems*, pages 371–386, London, UK, 1994. Springer-Verlag.

[18] R. J. Pooley, R. G. Dewar, and K. Li. Object-oriented analysis using natural language processing. 2007.

[19] A. M. Tjoa and L. Berger. Transformation of Requirement Specifications Expressed in Natural Language into an EER Model. In *Proceedings of the 12th International Conference on the Entity-Relationship Approach*, pages 206–217, London, UK, 1994. Springer-Verlag.

[20] R. S. Wahono and B. H. Far. Hybrid Reasoning Architecture for Solving the Object Classes Identification's Problems in the OOExpert System. In *Proceedings of the Annual Conference of JSAI*, pages 351–360, Washington, DC, USA, 2001. IEEE Computer Society.

[21] R. S. Wahono and B. H. Far. OOExpert: Distributed Expert System for Automatic Object-oriented Software Design. In *ICCI '02: Proceedings of the 13th Annual Conference of Japanese Society fo Artificial*, pages 456–457, Tokyo, Japan, 2002. Computer Society.

[22] J. Zavrel and W. Daelemans. Feature-rich memory-based classification for shallow nlp and information extraction. In *Text Mining*, pages 33–54. 2003.