A CONTROLLED NATURAL LANGUAGE INTERFACE TO CLASS MODELS

Imran Sarwar Bajwa
School of Computer Science, The University of Birmingham, Edgbaston, Birmingham, U.K.

M. Asif Naeem
Department of Computer Science,University of Auckland, Auckland, New Zealand

Ahsan Ali Chaudhri
Director of Academic Programs, Queens Academic Group, Auckland, New Zealand

Shahzad Ali
Department of Computer Science and Engineering, University of Electronic Science & Technology, Chengdu, China

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS

Keywords: Natural Language Interface, Controlled Natural Language, Natural Language Processing, Class Model, Automated Object Oriented Analysis, SBVR.

Abstract: The available approaches for automatically generating class models from natural language (NL) software requirements specifications (SRS) exhibit less accuracy due to informal nature of NL such as English. In the automated class model generation, a higher accuracy can be achieved by overcoming the inherent syntactic ambiguities and semantic inconsistencies in English. In this paper, we propose a SBVR based approach to generate an unambiguous representation of NL software requirements. The presented approach works as the user inputs the English specification of software requirements and the approach processes input English to extract SBVR vocabulary and generate a SBVR representation in the form of SBVR rules. Then, SBVR rules are semantically analyzed to extract OO information and finally OO information is mapped to a class model. The presented approach is also presented in a prototype tool NL2SBVR*via*SBVR that is an Eclipse plugin and a proof of concept. A case study has also been solved to show that the use of SBVR in automated generation of class models from NL software requirements improves accuracy and consistency.

1 INTRODUCTION

In natural language (NL) based automated software engineering, the NL (such as English) software requirements specifications are automatically transformed to the formal software representations such as UML (Bryant, 2008) models. The automated analysis of the NL software requirements is a key phase in NL based automated software modelling such as UML (OMG, 2007) modelling. In last two decades, a few attempts have been made to automatically analyze the NL requirement specification and generate the software models such as UML class models e.g. NL-OOPS (Mich, 196), D-H (Delisle, 1998), RCR (Börstler, 1999), LIDA (Overmyer, 2001), GOOAL (Perez-Gonzalez, 2002), CM-Builder (Harmain, 2003), Re-Builder (Oliveira, 2004), NL-OOML (Anandha, 2006), UML-Generator (Bajwa, 2009), etc. However, the accurate object oriented (OO) analysis is still a challenge for NL community (Denger, 2003), (Ormandjieva, 2007), (Berry, 2008). The main hurdle in addressing this challenge is ambiguous and inconsistent nature of NLs such as English. English is ambiguous because English sentence structure is informal. (Bajwa, 2007) Similarly, English is inconsistent as majority of English words have multiple senses and

Bajwa I., Naeem M., Chaudhri A. and Ali S..
A CONTROLLED NATURAL LANGUAGE INTERFACE TO CLASS MODELS.
DOI: 10.5220/0003509801020110
In *Proceedings of the 13th International Conference on Enterprise Information Systems* (ICEIS-2011), pages 102-110
ISBN: 978-989-8425-54-6
Copyright © 2011 SCITEPRESS (Science and Technology Publications, Lda.)

a single sense can be reflected by multiple words in English.

In this paper, the major contribution is three folds. Firstly, a Semantic Business vocabulary and Rule (SBVR) (OMG, 2008) based approach is presented to generate a controlled (unambiguous and consistent) representation of natural language software requirements specification. Secondly, we report the structure of the implemented tool NL2UML*via*SBVR that is able to automatically perform object-oriented analysis of SBVR software requirements specifications. Thirdly, a case study is solved that was originally solved with CM-Builder (Harmain, 2003) and the results of the case study are compared with available tools (used for automated OOA) to evaluate the NL2UML*via*SBVR tool.

Our approach works as the user inputs a piece of English specification of software requirements and the NL to SBVR approach generates SBVR (an adopted standard of the OMG) (OMG, 2008) based controlled representation of English software requirement specification. To generate a SBVR representation such as SBVR rule, first the input English text is lexically, syntactically and semantically parsed and SBVR vocabulary is extracted. Then, the SBVR vocabulary is further processed to construct a SBVR rule by applying SBVR's *Conceptual Formalization* (OMG, 2008) and *Semantic Formulation* (OMG, 2008). The last phase is extraction of the OO information (such as classes, methods, attributes, associations, generalizations, etc) from the SBVR's rule based representation.

The remaining paper is structured into the following sections: Section 2 explains that how SBVR provides a controlled representation to English. Section 3 illustrates the architecture of NL2UML*via*SBVR. Section 4 presents a case study. The evaluation of our approach is presented in section 5. Finally, the paper is concluded to discuss the future work.

2 SBVR BASED CONTROLLED NATURAL LANGUAGE

SBVR was originally presented for business people to provide a clear and unambiguous way of defining business policies and rules in their native language (OMG, 2008). The SBVR based controlled representation is useful in multiple ways such as due to its natural language syntax, it is easy to understand for developers and users. Similarly, SBVR is easy to machine process as SBVR is based on higher order logic (first order logic). We have identified a set of characteristics of SBVR those can be used to generate a controlled natural language representation of English.

2.1 Conceptual Formalization

SBVR provides rule-based conceptual formalization that can be used to generate a syntactically formal representation of English. Our approach can formalize two types of requirements: The structural requirements can be represented using SBVR structural business rules, based on two alethic modal operators (OMG, 2008): "*it is* necessary *that*..." and "*it is* possible *that*..." for example, **It is possible** that a customer *is a* member. Similarly, the behavioural requirements can be represented using SBVR operative business rule, based on two deontic modal operators (OMG, 2008): "*it is* obligatory *that* ..." and "*it is* permitted *that* ..." for example, **It is obligatory** that a customer *can borrow* **at most** two books.

2.2 Semantic Formulation

SBVR is typically proposed for business modeling in NL. However, we are using the formal logic based nature of SBVR to semantically formulate the English software requirements statements. A set of logic structures called semantic formulations are provided in SBVR to make English statements controlled such as atomic formulation, instantiate formulation, logical formulation, quantification, and modal formulation. For more details, we recommend user SBVR 1.0 document (OMG, 2008).

2.3 Textual Notations

SBVR provides couple of textual notations. Structured English is one of the possible SBVR notations, given in SBVR 1.0 document, Annex C (OMG, 2008), is applied by prefixing rule keywords in a SBVR rules. The other possible SBVR notation is Rulespeak, given in SBVR 1.0 document, Annex F (OMG, 2008), uses mixfixing keywords in propositions. Both SBVR formal notations typically help in expressing the natural language propositions with equivalent semantics that can be captured and formally represented as logical formulations.

3 THE NL2UML*via*SBVR

This section explains how English text is mapped to SBVR representation, object oriented analysis and finally generation of a class model. The used approach works in five phases (see figure 1):

- Processing natural language specification
- Extracting Business Vocabulary from NL text
- Generating Business Rules from business vocabulary
- Performing object oriented analysis
- Generating UML Class models

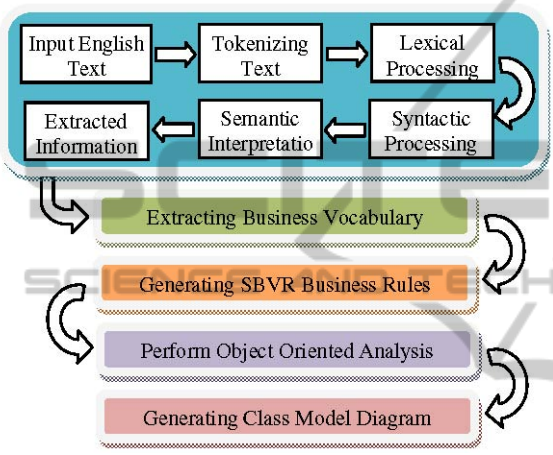

Figure 1: The NL2SBVR Approach.

3.1 Parsing NL Software Requirements

The first phase of NL2UML*via*SBVR is NL parsing that involves a number of sub-processing units (organized in a pipelined architecture) to process complex English statements. The NL parsing phase tokenizes English text and lexically, syntactically and semantically processes the English text.

3.1.1 Lexical Processing

The NL parsing starts with the lexical processing of a plain text file containing English software requirements specification. The lexical processing phase comprises following four sub-phases:

1. The input is processed to identify the margins of a sentence and each sentence is stored in an arraylist.
2. After sentence splitting, each sentence goes through the tokenization. Tokenization works as a sentence "A member can borrow at most two books." is tokenized as [A] [member] [can] [borrow] [at] [most] [two] [books] [.]
3. The tokenized text is further passed to Stanford parts-of- speech (POS) (Toutanova, 2000) tagger v3.0 to identify the basic POS tags e.g. A/DT member/NN can/MD borrow/VB at/IN most/JJS two/CD books/NNS ./. The Stanford POS tagger v3.0 can identify 44 POS tags.
4. The POS tagged text is further processed to extract various morphemes. In morphological analysis, the suffixes attached to the nouns and verbs are segregated e.g. a verb "applies" is analyzed as "apply+s" and similarly a noun "students" is analyzed as "student+s".

3.1.2 Syntactic Processing

We have used an enhanced version of a rule-based bottom-up parser for the syntactic analyze of the input text used in (Bajwa, 2009). English grammar rules are base of used parser. The text is syntactically analyzed and a parse tree is generated for further semantic processing, shown in Figure 2.

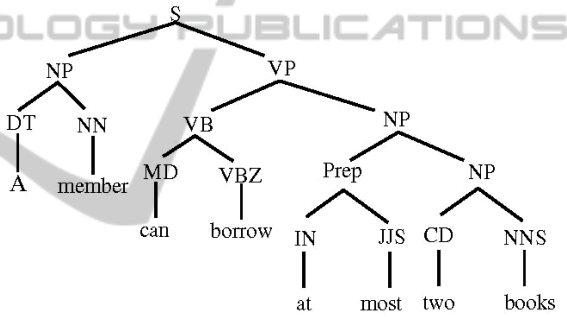

Figure 2: Parsing English text.

3.1.3 Semantic Interpretation

In this semantic interpretation phase, role labelling (Bajwa, 2006) is performed. The desired role labels are actors (nouns used in subject part), co-actor (additional actors conjuncted with '*and*'), action (action verb), thematic object (nouns used in object part), and a beneficiary (nouns used in adverb part) if exists, (see figure 3). These roles assist in identifying SBVR vocabulary and exported as an xml file.

A member can borrow at most two books .
Actor Action Quantity Them. Object

Figure 3: Semantic interpretation of English text.

3.2 SBVR Vocabulary Extraction

The similar rules to extract SBVR vocabulary from English text, we used in (Bajwa, 2011). We have

extended the rules to use in NL to UML translation via SBVR. In NL to SBVR translation phase, the basic SBVR vocabulary e.g. noun concept, individual concept, object type, verb concepts, fact type, etc are identified from the English input that is preprocess in the previous phase. The extraction of various SBVR elements is described below:

1. *Extracting Object Types:* All common nouns (actors, co-actors, thematic objects, or beneficiaries) are represented as the object types or general concept (see figure 4) e.g. belt, user, cup, etc. In conceptual modelling, the object types are mapped to classes.
2. *Extracting Individual Concepts:* All proper nouns (actors, co-actors, thematic objects, or beneficiaries) are represented as the individual concepts.

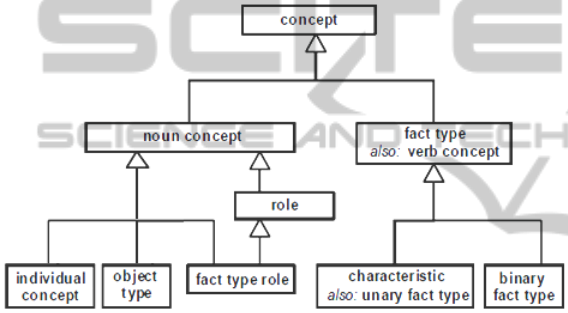

Figure 4: An extract of the SBVR metamodel: concepts.

3. *Extracting Fact Types*: The auxiliary and action verbs are represented as verb concepts. To constructing a fact types, the combination of an object type/individual concept + verb forms a unary fact type e.g. "vision system *senses*". Similarly, the combination of an object type/individual concept + verb + object type forms a binary fact type e.g. belt *conveys* part is a binary fact type.
4. *Extracting Characteristics*: In English, the characteristic or attributes are typically represented using *is-property-of* fact type e.g. "name *is-property-of* customer". Moreover, the use of possessed nouns (i.e. pre-fixed by *'s* or post-fixed by *of*) e.g. student's age or age of student is also characteristic.
5. *Extracting Quantifications:* All indefinite articles (*a* and *an*), plural nouns (prefixed with s) and cardinal numbers (2 or two) represent quantifications.
6. *Extracting Associative Fact Types:* The associative fact types (OMG, 2008) (section 11.1.5.1) (see figure 4) are identified by associative or pragmatic relations in English text. In English, the binary fact types are typical examples of associative fact types e.g. "The belt conveys the parts". In this example, there is a binary association in belt and parts concepts. This association is one-to-many as '*parts'* concept is plural. In conceptual modeling of SBVR, associative fact types are mapped to associations.
7. *Extracting Partitive Fact Type:* The partitive fact types (OMG, 2008) (section 11.1.5.1) (see figure 4) are identified by extracting structures such as "*is-part-of*", "*included-in*" or "*belong-to*" e.g. "The user puts two-kinds-of parts, dish and cup". Here 'parts' is generalized form of '*dish'* and '*cup'*. In conceptual modeling of SBVR, categorization fact types are mapped to aggregations.
8. *Extracting Categorization Fact Types:* The categorization fact types (OMG, 2008) (section 11.1.5.2) (see figure 4) are identified by extracting structures such as "*is-category-of*" or "*is-type-of*", "*is-kind-of*" e.g. "The user puts two-kinds-of parts, dish and cup". Here 'parts' is generalized form of '*dish'* and '*cup'*. In conceptual modeling of SBVR, categorization fact types are mapped to generalizations. All the extracted information shown in figure 5 is stored in an arraylist for further analysis.

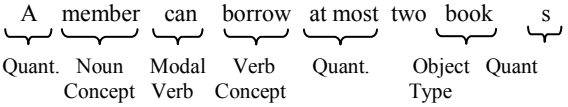

Figure 5: Semantic interpretation of English text.

3.3 SBVR Rules Generation

In this phase, a SBVR representation such as SBVR rule is generated from the SBVR vocabulary in previous phase. SBVR rule is generated in two phases as following:

3.3.1 Applying Semantic Formulation

A set of semantic formulations are applied to each fact type to construct a SBVR rule. There are five basic semantic formulations proposed in SBVR version 1.0 (OMG, 2008) but we are using following three with respect to the context of the scope of proposed research:

1. *Logical Formulation*: A SBVR rule can be composed of multiple fact types using logical operators e.g. AND, OR, NOT, implies, etc. For logical formulation (OMG, 2008), the tokens '*not'* or '*no'* are mapped to negation (¬ a).

Similarly, the tokens '*that*' & '*and*' are mapped to conjunction (a ∧ b). The token '*or'* is mapped to disjunction (a ∨ b) and the tokens '*imply'*, '*suggest'*, '*indicate'*, '*infer'* are mapped to implication (a ⟹ b).

2. *Quantification*: Quantification (OMG, 2008) is used to specify the scope of a concept. Quantifications are applied by mapping tokes like "*more than*" or "*greater than*" to at least n quantification; token "less than" is mapped to at most n quantification and token "*equal to*" or a positive statement is mapped to exactly n quantification.
3. *Modal Formulation*: In SBVR, the modal formulation (OMG, 2008) specifies seriousness of a constraint. Modal verbs such as '*can'* , '' or '*may'* are mapped to possibility formulation to represent a structural requirement and the modal verbs '*should'*, '*must'* or verb concept "*have to*" are mapped to obligation formulation to represent a behavioural requirement.

3.3.2 Applying Structured English Notation

The last step in generation of a SBVR is application of the Structured English notation in SBVR 1.0 document, Annex C (OMG, 2008). Following formatting rules were used: The noun concepts are underlined e.g. student; the verb concepts are italicized e.g. *should be*; the SBVR keywords are bolded e.g. at most; the individual concepts are double underlined e.g. Ahmad, England. Attributes are also italicized but with different colour: e.g. *name*. RuleSpeak (OMG, 2008) is the other available notation in SBVR. The NL2UML*via*SBVR tool supports both notations.

3.4 Object-oriented Analysis

In this phase, finally the SBVR rule is further processed to extract the OO information. The extraction of each OO element from SBVR representation is described below:

1. *Extracting Classes:* All SBVR object types are mapped to classes e.g. library, book, etc.
2. *Extracting* Instances: The SBVR individual concepts are mapped to instances.
1. *Extracting Class Attributes*: All the SBVR characteristics or unary fact types (without action verbs) associated to an object type are mapped to attributes of a class.
2. *Extracting Class Methods*: All the SBVR verb concepts (action verbs) associated to a noun concept are mapped to methods for a particular class e.g. issue() is method of library class.
3. *Extracting Associations*: A unary fact type with action verb is mapped to a unary relationship and all associative fact types are mapped to binary relationships. The use of quantifications with the respective noun concept is employed to identify *multiplicity* e.g. library and book(s) will have one to many association. The associated verb concept is used as caption of association as shown in figure 6.

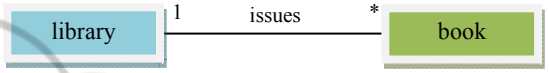

Figure 6: Extracting class associations.

4. *Extracting Generalization*: The partitive fact types are specified as generalizations. The subject-part of the fact type is considered the main class in generalization and object-part of the fact types is considered as the sub class.
5. *Extracting Aggregations*: The categorization fact types are mapped to aggregations. The subject-part of the fact type is considered the main class in aggregation and object-part of the fact types is considered as the sub class.

3.5 Drawing UML Class Model

This phase draws a UML class model by combining class diagram symbols with respect to the information extracted of the previous phase. In this phase, the java graphics functions (drawline(), drawrect(), etc) are used to draw the class diagram symbols.

4 A CASE STUDY

A case study is discussed from the domain of library information systems that was originally presented by Callan (1994) and later on solved by Harmain (2003). The problem statement for the case study is as follows:

> *A library issues loan items to customers. Each customer is known as a member and is issued a membership card that shows a unique member number. Along with the membership number other details on a customer must be kept such as a name, address, and date of birth. The library is made up of a number of subject sections. Each section is denoted by a classification mark. A loan item is uniquely identified by a bar code. There are two types of loan items, language tapes, and books. A language tape has a title*

language (e.g. French), and level (e.g. beginner). A book has a title, and author(s). A customer may borrow up to a maximum of 8 items. An item can be borrowed, reserved or renewed to extend a current loan. When an item is issued the customer's membership number is scanned via a bar code reader or entered manually. If the membership is still valid and the number of items on loan less than 8, the book bar code is read, either via the bar code reader or entered manually. If the item can be issued (e.g. not reserved) the item is stamped and then issued. The library must support the facility for an item to be searched and for a daily update of records.

The problem statement of the case study was given as input to the NL2UML*via*SBVR tool that is an Eclipse plugin implemented in java as a proof of concept. The generated SBVR representation is:

A library *issues* loan items to each customer. Each customer *is known* as a member and *is issued* a membership card that *shows* a unique member number. It is necessary that the membership number and other details on a customer *must be kept* such as a *name*, *address*, and *date-of-birth*. The library *is made up* of a number of subject sections. Each section *is denoted* by a *classification-mark*. A loan item *is identified* by a *bar-code*. There *are* exactly two types of loan items, language tapes, and books. A language tape *has* a *title-language*, and *level*. A book *has* a *title*, and *author*(s). It is possibility that each customer *may borrow* up to at most 8 items. It is possibility that each item *can be borrowed*, *reserved* or *renewed* to *extend* a current loan. When an item *is issued* the customer's *membership-number is scanned* via a bar code reader or *entered* manually. If the membership *is valid* and the number of items on loan at most 8, the book's *bar-code is read*, either via the bar code reader or *entered* manually. It is possibility that if the item *can be issued* the item *is stamped* and then *issued*. It is necessary that the library *must support* the facility for an item to *be searched* and for a daily *update* of records.

After generating the SBVR representation, OOA was performed to extract following information:
There were some synonyms for the used classes such as Item and Loan_Item, Section and Subject_Section. Our system keeps only one of the similar classes. Here, customer and member are also synonyms, but our system is not able to handle such similarities. There is only one wrong class that is Member_Number as it is an attribute. There are two incorrect associations: "Library support facility" is not an association and "Library made up of Subject_sections" is an aggregation but classified as an association.

Table 1: Object Oriented Analysis results.

Type	Count	Details
Classes	10	Library, Loan_Items, Member_Number, Customer, Book, Language_Tape Member, Bar_Code_Reader, Subject_Section, Membership_Card
Attributes	10	name, address, date-of-birth, bar_code, classification_mark, title, author, Level, membership-number, valid
Methods	11	issue(), show(), denote(), identify(), extend(), scan(), enter(), read_barcode(), stamp(), search(). update()
Associations	07	Library issues Loan_Items; Member_Card issued to Member; Library made up of Subject_sections; Customer borrow Loan_items; customer renew Loan_item; customer reserve_Loan_item; Library support facility
Generalizations	02	Loan Items is type-of Language_tapes, Loan Items is type-of Books
Aggregations	00	-
Instances	00	-

A screen shot of a class model generated for the case study shown in figure 7.

5 EVALUATION

We have done performance evaluation to evaluate the accuracy of NL2UML*via*SBVR tool. An evaluation methodology, for the performance evaluation of NLP tools, proposed by Hirschman and Thompson (1995) is based on three aspects:

- Criterion specifies the interest of evaluation e.g. precision, error rate, etc.

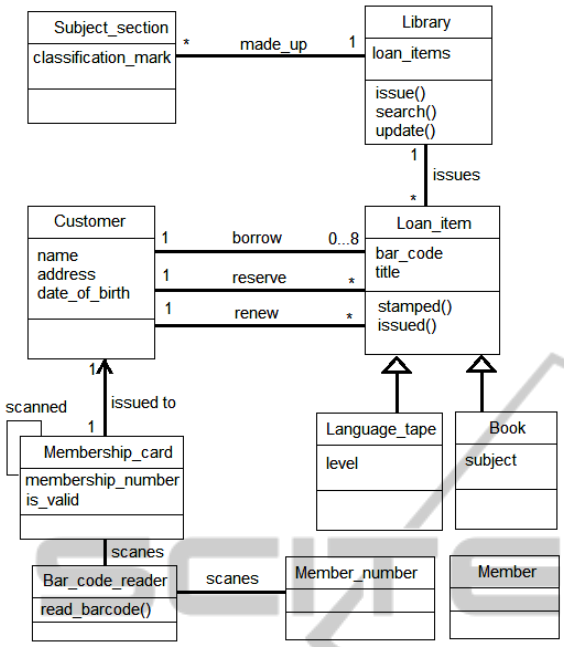

Figure 7: A class model of case study generated by NL2UML*via*SBVR.

- Measure specifies the particular property of system performance someone intends to get at the selected criterion e.g. percent correct or incorrect.
- Evaluation method determines the appropriate value for a given measure and a given system.

As we want to compare the results of performance evaluation with other tools such as CM-Builder (Harmain, 2003), we have a used a similar evaluation methodology used for CM-Builder. Following is the evaluation methodology used to evaluate the performance of NL2UML*via*SBVR.

5.1 Evaluation Methodology

Our evaluation methodology is based on three items, described in (Harmain, 2003):

a. Criterion

For evaluation of the designed system, a criterion was defined that how close are the NL2UML*via*SBVR output to the opinion of the human expert (named sample results). Different human experts produce different representations and can be good or bad analysis. However, we gained a human expert's opinion for the target input and used it as a sample result.

b. Measure

We have used two evaluation metrics: recall and precision. These metrics are extensively employed to evaluate NL based knowledge extraction systems. We can define these metrics as following:

1. *Recall:* The completeness of the results produced by system is called recall. Recall can be calculated by comparing the correct results produced by the system's with the human expert's opinion (sample results). Recall can be calculated by using the following formula also used in (Harmain, 2003):

$$\text{Recall} = \frac{N_{correct}}{N_{sample}}$$

Where $N_{correct}$ is the number of correct results generated by the tool and N_{sample} is the number of sample results (opinion of human expert).

2. *Precision:* The second metrics precision expresses accuracy of the designed system where system accuracy means the correct number of results produced by the system. Precision is measured by comparing designed system's number of correct results by all (incorrect and correct) results produced by the system. Precision is calculated as:

$$\text{Precision} = \frac{N_{correct}}{N_{incorrect} + N_{correct}}$$

Where $N_{incorrect}$ is the number of incorrect results and $N_{correct}$ is the number of correct results.

c. Method

To evaluate the results of NL2UML*via*SBVR, each outcome (class names, attributes names, method names, associations, multiplicity generalizations, aggregations, and instance names) of the NL2UML*via*SBVR's output was matched with the expert's opinion (N_{sample}) (sample solution). The outcome that accurately classified into respective category was declared correct ($N_{correct}$) otherwise incorrect ($N_{incorrect}$). Additionally, the information that was not extracted (or missed) by the NL2SBVR tool but it was given in the human expert's opinion (N_{sample}) was categorized as the missing information ($N_{missing}$).

5.2 Evaluation Results

The results of the case studies were used to calculate recall and precision values as shown in table 2.

Table 2: NL2UML*via*SBVR Evaluation results.

Example	N_{sample}	$N_{correct}$	$N_{incorrect}$	$N_{missing}$	*Rec%*	*Prec%*
Results	40	35	3	2	87.50	92.10

Average recall for English requirement specification is calculated 87.5% while average precision is calculated 92.1%. These results are very encouraging for the future enhancements.

We have also compared the results of NL2UML*via*SBVR with other available tools that can perform automated analysis of the NL requirement specifications. Recall value was not available for some of the tools. We have used the available recall and precision values of the tools for comparison as shown in table 3:

Table 3: A comparison of performance evaluation - NL2UML*via*SBVR vs other tools.

NL Tools for Class Modelling	*Recall*	*Precision*
CM-Builder (Harmain, 2003)	73.00%	66.00%
GOOAL (Perez-Gonzalez, 2002)	-	78.00%
UML-Generator (Bajwa, 2009)	81.29%	87.17%
NL-OOML (Anandha, 2006)	-	82.00%
LIDA (Overmyer, 2001)	71.32%	63.17%
NL2UML*via*SBVR	87.50%	92.10%

Here, we can note that the accuracy of other NL tools used for information extraction and object oriented analysis is well below than NL2UML*via*SBVR.

Moreover, the various tools' functionalities (if available, is automated or user involved) are also compared with NL2UML*via*SBVR as shown in Table 4:

Table 4 shows that besides NL2UML*via*SBVR, there are very few tools those can extract information such as multiplicity, aggregations, generalizations, and instances from NL requirement. Thus, the results of this initial performance evaluation are very encouraging and support both the approach adopted in this paper and the potential of this technology in general.

Table 4: Comparison of NL2UML*via*SBVR with other tools.

Support	CM Builder	LIDA	GOOAL	NL OOML	NL2UML *via*SBVR
Classes	Yes	User	Yes	Yes	Yes
Attributes	Yes	User	Yes	Yes	Yes
Methods	No	User	Yes	Yes	Yes
Associations	Yes	User	Semi-NL	No	Yes
Multiplicity	Yes	User	No	No	Yes
Aggregation	No	No	No	No	Yes
Generalization	No	No	No	No	Yes
Instances	No	No	No	No	Yes

6 CONCLUSIONS

The primary objective of the paper was to address the challenge of addressing ambiguous nature of natural languages (such as English) and generate a controlled representation of English so that the accuracy of machine processing can be improved.

To address this challenge we have presented a NL based automated approach to parse English software requirements specifications and generated a controlled representation using SBVR. Automated object oriented analysis of SBVR specifications of software requirements using the NL2UML*via*SBVR provides a higher accuracy as compared to other available NL-based tools. Besides better accuracy, SBVR has also enabled to extract OO information such as association multiplicity, aggregations, generalizations, and instances as other NL-based tools can't process and extract this information.

Some non-functional requirements in the case study such as "*If the membership is still valid and the number of items on loan less than 8, the book bar code is read*" and "*If the item can be issued (e.g. not reserved) the item is stamped and then issued.*" are not part of the output class model. These are basically constraints and it is our future work to also generate Object Constraint language (OCL) for these natural language constraints.

REFERENCES

Bryant B. R, Lee, B. S., et al. 2008. From Natural Language Requirements to Executable Models of Software Components. In Workshop on S. E. for Embedded Systems:51-58.

OMG. (2007). Unified Modelling Language (UML) Standard version 2.1.2. Object Management Group, Available at: http://www.omg.org/mda/

Mich, L. 1996. NL-OOPS: from natural language to object oriented requirements using the natural language processing system LOLITA. Natural Language Engineering. 2(2):167-181

Delisle, S. Barker, K. Biskri, I. 1998. Object-Oriented Analysis: Getting Help from Robust Computational Linguistic Tools. 4th International Conference on Applications of Natural Language to Information Systems, Klagenfurt, Austria:167-172.

Bajwa, Imran Sarwar, Lee, Mark G., Bordbar, Behzad. 2011. SBVR Business Rules Generation from Natural Language Specification. AAAI Spring Symposium 2011, San Francisco, USA. pp.2-8

Börstler, J. 1999. User - Centered Requirements Engineering in RECORD - An Overview. Nordic Workshop on Programming Environment Research NWPER'96, Aalborg, Denmark:149-156.

Overmyer, S. V., Rambow, O. 2001. Conceptual Modeling through Linguistics Analysis Using LIDA. 23rd International Conference on Software engineering, July 2001

Perez-Gonzalez, H. G., Kalita, J. K. 2002. GOOAL: A Graphic Object Oriented Analysis Laboratory. 17th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA '02), NY, USA: 38-39.

Harmain, H. M., Gaizauskas R. 2003. CM-Builder: A Natural Language-Based CASE Tool for Object-Oriented Analysis. Automated Software Engineering. 10(2):157-181

Oliveira, A., Seco N. and Gomes P. 2006. A CBR Approach to Text to Class Diagram Translation. TCBR Workshop at the 8th European Conference on Case-Based Reasoning, Turkey, September 2006.

Anandha G. S., Uma G.V. 2006. Automatic Construction of Object Oriented Design Models [UML Diagrams] from Natural Language Requirements Specification. PRICAI 2006: Trends in Artificial Intelligence, LNCS 4099/2006: 1155-1159

Bajwa I. S., Samad A., Mumtaz S. 2009. Object Oriented Software modeling Using NLP based Knowledge Extraction. European Journal of Scientific Research, 35(01):22-33

OMG. 2008. Semantics of Business vocabulary and Rules. (SBVR) Standard v.1.0. Object Management Group, Available: http://www.omg.org/spec/SBVR/1.0/

Toutanova. K., Manning, C. D. 2000. Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger. In Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora: 63-70.

Li, K., Dewar, R. G., Pooley, R. J. 2005. Object-Oriented Analysis Using Natural Language Processing, Linguistic Analysis (2005)

Bajwa I. S., Hyder, I. S. 2007. UCD-generator - A LESSA application for use case design, International Conference on Information and Emerging Technologies, 2007. ICIET 2007.

Callan. R. E. 1994. Building Object-Oriented Systems: An introduction from concepts to implementation in C++. In Computational Mechanics Publications, 1994.

Hirschman L., and Thompson, H. S. 1995. Chapter 13 evaluation: Overview of evaluation in speech and natural language processing. In Survey of the State of the Art in Human Language Technology.

Berry M. D., 2008. Ambiguity in Natural Language Requirements Documents. In Innovations for Requirement Analysis. From Stakeholders' Needs to Formal Designs, LNCS-5320/2008:1-7

Ormandjieva O., Hussain, I., Kosseim, L. 2007. Toward A Text Classification System for the Quality Assessment of Software Requirements written in Natural Language. in 4th International Workshop on Software Quality Assurance (SOQUA '07):39-45.

Bajwa, I. S., Choudhary, M. A. (2006) "A Rule Based System for Speech Language Context Understanding" *Journal of Donghua University, (English Edition)* 23 (6), pp. 39-42.

Denger, C., Berry, D. M. Kamsties, E. 2003. Higher Quality Requirements Specifications through Natural Language Patterns. In Proceedings of IEEE International Conference on Software-Science, Technology \& Engineering (SWSTE '03):80-85

Ilieva, M. G., Ormandjieva, O. 2005. Automatic Transition of Natural Language Software Requirements Specification into Formal Presentation. in proc. of Natural Language Processing and Information Systems LNCS- 3513/2005:427-434

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS