NL2 Alloy: A Tool to Generate Alloy from NL Constraints

Imran Sarwar Bajwa[1], Behzad Bordbar[2], Mark Lee[2], Kyriakos Anastasakis[2]
[1]Department of Computer Science & IT
The Islamia University of Bahawalpur
Bahawalpur, 63100
Pakistan
[2]School of Computer Science
University of Birmingham
Birmingham, B15 2TT
UK
imran.sarwar@iub.edu.pk, {b.bordbar, m.g.lee, k.Anastasakis}@cs.bham.ac.uk

JDIM
Journal of Digital Information Management

ABSTRACT: *In this paper, we present a novel approach to generate Alloy code from Natural Language (NL) constraints. The proposed research is basically an extension of two projects, NL2OCL and UML2Alloy.Our method uses Natural Language Processing (NLP) and Model Transformation (MT) to transform constraints expressed in NL to Alloy. We do not directly transform NL to Alloy, instead we use multiple intermediate translations such as NL2SBVR and SBVR2OCL and finally OCL2Alloy. Such intermediate translations help us to monitor the whole process of translation and produce intermediate artifacts such as OCL constraints, which can be used for documentation purposes. and ensure that actual semantics of NL have been transformed to Alloy. The generated Alloy code can then be used to check if the original NL constraints are consistent. We also introduce the use of OMG's standard SBVR in translation of NL to formal languages. The NL2Alloy approach is also implemented as an Eclipse plugin.*

Categories and Subject Descriptors:
I.2.7 [**Natural Language Processing**]: Language models; **F.4.3** [**Formal Languages**]

General Terms:
Natural Language Processing, Language Models

Keywords: UML, Alloy, SBVR, Natural Language, NL2OCL

Received: 28 June 2012, Revised 29 August 2012, Accepted 3 September 2012

1. Introduction

In this paper, we aim to automate the typical process of manual translation of Natural Language (NL) constraints/ specification to a formalism that provides analysis capabilities. The method suggested here is particularly applicable in early stages of software development; it is common during the early stages of the software development process to gather requirements. Typically these requirements are expressed in NL. Designers and analysts then manually transform these requirements to a software specification, usually in the form of UML models. One of the most challenging aspects is turning constraints about the system into formal representations such as OCL and Alloy. In additions, sometimes constraints embedded in the requirements are in conflict and manual inspection of the Natural Language statements to discover the inconsistencies is highly non-trivial. The method suggested here allows automated creation of formal representation from NL and analysis of the modes to discover inconsistencies at early stages of the software development lifecycle.

We propose Alloy as the formalism for the consistency analysis of the requirements specification. Alloy is a well-known language ideal to express complex structural constraints. The language is supported by a tool, the Alloy Analyzer, which provides the ability to analyse Alloy models automatically. In particular the tool can automatically detect and inconsistent models. A valuable feature is its UnSat Core functionality [4], which highlights conflicting statements, making it easier to debug over constrained models.

In order to automate the generation of the Alloy code from NL specifications, a sequence of transformations is used.

First, all NL constraint are syntactically and semantically analysed to generate a logical representation that can be mapped to formal languages such as Alloy. Here, the logical representation is based on the Semantic of Business Vocabulary and Rules (SBVR) [6] standard. Then, SBVR based logical representation is mapped to OCL constraints. Finally, the OCL constraints are mapped to Alloy expressions for a UML class model in OCL2Alloy transformation [8].

The rest of the paper is structured as follows. Section 2 describes the preliminary concepts of the research; section 3 highlights main phases of NL to Alloy approach and the working of NL2Alloy approach is explained with the help of a running example in section 4; section 5 describes results and evaluation of the tool. The paper ends with a conclusion section.

2. Preliminaries

In this section, the primary concepts involved in the presented research such as OCL and Alloy are introduced.

2.1 Alloy

Alloy [4] is a declarative textual modeling language. An Alloy model consists of a number of *signature*, that are supplemented by *field*, *fact* and *predicate declarations*. S*ignatures* denote a set of *atoms*, which are the basic entities in Alloy. F*ields* need to be declared under a *signature* and represent a relation between two or more *signatures*. In essence a *field* represents a set of tuples of atoms. *Facts* are declarative statements in first-order logic that define constrains on the declared *signatures* and *fields*. *Predicates* are in essence parameterized constraints that can be referenced from within other *predicates* or *facts*.

The Alloy language is supported by a tool, the *Alloy Analyzer* which supports fully automated analysis of Alloy models. The tool works by converting the Alloy model to a Boolean formula that is then provided as input to SAT solvers [4]. The Alloy Analyzer has three main functionalities. The *simulation* functionality can produce random instances of the model that conform to the constrains. It can also check if the model satisfies certain desirable properties. These properties can be expressed in the Alloy language and the tool checks if model satisfies the properties. These statements are called *assertions*. Moreover it provides support to debug over-constrained models by locating the parts of the model that cause the inconsistency. This is known as *UnSAT core* functionality.

2.2 Object Constraint Language (OCL)

A constraint is a way to restrict state or behaviour of an entity in a UML model [2]. Since 1997, OCL is a foremost language used for annotation of the UML models with the constraints [3]. OCL is based on first-order logic. Typically, OCL is implicated in representation of functional requirements and operations using class invariants [3, Section 7.3.3], pre and post conditions [3, Section 7.3.4], respectively.

In OCL, if a constraint results true, the system is in valid state and vice versa. Moreover, OCL is a specification language that is strongly typed. All well-formed expressions must conform to the rules of OCL.

2.3 UML2Alloy

The UML2Alloy framework was introduced to automatically transform UML Class Diagrams enriched with OCL constraints into an Alloy model. The generated, Alloy model can be automatically analysed, either from within UML2Alloy or using the Alloy Analyzer. The tool supports the subset of UML and OCL that is directly expressible in Alloy. Moreover in order to bridge other differences a UML profile for Alloy was developed that is used by the tool. In particular there are a few similarities between UML Class Diagrams and Alloy from a semantic point of view such as both Alloy and UML models can be interpreted by sets of tuples [4], [25]. Alloy is based on first-order logic and is well suited for expressing constraints on Object-Oriented models. Similarly, OCL has extensive constructs for expressing constraints as first-order logic formulas. In spite of such similarities, the UML and the Alloy have some fundamental differences [26]. For example, Alloy makes no distinction between sets, scalars and relations, while the UML distinguishes between the three. To bridge some of those semantic differences between UML and Alloy model elements a UML profile for Alloy has been developed [26].

3. Problem Description

The framework used for NL to Alloy translation involves deep syntactic and semantic analysis of NL constraints. In NL2Alloy translation, a chain of transformations is involved such as NL to SBVR, SBVR/UML to OCL and OCL/UML to Alloy. As Alloy will be generated on the basis of extracted semantics of NL constraints, even a minor error or mistake generated at the earlier phases of translation will propagate in rest of the phases and will ultimately result in wrong OCL and wrong Alloy.

In NL to SBVR translation, we have used the Stanford parser to perform syntactic analysis by producing a parse tree and a set of (typed) dependencies. Experiments [7] manifest that the Stanford parser is 84.1% accurate. However, during translation of NL constraints to SBVR, it was found that the Stanford parser is unable to deal with attachment ambiguity as it generates wrong (typed) dependencies. This type of ambiguity is categorized as the prepositional phrase (PP) attachment ambiguity. In PP attachment, the actual problem is typical pattern of a sentence i.e. (VP (NP PP)). The NL sentences with such patterns can be parsed by the human beings but for the automatic parsers it is a difficult task to perform. Following are examples of such cases where the same pattern is interpreted as (VP (NP PP)) in (1) and as (VP (NP) (PP)) in (2) by the Stanford parser.

(1) *A directory is assigned to all files with an extension.*

(2) *A directory object is assigned to all files with a directory.*

Figure 1 highlights the case that is misanalysed by the Stanford parser in example (2).

English: A directory is assigned to all files with an extension.

Typed Dependency (Collapsed):

```
det(directory-2, A-1)
nsubjpass(assigned-4, directory-2)
auxpass(assigned-4, is-3)
root(ROOT-0, assigned-4)
det(files-7, all-6)
prep_to(assigned-4, files-7)
det(extension-10, an-9)
prep_with(files-7, extension-10)
```

English: A directory object is assigned to all files with a directory.

Typed Dependency (Collapsed):

```
det(object-3, A-1)
nn(object-3, directory-2)
nsubjpass(assigned-5, object-3)
auxpass(assigned-5, is-4)
root(ROOT-0, assigned-5)
det(files-8, all-7)
prep_to(assigned-5, files-8)
det(directory-11,a-10)
prep_with(files-8, directory-11)
```

Figure 1.Typed dependencies generated by the Stanford Parser

Figure 1 highlights that the typed dependencies generated by the Stanford parser are correct for example (1) such as `prep_with(files-8, extension-10)` but wrong for example (2) such as `prep_with(files-8, Directory-10)`. However, the correct typed dependency for the example (2) should be `prep_with(object-3, Directory-10)` to represent the actual meanings of the example i.e. a Directory Object with Directory is assigned to all the files. This problem becomes more critical when we map these (typed) dependencies to SBVR vocabulary and OCL. Wrong dependencies generated by the Stanford parser will result in wrong SBVR and wrong OCL. Similarly, the wrong OCL will be mapped to wrong Alloy.

4. NL2Alloy: Sketch of the Solution

This section elaborates the architecture of the solution to generate Alloy from NL constraints. The proposed architecture of the solution is an integration of our existing tools and newly developed modules. The newly developed modules integrate our existing tools. Our existing tools are NL2SBVR, SBVR2OCL and UML/OCL to Alloy. There was need to integrate all these tools and optimize the output of one tool so that next tool in the chain may process text effectively. Additionally, a semantic analyzer was developed to deal with attachment ambiguity of NL constraints discussed in section 3. The NL2Alloy architecture is shown in Figure 2.

To translate NL constraints to Alloy, a set of two inputs are required such as a piece of English text (NL constraint) and domain of English text (a UML class model). Both inputs are processed by our tool and SBVR rule representation is generated. Such SBVR rule representation helps the user to double-check the correctness of semantic analysis of input English. Such measures help in restricting the complexities associated to the inherent ambiguities of a natural language. This is the only semi-automated part of our approach. The remaining steps of the transformation are fully automated. SBVR is transformed into OCL and then into the Alloys module. These steps are discussed in the following chapters. It is possible to produce the UML diagram via one of the many Class diagram extraction tools. However, this step, which is a minor extension of work, remains a task for future.

4.1 NL to SBVR Rules Translation

In this phase, a SBVR business rule is generated from NL constraint that is a semantically unambiguous representation. To overcome ambiguity of a natural language, basic natural language processing (NLP) (lexical analysis [27], syntax analysis [35], and semantic analysis [33]) phases are applied to understand the actual meanings of the NL statement and then map NL statement to a SBVR statement. Following sections explains the transformation of NL text to SBVR rules.

4.1.1 Lexical Analysis

First phase in analysis of natural language specification of Alloy text is lexical analysis. In this step, the input

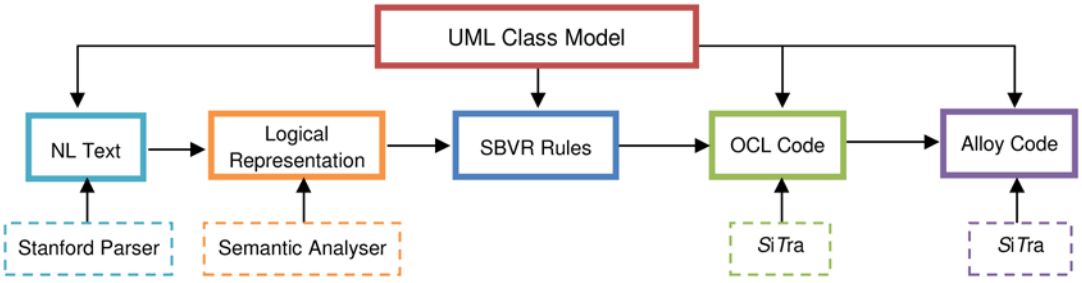

Figure 2. Architecture of NL2Alloy

English text is tokenized and part-of-speech (POS) tagging is performed using the Stanford POS tagger [17]. An example of POS tagging using the Stanford POS tagger is shown in Figure 3.

English: A directory object is assigned to all files with a directory.

Tags:A/DTdirectory/NNobject/NNis/VBZassigned/VBNto/TOall/DTfiles/NNSwith/INa/DTdirectory/NN./.

Figure 3. Part-of-Speech tagged text

After POS tagging, the inflectional endings are removed and the base or dictionary form of a word is extracted, which is known as the lemma. The lemma or base form of a token (all nouns and verbs) is identified by removing various suffixes attached to the nouns and verbs e.g. in Figure 3, verb "*assigned* " is analyzed as "*assign+ed* ". Similarly, the noun "*files*" is analyzed as "*file + s*".

4.1.2 Syntactic Analysis

We have used the Stanford parser to parse the pre-processed English text. The Stanford parser is 84.1% accurate (Cer, 2010). However, the Stanford parser is not capable of voice-classification. Hence, we have developed a small rule-based module classifies the voice in English sentences. We have used the Stanford parser to generate parse tree and (typed) dependencies (Marneffe, 2006) from NL text. To address the identified cases of attachment ambiguity, discusses in the Section 1, we need the context of the NL statement that is a UML class model is the context of the Alloy code. Therefore, we have used the UML class model shown in Figure 6 to correct dependencies. We have used the given relationships in the UML class model such as the associations (directed and un-directed) to deal with the attachment ambiguity. For example in Figure 8, it is shown that there is no direct association in '*Directory*' and '*File*' class, while class 'Directory' is directly associated to class '*Directory Object*'. By using such information, we can correct the dependency as`prep_with(object-3, Directory-11)` instead of the `prep_with(files-8, Directory-11)` identified by the Stanford Parser.

A last phase in syntactic analysis of NL constraints is identification of (active/passive) voice of a sentence. Since, an active voice sentence is treated differently from a passive voice sentence. The Stanford Parser does not classify the voice of English sentences. Various grammatical features manifest passive-voice representation such as the use of past participle tense with main verbs can be used for the identification of a passive-voice sentence. Similarly, the use of 'by' preposition in the object part is also another sign of a passive-voice sentence. However, the use of by is optional in passive-voice sentences.

4.1.3 Semantic Analysis

In semantic analysis phase, we aim to understand the exact meanings of the input English text; to identify the relationships in various chunks and generate a logical

English: A directory object is assigned to all files with a directory.

Parse Tree:

```
(ROOT
(S(NP (DT A) (NN directory) (NN object))
(VP (VBZ is)
(VP (VBN assigned)
(PP (TO to)
(NP
(NP (DT all) (NNS files))
(PP (IN with)
(NP (DT a) (NN directory)))))))
(. .)))
```

Typed Dependency (Collapsed):

```
det(object-3, A-1)
nn(object-3, directory-2)
nsubjpass(assigned-5, object-3)
auxpass(assigned-5, is-4)
root(ROOT-0, assigned-5)
det(files-8, all-7)
prep_to(assigned-5, files-8)
det(directory-11,a-10)
prep_with(object-3, directory-11)
```

Figure 4. Corrected (typed) dependencies

representation. For semantic analysis English constraints, we have to analyze the text in respect of particular context such as UML class model. Our semantic analyzer performs following three steps to identify relations in various syntactic structures:

a) Shallow Semantic Parsing

In shallow semantic parsing, the semantic or thematic roles are typically assigned to each syntactic structure in English sentence. We use SBVR vocabulary as the target semantic roles due to the fact that the mapping of SBVR vocabulary to OCL is easy and straightforward. We have identified mappings of English text elements to SBVR vocabulary (see Table 1).

English Text elements	SBVR Vocabulary
Common Nouns	Object Type
Proper Nouns	Individual Concept
Generative Noun, Adjective	Characteristic
Action Verbs	Verb Concepts
Subject + verb + Object	Fact Type

Table 1. Mapping class model to English

Following is the procedure used for semantic role labelling of English constraints:

To identify predicates, first of all system identifies the words in the sentence that can be semantic predicates or semantic arguments. In English text, a predicate can be in the form of a simple verb, a phrasal verb or a verbal collocation. Similarly, the predicate arguments can be nouns in subject and object part of a sentence. In English, nouns can have pre-modifiers such as articles (determiners) and can also have post-modifiers such as prepositional phrases, relative (finite and non-finite) clauses, and adjective phrases. Once the predicates are identified, semantic roles are assigned by using the

A_{Object_Type}[directory object] $_{verb\_concept}$[is assigned] to all$_{Object_Type}$[files] with$_{Object_Type}$[directory].

Figure 5. Semantic Roles assigned to input English sentence

mappings given in Table 1. Role classification is performed as the syntactic information (part of speech and syntactic dependencies). The output of this phase is shown in Figure 5.

b) Deep Semantic Analysis

The computational semantics aim at grasping the entire meanings of a natural language sentence, rather than focusing on text portions only. For computational semantics, we need to analyze the deep semantics of the input English text. The deep semantic analysis involves generation of a fine-grained semantic representation from the input text. Various aspects are involved in deep semantics analysis. However, we are interested in quantification resolution (see Figure 6) and quantifier scope resolution:

In English constraints, the quantifiers are most commonly used. We not only cover all two traditional types (Universal and Existential) of quantifications in FOL but also we have used two other types: Uniqueness and Solution quantification.

Besides, the quantification resolution, we also need to resolve the scope of quantifiers in input English text. Moreover, the multiplicity given in the target UML class model also helps in identifying a particular type of quantification. For example, in Figure 7, the multiplicity '0…1' specifies that customer can get at most one credit card. This will be equal to At-most *n* quantification in SBVR.

$_{Univeral\_Quantification}$[A]$_{Object\_Type}$[directory object]$_{verb\_concept}$[is assigned] to$_{Univeral_Quantification}$[all]$_{Object\_Type}$[files] with$_{Object_Type}$[directory].

Figure 6. Semantic roles assigned to Input English Sentence

Finally, a semantic interpretation is generated that is mapped to SBVR and OCL in later stages. A simple interpreter was written that uses the extracted semantic information and assigns an interpretation to a piece of text by placing its contents in a pattern known independently of the text. Finally,the logical representation is mapped to SBVR rules. Details of the mapping are given in [14].Figure 7 shows an example of the semantic interpretation and a SBVR representation, we have used in the NL to OCL approach:

4.2 SBVR to OCL Transformation

After generation of SBVR rules, BVR is mapped to OCL constraints by using model transformation technology. In SBVR2OCL transformation, *SiTra* transformation engine is used. A set of model transformation rules were used to transform SBVR rules to OCL constraints [9]. For model transformation of NL to OCL, we need following two things to generate OCL constraints:

A. Select the appropriate OCL template (such as invariant, pre/post conditions, collections, etc.)

B. Use set of mappings that can map source elements of logical form to the equivalent elements in used OCL templates.

A set of OCL templates were designed to generate common OCL expressions such as OCL invariant, OCL pre-condition, and OCL post-condition. User has to select one of these three templates manually. Once the user selects one of the constraints, the missed elements in the template are extracted from the logical representation of English constraint. Following is the template for invariant:

In the all above shown templates, elements written in brackets '[ ]' are required. We get these elements from the logical representation of English sentence. Following mappings are used to extract these elements:

- UML-Package is package name of the target UML class model.
- *UML-Class* is name of the class in the target *UML-Class* model and UML-Class should also be an Object Type in the subject part of the English Constraint.
- *Class-Op* is one of the operations of the target class (such as context) in the UML Class model and *Class-Op* should also be the Verb Concept in English constraint.
- *Param* is the list of input parameters of the Class-Op and we get them from the UML class model. These parameters should be of type Characteristics in English constraint.
- *Return-Type* is the return data type of the *Class-Op* and we get them from the UML class model. The return type is the data-type of the used Characteristic in English constraint and this data type is extracted from the UML class model.

```
(assign
  (object_type = (∃ = 1X ~ (Directory_Object ? X)) AND
  (object_type = (∃ = 1Y ~ (Directory? Y)))
        (object_type = (∀Z ~ (Files ? Z))))
```

SBVR: **It is obligatory** that a directory object*is assigned* to **each**file with directory.

Figure 7. Semantic roles assigned to input English sentence

• *Body* can be a single expression or combination of more than one expression. The details of *Body* are given in the next section.

4.3 OCL to Alloy Transformation

UML2Alloy can map an OCL expression to Alloy code by using model transformation that incorporates the mapping rules between OCL and Alloy. Every OCL *invariant* expression maps to an Alloy *fact* statement. OCL Boolean operations, such as conjunction, disjunction and statements (i.e. *and*, *or*, *not*) map to theequivalentAlloy conjunction (*&&*), disjunction (||) and negation (*!*) operators. Most of the OCL operations on collections have a corresponding Alloy expression.More detailed information on the OCL to Alloy transformation can be found in [30].

5. A Running Example

To explain the presented approach, we have applied our approach to the following NL text:

Consider a model of file system in which *every entity is a DirectoryObject. A DirectoryObject could be a File or a Directory. Each Directory may include a number of DirectoryObjects, which are the entries of the directory. If a DirectoryObject has a Directory as its entity, it is its parent. A DirectoryObject is assigned to all Files with Directory. There is a root Directory without any parents. A directory cannot not be a parent of itself.*

The above example was used described in Section 3. Figure 6 UML Class Diagram of a Simple File System Model depicts a UML class diagram of the file system. This UML Class Diagram model was generated manually from the text description, but tools like CM-Builder [22] can be used to automate this task.

5.1 Generating Alloy

In the following we present the transformation of the NL statements to SBVR, then to OCL and finally to Alloy.

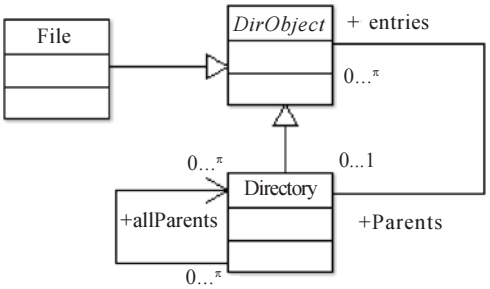

Figure 8.UML Class Diagram of a Simple File System Model

5.2 Model Analysis

The analysis of the model can be carried out from within the NL2OCL, using the UML2Alloy and the Alloy Analyzer APIs

First we try to simulate the model with a *scope*[31] of4. This means that the Alloy Analyzer will attempt to find instances, which conform to the model and its constraints using combinations of up to four *File* and *Directory* instances. After producing a number of acceptable instances, the Alloy Analyzer returned the instance depicted in Figure 7. This was automatically transformed from the Alloy Analyzer analysis notation to UML Object Diagrams by UML2Alloy. The instance shows a directory (*Directory* 0), which is not part of the directories hierarchy. Moreover we see that *Directory*1 is indirectly a parent of itself (through Directory 2).

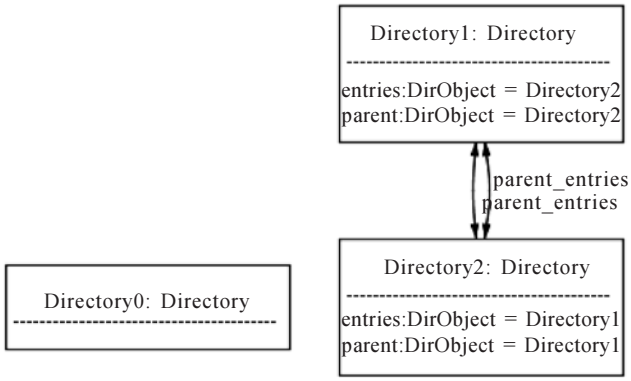

Figure 9. Instance provided by the Alloy Analyzer

Constraint1:

English: There is exactly one directory that has no parent.
SBVR: **It is obligatory** that there *isexactly* one directory that *has* no parent.
OCL:
```
context Directory
invoneRootDirectory : Directory.allInstances() ->
select ( d : Directory | d.parent ->isEmpty() ) -> size() = 1
```
Alloy:
```
fact { Directory_oneRootDirectory[]}
predDirectory_oneRootDirectory[]{# {d:Directory | no d . parent }=1}
```

Constraint2:

English: A directory may not be a parent of itself.
SBVR: **It is possibility** that a directory*may* not *be* a parent of itself.
OCL:
```
context Directory
inv:self.parent -> excludes (self)
```
Alloy:
```
fact{all self: Directory | Directory_notAncestorOfItself[ self ] }
predDirectory_notAncestorOfItself[self: Directory]{
        self !in self.parent }
```

This is clearly an instance that is not desirable. Inspecting our initial model, we can assume that *Constraint* 2 needs to be augmented to express that a directory may not be directly **or indirectly** a parent of itself (i.e. we need to express that the *parent* association is acyclic). In order to do that we would need to express transitive closure using natural language in the NL2Alloy tool. We use an auxiliary self-association on the Directory class as shown in Figure 8. This self-association relates a Directory to all its direct and indirect parents (through the allParents association end). We replaced Constraint1, so that instead of the "*parent*" it uses the "*allParents*" reference. After this change, simulating the system provided only valid instances.

5.3 Results & Discussion

NL2Alloy tool was used to translate 10 examples, similar to the ones presented in this section. All examples contained a UML model and different English description examples to generate Alloy code. The largest English example was composed of 23 words and the smallest sentence was composed of 9 words. We calculated total required (sample) elements in all 10 examples and extracted (correct, incorrect, missing) elements from English description. The Calculated recall, precision and f-values of the solved examples are shown in table 1.

Type	N_{sample}	$N_{correct}$	$N_{incorrect}$	$N_{missing}$	Rec%	Prec%	F-Value
Data	48	43	4	1	89.58	91.48	90.07

Table 2. Evaluation results of NL to Alloy

The average F-value is 90.07 is encouraging for initial experiments. However, it was not possible to compare our results to any other tool as NL-based constraint tool is a novel idea and no comparative tool is available for comparison. However, we can note that other language processing technologies, such as information extraction systems, and machine translation systems, have found commercial applications with precision and recall figure well below this level. Thus, the results of this initial performance evaluation are very encouraging and support both NL2Alloy approach and the potential of this technology in general.

6. Related Work

Many contributions have been made in the field of automated transformations to soft the various processes an phases of software modelling. Recent improvements in model transformation technology, particularly Model Driven Development [7] (MDD), have allowed production of one model from another automatically. There are any instances of this type of transformations example OCL/UML to Alloy [8], SBVR to OCL [9], SBVR to UML [10], UML/OCL to SBVR [11], OCL to B [5], SBVR to SQL [12], etc. Such automated transformations has made easy and simple to reuse the existing information.

7. Conclusion

In this research paper, a framework is presented for dynamic generation of the Alloy code from the NL constraints input by the user. Here, the user is supposed to write simple and grammatically correct English. The designed system can find out the required information to generate a SBVR representation and then transform to a complete SBVR rule, after mapping with the input UML model. The SBVR rules are transformed to OCL expressions and finally translated to Alloy code.

References

[1] Bajwa, I. S., Bordbar, B., Anastasakis, K., Lee, M.G. (2012). On A Chain of Transformations for Generating Alloy from NL Constraints, *IEEE ICDIM*, Macau

[2] OMG. (2007). Unified Modeling Language (UML), OMG Standard, v. 2.3.

[3] OMG. (2006). Object Constraint Language (OCL), OMG Standard, v. 2.0.

[4] Jackson, D. (2006). Software Abstractions: Logic, Language, and Analysis. The MIT Press, London, England.

[5] Kitchin, D. E., McCluskey, T. L., West, Margaret M. B. (2005). vsOCL: comparing specification languages for Planning Domains. *In*: Proceedings of the Fifteenth International *Conference on Automated Planning and Scheduling* (ICAPS)

[6] OMG. (2008). Semantics of Business vocabulary and Rules (SBVR), OMG Standard, v. 1.0.

[7] Michael Azoff. (2008). The Benefits of Model Driven Development: MDD in Modern Web-based Systems, Butler Group, Marc, Available at: http://www.ca.com/~/media/Files/whitepapers/the-benefits-of-model-driven-development.pdf

[8] Anastasakis, K., Bordbar, B., Georg, G., Ray, I. (2007). UML2Alloy: A Challenging Model Transformation, ACM/IEEE 10th International *Conference on Model Driven Engineering Languages and Systems*, LNCS, 735, 436-450

[9] Bajwa, I. S., Lee, M. G. (2011). Transformation Rules for Translating Business Rules to OCL Constraints. *In*: ECMFA 2011- Seventh European *Conference on Modelling Foundations and Applications*, Birmingham, UK, June.

[10] Raj, A., Prabhakar, T. V., Hendryx, S. (2008). Transformation of SBVR business design to UML models, *In*: Proceedings of the 1st India software engineering conference, February 19-22, Hyderabad, India

[11] Cabot, J., et al. (2009). UML/OCL to SBVR Specification: A challenging Transformation, *Journal of Information Systems*.

[12] Moschoyiannis, S., Marinos, A., Krause, P. J. (2010). Generating SQL Queries from SBVR Rules. In RuleML 128-143

[13] Shah, S., Anastasakis, K., Bordbar, B. (2009). From UML to Alloy and Back, 6th Workshop on Model Design, Verification and Validation (MODEVVA 09), *In*: ACM International Conference Proceeding Series; 413, p. 1-10.

[14] Bajwal, S., Behzad, B., Lee, M. (2010). OCL Constraints Generation from Natural Language Specification. EDOC 2010 – 14th *IEEE EDOC Conference*, Vitoria, Brazil, p. 204-213.

[15] Bajwa, I. S., Lee, M. G., Behzad, B. (2011). SBVR Business Rules Generation from Natural Language Specification. AAAI 2011 Spring symposium – AI for Business Agility, San Francisco, USA, p. 2-8.

[16]Richters, M. (2002). A Precise Approach to Validating UML Models and OCL Constraints. Universitaet Bremen. Berlin : Logos Verlag. BISS Monographs, (14).

[17]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*, p. 63-70.

[18]Kleppe, A., Warmer, J., Bast, W. (2003). MDA Explained: The Model Driven Architecture Practice and Promise. The Addison-Wesley Object Technology Series. Addison-Wesley.

[19] Akehurst, D. H., Boardbar, B. et al. (2006). SiTra: Simple Transformations in Java, ACM/IEEE 9th International *Conference on Model Driven Engineering Languages and Systems*, LNCS, 4199, 351-364.

[20] Anastasakis, K. (2009). A Model Driven Approach for the Automated Analysis of UML Class Diagrams, University of Birmingham, PhD Thesis.

[21] Mendel, L. (2007). Modeling By Example. M. Eng. Thesis. Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

[22] 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

[23] NL2Alloy Webpage, Available at: http://www.cs.bham.ac.uk/~bxb/ NL2OCLviaSBVR /NL2Alloy.html

[24] Reddy, Anmandla Sindhura (2011). Building Concept Maps from Unified Medical Language System (UMLS) Dataset, *Journal of E-Technology* , 2 (3) 98-103.

[25] Kaddes, Mourad., Amanton, Laurent., Sadeg, Bruno., Ali, Mouez ., Abdouli, Majed., Bouaziz, Rafik. (2011). RT-ETM: Toward Analysis and Formalizing Transaction and Data Models in Realtime Databases, *International Journal of Web Applications,* 3 (2) 72-79.

[26] Malek, Ben Youssef., Ahmed, Lbath (2011). Towards a method of design of real-time GeoProcessing Applications for Geospatial databases, *Journal of Networking Technology,* 2 (2) 56-62.

[27] Carvalho e Silva, Hedwio ., Cassia C. de Castro, Rita de., Gomes, Marcos Jose Negreiros., Garcia, Anilton Salles (2012). IT Architecture from the Service Continuity Perspective: Application of Well-Founded Ontology in Corporate Environments, *Journal of Information Security Research*, 3 (2) 47-63.