# Semantic Interpretation of Requirements through Cognitive Grammar and Configuration*,**

Matt Selway, Wolfgang Mayer, and Markus Stumptner

Knowledge and Software Engineering Laboratory
University of South Australia, Adelaide, SA 5000, Australia
{⟨first⟩.⟨last⟩}@unisa.edu.au
http://kse.cis.unisa.edu.au/

**Abstract.** Many attempts have been made to apply Natural Language Processing to requirements specifications. However, typical approaches rely on shallow parsing to identify object-oriented elements of the specifications (e.g. classes, attributes, and methods). As a result, the models produced are often incomplete, imprecise, and require manual revision and validation. In contrast, we propose a deep Natural Language Understanding approach to create complete and precise formal models of requirements specifications. We combine three main elements to achieve this: (1) acquisition of lexicon from a user-supplied glossary requiring little specialised prior knowledge; (2) flexible syntactic analysis based purely on word-order; and (3) Knowledge-based Configuration unifies several semantic analysis tasks and allows the handling of ambiguities and errors. Moreover, we provide feedback to the user, allowing the refinement of specifications into a precise and unambiguous form. We demonstrate the benefits of our approach on an example from the PROMISE requirements corpus.

**Keywords:** natural language processing, natural language understanding, semantic interpretation, requirements specifications

## 1 Introduction

Natural Language Processing (NLP) has been used to perform object-oriented analysis of requirements specifications for the last 20-30 years. In many cases, the aim has been to produce *initial* models of classes, attributes, methods, and associations as a starting point for manual refinement into design models. While the approaches vary in the number and types of models produced, they invariably use shallow NLP techniques to create model elements based on the identification of nouns, noun phrases, verbs, adjectives, etc. When compared to models produced by human experts, the results of such techniques typically show

* The final publication is available at link.springer.com
** DOI: 10.1007/978-3-319-13560-1_40

quite low precision and recall (and high over-specification) ranging from less than 40% up to 90% (Harmain and Gaizaukas 2003, Bajwa and Choudary 2012).

While these approaches are considered to save time and effort in producing an initial object-oriented analysis, they are lacking in the context of Model-Driven Engineering (MDE). MDE is an approach to software development centred around the idea of models as core artefacts of the development process and where automated *model transformations* are used to refine higher-level models (e.g. requirements) into lower-level models for design and eventually executable code. As such, high-quality models are required as inputs to MDE processes. However, it is preferable to refine specifications into precise requirements, rather than patch models independently later, as the natural language description is used for communication between stakeholders and is the basis of contractual agreements.

In this paper, we present a Natural Language Understanding approach to translating natural language requirements specifications into formal models. By performing deep, *semantic*, analysis of the requirements and providing feedback to the user, we are able to actively support the refinement of the natural language specifications and produce more complete and precise models. The approach utilises Cognitive Grammar, from the field of Cognitive Linguistics, and Knowledge-based Configuration to handle ambiguities, errors, and inconsistencies in requirements specifications. For example, unresolved relations, indicating tacit or incomplete information in requirements, can be identified; ambiguous quantifiers can be presented to the user for revision, etc. The approach is novel in the following respects: (1) use of flexible syntactic analysis based on word order alone; (2) semantic analysis of requirements using Knowledge-based Configuration as a unifying process for disparate semantic analysis tasks; (3) minimal usage of specialised notations for specifying vocabularies, terms, and definitions, which better supports collaboration and validation of requirements by non-technical stakeholders; and (4) analysis of both functional and non-functional requirements.

The remainder of this paper is organised as follows: Section 2 discuss the limitations of previous approaches to creating formal models from natural language specifications, particularly in the context of MDE; Section 3 describes some of the theories forming the basis of this work; Section 4 presents our approach, which combines these theories; Section 5 demonstrates the approach on examples from a requirements corpus; finally, Section 6 concludes the paper.

## 2 Related Work

In the areas of Software and Requirements Engineering, there have been numerous approaches to creating object-oriented models from natural language requirements. Examples include Harmain and Gaizaukas (2003), Anandha Mala and Uma (2006), Deeptimihanti and Sanyal (2011), Bajwa and Choudary (2012), and Letsholo et al. (2013), in which shallow NLP techniques are used to identify (to varying degrees) classes, attributes, associations, and methods. These approaches use rules to map from identified aspects of the text, e.g. noun and verb phrases, into elements of the model, e.g. classes and methods (respectively). In many

cases the rules are based on the work of Abbot (1983) and Booch (1994), who described manual approaches to identifying object-oriented model elements from text. Where the approaches differ is in the types of models produced—most create only static UML Class Diagrams while some create behavioural models—and in the rules used for filtering the candidate classes, attributes, etc. Moreover, these approaches focus on functional requirements and on revising the models independently of the original specifications.

The approach closest to ours in both motivation and technique is that of CIRCE (Ambriola and Gervasi 1997, 2006), in which natural language is seen as the common thread between the stakeholders (incl. customers and requirements analysts). While text analysis in CIRCE is basically shallow information extraction, the matching rules are augmented with simple semantic tags. In contrast, our semantic analysis goes much deeper. Moreover, in CIRCE, the user must attach semantic tags to terms defined in a glossary and specify *definitions* in a special notation. This requires more technical knowledge on the part of the user and a conceptual leap from initial requirements specification to a partially formalised specification. Such a leap is undesirable as it can lead to inconsistencies and errors; instead we aim to assist the user in making the transition. Finally, the initial "identity" transformation of CIRCE abstracts from the specification, whereas we produce a model that is more closely tied to the textual specification (after which abstractions can be created using MDE-based model transformations).

Another approach that performed deeper semantic analysis was NL-OOPS (Mich 1996). NL-OOPS used the NLP system LOLITA, which processed text into its semantic network (a pre-populated ontology largely derived from WordNet) using a large formal grammar. The object-oriented model was then extracted from the semantic network. While it showed promise, the approach was only ever a prototype, which is no longer available, and empirical results were never reported to the best of our knowledge. Moreover, to our knowledge, no approach has since attempted to utilise similar NLP frameworks to perform such analysis.

Various approaches to semantic interpretation have been proposed in the field of Computational Linguistics, such as Liang et al. (2013). Computational linguistic approaches typically train probabilistic models on annotated corpora, learning the most likely logical forms for sentences. However, to make the problem more tractable and improve the accuracy of the results, training is often performed on a specific domain. Therefore, these approaches are not suitable for application to requirements specifications as they would likely need retraining for each different specification. Although requirements specifications could all be considered part of the "requirements domain", each specification is within a different *application* domain with different vocabulary etc. Moreover, such approaches typically report precision and recall of around 85%, which, as already discussed, is not adequate in the context of MDE.

## 3 Preliminaries

### 3.1 Cognitive Grammar

Cognitive Grammar is a theory of grammar from the field of Cognitive Linguistics that focuses on meaning, is psychologically plausible, and based on conceptualisation (Langacker 2008). Moreover, it provides a unified account of aspects of linguistics that are traditionally considered separate (e.g. syntax, semantics, pragmatics). The basic tenet of Cognitive Grammar is that grammar consists solely of *symbolic structures*; the composition of a *semantic structure* (the meaning of an expression) and a *phonological structure* (its representation) such that one is able to evoke the other (Langacker 2008). These symbolic structures combine to form more complex *symbolic assemblies*. In this view morphology, lexicon, and syntax exist on a continuum that varies along two dimensions: (1) degree of schematicity/specificity, i.e. the level of detail of the information held in a structure; and (2) degree of symbolic complexity, repeated combination of symbolic structures results in assemblies of greater complexity.

Cognitive Grammar expresses several desirable characteristics such as the emergence of traditional lexical categories and grammar rules from the schematisation of expressions occurring in language (Langacker 2008). For example, the noun category is a simple, maximally schematic symbolic structure where the semantic structure is the most general meaning of 'thing' and the phonological structure contains no specific properties. A traditional grammar rule (e.g. that specifying a noun phrase) exists at the maximally schematic and more complex end of the spectrum as it is an assembly of schematic symbolic structures.

The idea that traditional syntax is semantic introduces the possibility of parsing without depending on POS tags. This can be a limiting factor of traditional parsers as errors produced by POS taggers are propagated through further analysis (Naradowsky et al. 2012). In a Cognitive Grammar based approach, traditional syntactic categories can be taken into account, either directly or indirectly, during semantic analysis in parallel with other disambiguation tasks. Therefore, we believe such an approach can improve the precision of the models produced from natural language specifications; although, we simplify the semantic aspects by modelling the semantic structures using the Semantics of Business Vocabulary and Business Rules (OMG 2008) and by utilising Knowledge-based Configuration to search for composite semantic structures. However, we rely on *expectation-based parsing*: a form of syntactic analysis based on word order alone and introduced by Holmqvist (1993) as part of a computational model of Cognitive Grammar. Iteratively performing expectation-based parsing and configuration results in a general framework for the semantic analysis of text.

### 3.2 Semantics of Business Vocabulary and Business Rules

We use the Semantics of Business Vocabulary and Business Rules (SBVR) standard (OMG 2008) as a replacement for the complex semantic structures of Cognitive Grammar. While our NLP framework could use an alternative formal

model for the semantic structures, SBVR is an ideal candidate for our application in the context of MDE as it is intended to bridge the gap between natural language and formal representation within the Model-Driven Architecture (an approach to MDE advocated by the Object Management Group). As such, it partly retains the text structure as an aspect of the model allowing back-references to the source text and supporting traceability between the two. There are two aspects to SBVR: (1) a meta-model for representing vocabularies, facts, and rules in a machine processable format; and (2) a controlled language, SBVR Structured English, for their representation in a form better suited to people.

**The SBVR Meta-model** standardises a set of concepts used to define business vocabularies and rules. The meta-model constitutes the semantics of a formal language that can be interpreted in first-order logic with extensions into modal logic and higher-order logic (using Henkin semantics). This allows for the representation of a wide variety of rules typically stated in requirements specifications and supports various forms of reasoning and consistency checking.

There are two main parts to the meta-model. The first, 'Meanings and Representations', allows for the definitions of vocabularies as sets of interrelated object types[1] (generally common nouns), individual concepts (typically proper nouns), and fact types (usually verbs). Each concept (or meaning in general) can be represented by any number of representations. By separating the representation from the meaning, the meta-model allows for different words (including those in different languages), sounds, and/or images to represent a single concept.

The second aspect of the meta-model is 'Logical Formulations', which provides a conceptualisation of formal logic that is used to represent the semantic structure of rules. As such, it includes concepts for first-order logic operators (conjunction, disjunction, etc.), quantifications (universal, existential, etc.), and modal operators (necessity, obligation, etc.).

**SBVR Structured English (SE)** is a controlled language intended for non-technical business experts to describe and define their business vocabulary (as a glossary), their business rules, and their operations with minimum difficulty. We use SBVR SE as a starting point as it covers a variety of linguistic phenomena while maintaining a higher level of precision. Characteristics of SBVR SE include: (1) verbose expressions with deeply nested clauses; (2) a high degree of polysemy (e.g. 'local area *includes* branch' and 'company *includes* local area' are considered different relations); (3) negation always has local scope; (4) modalities always have wide scope; (5) quantification scope must be resolved; (6) anaphoric reference is limited to within each rule and to typed references (e.g. 'the branch' as opposed to 'it'); (7) ellipsis in coordination of conjunctions and disjunctions.

SBVR SE has reduced but not eliminated ambiguity. For example, 'a' can be used for both universal or existential quantification; 'the' can be used for definite description or anaphoric reference; and, together, (2) and (7) cause repeated verbs to appear equivalent to the user but semantically different to the computer.

[1] sans serif font is used when mentioning concepts from the meta-model

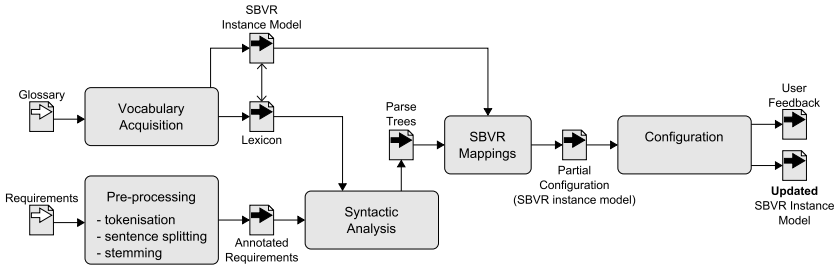


**Fig. 1.** Simple Process Overview

### 3.3 Knowledge-Based Configuration

Configuration is a constraint-based search method historically used to create customised physical products from generic components (Soininen et al. 1998). A configurator requires three types of knowledge: (1) *configuration model knowledge*, i.e. the types of entities, properties, and rules that specify a correct configuration; (2) *configuration solution knowledge*, i.e. a *partial* configuration as input; and (3) *requirements knowledge*, i.e. additional constraints on the configuration that are not strictly part of the configuration model. A configurator can then produce valid configurations or an explanation of why a configuration cannot be found.

Previous applications of configuration to natural language processing, such as Estratat and Henocque (2004) and Kleiner et al. (2009), have been limited to traditional syntactic processing. In this work, we perform configuration on SBVR instance models that form the semantics of the sentences being parsed; which, incorporate the traditional syntactic information as it is represented through the different types of SBVR concepts. In particular, we make use of Generative CSPs, which combine the benefits of other types of Constraint Satisfaction Problems (CSPs) while allowing the configurator to create new instances of model elements in its search for a solution (Stumptner et al. 1998).

## 4 Approach

Our approach follows a common pipeline in language processing including: tokenisation, sentence splitting, morphological analysis (or stemming), syntactic analysis, and semantic analysis. Moreover, our approach acquires a lexicon from a vocabulary or glossary provided by the user. The lexicon is then refined during the later analysis stages. Finally, our approach incorporates an interactive feedback loop with the user to improve the quality of the requirements and resolve errors that cannot be handled automatically. A simple diagram of the pipeline is displayed in Figure 1.

### 4.1 Vocabulary Acquisition

During the initial processing of a specification, glossary entries are differentiated from rules or requirements. The glossary entries, or vocabulary, are then acquired

into a lexicon and an initial SBVR instance model that are used during the parsing process. The glossary format is not strictly specified and can be a simple list of words or include definitions and other information. However, we do prefer the specification of verbs (or relations) using the SBVR fact type forms where the concepts being related are included; e.g. 'local area *includes* branch'.[2] This allows simpler extraction of *grammatical expectations* from the terms, makes the vocabulary more explicit and reduces the need to rely on "general knowledge", which may not include the specialised meanings or usage of the domain.

Initially the vocabulary is acquired using a classifier over the glossary entries. It takes into account various features, such as capitalisation and the existence of other terms in the expression. Once acquired, the vocabulary is maintained as a set of lexical entries in a lexicon defined as a tuple $l = (E, LE, evoke)$; where $E$ is a set of expressions, $LE$ is a set of lexical entries, and $evoke : E \rightarrow 2^{LE} \setminus \emptyset$. Moreover, a lexical entry is a tuple $le = (e, ss, GE)$; such that $e = \langle t_1, \ldots, t_n \rangle$ is a phonological structure, i.e. a (possibly schematic) expression consisting of one or more tokens, $ss$ is its associated semantic structure, and $GE$ is an ordered set of grammatical expectations.

In most cases, the semantic structure of a lexical entry consists of instances of elements from the 'Meanings and Representations' aspect of SBVR; with the exception of keywords, which relate to specific types of logical formulation. Moreover, using SBVR, the grammatical expectations can be inferred from the place-holder structures in the representations of fact types. The grammatical expectations form the basis of the syntactic analysis and are defined as a tuple $ge = (index, type, required, se)$; where $index \in \mathbb{Z}^+$ denotes a unique identifier for the expectation within its set, $type \in \{backward, internal, forward\}$ indicates the direction of search to fill the expectation, $required \in \{0, 1\}$ is a boolean indicating whether or not the expectation must be filled for a parse to be valid, and $se$ is an entity of the semantic structure (linking the elaboration site of the "phonological" pole to the elaboration site of the "semantic" pole).

For example, the following minimal glossary—extracted from (OMG 2008, Annex E)—of two object types and one partitive fact type would produce a lexicon of four lexical entries (one for each glossary entry, plus one for the synonym/ inverse form). In the example, the semantic structures are simply represented by the name of an SBVR type and a numeric identifier of the instance for brevity.

```
branch
  Definition: (informal) organisation unit that has rental responsibility

local area
  Definition: (informal) organisation unit that has area management responsibility

local area includes branch
  Synonymous Form: branch is included in local area
```

[2] The underlining and italics are for example purposes and not required in the glossary.

$$
\begin{aligned}
l = (&E = \{\langle \text{branch}\rangle, \langle \text{local, area}\rangle, \langle \text{includes}\rangle, \langle \text{is, included, in}\rangle\},\\
&LE = \{le_1 = (\langle \text{branch}\rangle, \mathsf{object\_type\#1}, \emptyset), le_2 = (\langle \text{local, area}\rangle, \mathsf{object\_type\#2}, \emptyset),\\
&\quad le_3 = (\langle *, \text{includes}, *\rangle, \mathsf{partitive\_fact\_type\#1}, \{(1, backward, 0, \mathsf{role\#1}),\\
&\qquad (2, forward, 0, \mathsf{role\#2})\}),\\
&\quad le_4 = (\langle *, \text{is, included, in}, *\rangle, \mathsf{partitive\_fact\_type\#1}, \{(1, backward, 0, \mathsf{role\#2}),\\
&\qquad (2, forward, 0, \mathsf{role\#1})\})\},\\
&evoke : E \rightarrow 2^{LE}\\
&\qquad \langle \text{branch}\rangle \mapsto le_1, \langle \text{local, area}\rangle \mapsto le_2, \langle \text{includes}\rangle \mapsto le_3, \langle \text{is, included, in}\rangle \mapsto le_4)
\end{aligned}
$$

### 4.2 Syntactic Analysis

To generate syntactic parse trees, we utilise and extend the expectation-based parsing technique (Holmqvist 1993), which suggests possible composite structures and leaves it to the semantic analysis to determine their efficacy. As such, it is purely syntactic; relying only on word order and *grammatical expectations*.[3] The basic expectation-based parsing algorithm is displayed in Algorithm 1.

Initially, the *suggestion list* ($SL$), an ordered set of suggested parses created by the parsing process, is initialised to empty. The algorithm processes the token stream and on lines 3–5, evokes a lexical entry, trivially creates a new suggested parse, and adds it to the suggestion list. A suggested parse (or *suggestion*) is a tuple $s = (le, C, GE, be, lc, gc)$ such that $le$ is an associated lexical entry, $C$ is a set of catches, $GE$ is a set of grammatical expectations, $be$ is the binding energy, $lc$ is the local coverage, and $gc$ is the global coverage. Binding energy, local coverage, and global coverage are defined by Holmqvist (1993) and used to order the suggestions in $SL$, such that the "best" suggestion is first. Moreover, we incorporate two weight functions: $w_{ctch} : SL \rightarrow \{0..1\}$ and $w_{cght} : SL \rightarrow \{0..1\}$. The calculations of these measures are not shown in the algorithm for brevity; in particular, the weight functions are application specific and can potentially be learnt from training over a corpus. The suggestion list is similar to a *chart* in chart parsing (Jurafsky and Martin 2008) except that the processing is driven by evoked lexical entries rather than a context-free grammar.

Parse trees are formed by suggestions and their *catches*, as determined by the expectations of a suggestion and its relative position to the other suggestions in $SL$. More formally, each catch of a suggestion, $c \in s_1.C$, is a tuple $c = (ge, s_2, cd)$ that associates a grammatical expectation of the catching suggestion to the caught suggestion and a catching distance. The catching distance is a measure of the distance between the relative locations of two suggestions in the token stream (Holmqvist 1993); the calculation of catching distance is not shown for brevity.

[3] While expectations are linked to semantic structures, their semantic content is not considered when suggesting composite structures.

```
   Input  : a token stream TS = ⟨t_1, ..., t_n⟩
   Output : set of best parses in SL
 1 SL ← ∅                                                        // suggestion list
 2 for t_i ∈ TS do
 3 |   le ← evoke(t_i)
 4 |   s_n ← (le, ∅, le.GE, 0, localCoverage(le), globalCoverage(le, i))
 5 |   SL ← SL ∪ {s_n}
 6 |   SN ← {s_n} ∪ CatchBackward(SL, s_n)
 7 |   for s_x ∈ SL do
 8 |   |   for ge ∈ s_x.GE do
 9 |   |   |   for s_i ∈ SN do
10 |   |   |   |   if catchingDistance(s_x, s_i) ≤ catch_max ∧
11 |   |   |   |   (ge.type = forward ∨ ge.type = internal ∧ within(s_i, s_x)) then
12 |   |   |   |   |   s_c ← (s_x.le, s_x.C ∪ {(ge, s_i, catchingDistance(s_x, s_i))},
13 |   |   |   |   |       s_x.GE \ {ge}, bindingEnergy(s_x, s_i),
14 |   |   |   |   |       localCoverage(s_x, s_i), globalCoverage(s_x, s_i))
15 |   |   |   |   |   if valid(s_c) then
16 |   |   |   |   |   |   SL ← SL ∪ {s_c}
17 |   SL ← SL \ {s_x ∈ SL | shouldPrune(s_x)}
18 Function CatchBackward(SL, s)
   |   Input  : a suggestion list SL, and a suggestion s
   |   Output : a set of newly added suggestions SN
19 |   SN ← ∅
20 |   for s_x ∈ SL do
21 |   |   for ge ∈ s.GE where ge.type = backward do
22 |   |   |   if catchingDistance(s, s_x) ≤ catch_max then
23 |   |   |   |   s_c ← ...      // same as lines 12--14, except for s and s_x
24 |   |   |   |   if valid(s_c) then
25 |   |   |   |   |   SL ← SL ∪ {s_c}
26 |   |   |   |   |   SN ← SN ∪ {s_c} ∪ CatchBackward(SL, s_c)
```

**Algorithm 1:** Basic Expectation-based Parsing Algorithm

Furthermore, a suggestion can be simple if it contains no catches, or composite, if it has at least one catch. Simple suggestions correspond to leaf nodes in a parse tree, while composite suggestions correspond to non-leaf nodes.

The catches are created by comparing newly created suggestions with those already in $SL$. This is handled by the algorithm in two parts: (1) backward expectations are tested for the new simple suggestion and any new suggestion created by it recursively (to handle multiple backward expectations), see line 6 and lines 19–25; and (2) internal and rightward expectations of existing suggestions are tested against any newly added simple and composite suggestions, lines 7–14. A new catch, and new composite suggestion, is only created if the catching distance is within the maximum allowable catching distance ($catch_{max}$). In our case, we use very strict parsing and, hence, set a maximum catching distance of zero. When a new composite suggestion is created, the grammatical expectations, catches, etc. are propagated from the catching suggestion (shown on lines 12–14).

Moreover, new suggestions are only added to $SL$ if they are considered *valid*. A suggestion is valid if its caught expectations do not include any free, required expectations and if the catch/caught weights are above a specified threshold. For our application, the thresholds are tuned to severely penalise or outright prune parse trees that would lead to invalid SBVR instance models, helping to control the explosion of possible parses. The parse trees formed by suggestions and their catches differ greatly from the parse trees of traditional methods. A comparison between the two is shown below using a bracketed notation.[4]

```
[Each *[branch]]* is included in *[exactly one *[local area]]
```

```
[S[NP[DET Each] [N branch]] [VP[V is included in] [NP[DET exactly one]
[N local area]]]]
```

Finally, the *pruning strategy* indicates whether or not a suggested parse should be removed from the suggestion list and is simply a boolean function. In our application, the function takes into account the heuristics mentioned above, whether or not a suggestion has successfully passed semantic analysis, and the distance between the suggestion and the current position of the token stream. With an appropriate pruning strategy and definition of a valid suggestion, the process can perform an exhaustive search of the possible parses if necessary.

### 4.3 Semantic Analysis

Since SBVR forms the basis of our semantic structures, the *configuration model knowledge*—the basic entities, relationships, and constraints that determine a valid configuration—for our application comes from the SBVR meta-model. A simple SBVR instance model resulting from the configuration of the sentence 'Each branch is included in exactly one local area' (OMG 2008, Annex E) is shown in Figure 2 and will be used as an example throughout this section.

The *configuration solution knowledge*, i.e. the initial partial configuration, is created from a number of sources. First, the lexicon is transformed into an SBVR instance model, instantiating elements from the 'Meanings and Representations' aspect of the meta-model (shown as white boxes in Figure 2). Next, the suggested parse tree is processed by a set of *mapping rules* to create the SBVR representation of the lexical entries of the parse tree. The mappings create the appropriate SBVR elements, such as those for logical formulations, or simply reference existing vocabulary elements. Below are a few example mapping rules, where $s$ is a variable for a suggestion and $type(x)$ provides the concept from the SBVR meta-model:

$$type(s.le.ss) = \mathsf{indiviudal\_concept} \vee \mathsf{object\_type} \implies s.le.ss \tag{1}$$

$$type(s.le.ss) = \mathsf{fact\_type} \implies \exists x\ \mathsf{atomic\_formulation}(x) \wedge basedOn(s.le.ss) \tag{2}$$

$$\begin{aligned} s.le.ss = \mathsf{universal\_quantification} &\implies \exists x\ \mathsf{universal\_quantification}(x) \\ &\wedge \exists y\ \mathsf{variable}(y) \wedge introduces(x, y) \end{aligned} \tag{3}$$

[4] The asterisks (*) represent the catching expectation, e.g. `each *[branch]` means that the term 'each' catches the term 'branch'; hence, sub-trees are in nested brackets.

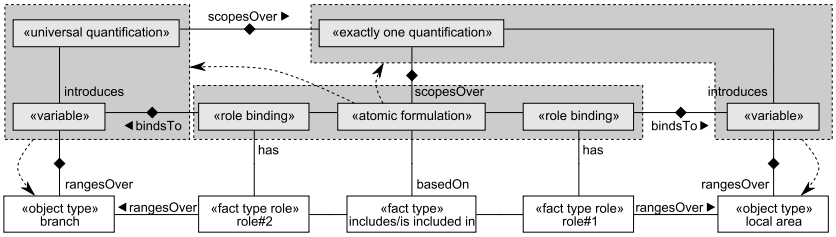


**Fig. 2.** Configuration Example

The light grey boxes in Figure 2 represent those instances that are created due to mapping rules of the kind shown above, while the darker boxes group the elements that are created as part of a single rule. More complex rules support ellipsis in coordination by duplicating aspects of the parse tree. This is necessary, for SBVR, as a natural language rule such as 'EU-Rent operates out of Australia and New Zealand and . . . ' must be represented with distinct elements for 'EU-Rent operates out of Australia', 'EU-Rent operates out of New Zealand', etc.

Finally, the *configuration requirements knowledge* is provided by the catch relationships that form the edges of the parse tree. These relationships are incorporated into the SBVR instance model to encode the word-order constraints of the natural language sentences. These constraints are represented in Figure 2 by dashed, arrowed lines. The catch constraints ensure that relationships are created between the correct components preventing, for example, the variable of the universal quantification from being bound to 'local area', which would change the meaning entirely. In contrast, the constraints of the SBVR meta-model ensure that the model is valid overall: for example, the type that the variable ranges over must be compatible with the type ranged over by the role of the binding that binds to the variable. Therefore, the variable of the universal quantification cannot range over 'branch' while being bound to the role binding of 'role#1'.

A successful parse is determined by using the configurator to find one or more complete configurations. In the example, relationships created by the configurator as part of a solution are represented by lines with diamonds. The configuration process subsumes most of the complex mental processes described by (Langacker 2008) and modelled in (Holmqvist 1993), allowing many semantic analysis tasks (e.g. word-sense disambiguation, co-reference resolution, etc.) to be performed relatively simply by a unified process.

### 4.4 Interactive feedback and corrections

If the configurator is unable to find a valid configuration, or if it finds multiple configurations, feedback is provided to help the user improve the correctness, precision, and quality of the requirements. In the first case, the feedback consists of the explanation produced by the configurator as to why a configuration could not be found—typically the constraint(s) that were violated—and the location in the

document that the error occurred. In the second case, the different interpretations are paraphrased and presented to the user.[5]

Not only can the errors be reported, but the generative aspects of the configurator can be exploited to make suggestions of possible solutions for the error. The partial configuration used as input can be modified by relaxing various constraints and enforcing others, which can then be processed by the configurator to produce the suggestions. For example, if a fact type could not be accommodated due to binding to an incorrect object type, the restriction to the specific object type can be removed, allowing the configurator to search for an alternative object type that can be bound to the fact type successfully. Moreover, by removing the constraint on finding existing vocabulary, the configurator can generate a new lexical entry that is suggested to the user as possibly missing vocabulary to be added.

In most cases the effort required by the user to revise an erroneous sentence will be to select the intended meaning from a set of solutions suggested by the configurator. In other situations, new vocabulary may need to be added by the user or manual revisions made taking the advice of the produced error(s) into account. However, due to the presence of keywords that are part of the underlying semantic model, there will often be enough of a partial parse to enable (partial) inference of the missing information, which can be further elaborated by the user.

## 5 Example

We will demonstrate our approach on a case taken from the PROMISE requirements corpus (Menzies at al. 2012). The corpus contains requirement specifications for 15 projects that include both functional and non-functional requirements and are written in unrestricted natural language. While each requirements specification of the corpus is quite small (the largest consists of 102 requirements and 1,446 words), it is intended for the mining of predictive models of different types of requirements and, hence, is considered to contain requirements that are indicative of real-world projects. The included requirements are primarily of the form 'The System shall ...', as is common in software requirements specifications.

The example requirements, (3) and (4) below, are from a specification for a system that visualises information about events in real-time. It is one of the smaller documents of the corpus, at 33 requirements consisting of 483 words; however, it demonstrates a number of features, such as: modality, tacit information, and recurring events. Since we do not yet include a mechanism to automatically obtain candidate vocabularies from the text, a domain glossary must be manually created for a given specification. This was done iteratively, refining the vocabulary and the requirements over several iterations based on the feedback provided by our prototype tool. After defining appropriate vocabulary, all of the requirements are interpreted by our prototype. The result is a more formally specified textual specification backed by a formal model of the requirements as an SBVR instance model. In the following we describe some of the issues detected and changes made to the example requirements as a result of the analysis performed by the tool.

[5] For brevity, how the paraphrasing is performed is beyond the scope of this paper.

(3) *The system shall refresh the display every 60 seconds.*
(4) *On a 10x10 projection screen, 90% of viewers must be able to read Event / Activity data from a viewing distance of 30.*

First of all, unknown words used in requirements are flagged for the user to add to the glossary. Depending on the number of unknown words in a requirement, the configurator may suggest new lexical entries and automatically attempt to process the sentence using them. The user can then accept the provided interpretation, if it is correct, or add the vocabulary and revise the requirement themselves.

Secondly, the tool identifies tacit information that must be made explicit. For example, while the syntactic analysis of (3) would be correct, the SBVR instance model created from it is incomplete due to unresolved anaphoric references (i.e. 'the system' and 'the display'). While this information is likely obvious to the author of the requirement, it is implicit and leads to incorrect/incomplete models. There are two solutions: (1) allow the configurator to create an appropriate model element as part of an implicit, global context; or (2) explicitly add these entities to the vocabulary. The first solution would allow the anaphoric references with no local resolution to be bound to the global entities, while the second would produce a pair of vocabulary entries that would allow a similar result with the benefit of providing explicit documentation for other stakeholders. For example,[6]

`The System`
  `Definition:` (informal) the system under consideration
`The Display`
  `Definition:` (informal) the graphical display of the system

Lastly, the tool will identify a number of ambiguities, such as those exemplified by (4). For example, without a specific quantification the '90% of viewers' would be interpreted as 'exactly 90%', which is not always the intended meaning ('at least 90%' being the intended meaning in this case); therefore, the tool would suggest that the user revise the sentence to make the quantification explicit. Another ambiguity is introduced by 'Event / Activity data' as it could be interpreted as either 'or' or 'and' (the desired interpretation in this case). While this distinction may not appear significant in the author's mind, it can have a large impact on the reasoning performed over the formal model of the requirement. Finally, assuming a common vocabulary that requires a distance to have a unit of measurement, the 'distance of 30' would provide a semantic error on the grounds of missing its unit of measurement. As a result of the feedback from the tool, the user may produce a revised, more explicit, version of the requirement such as:

(5) *At least 90% of viewers must be able to read Event data and Activity data from a given viewing distance of 30 metres when the data is projected on a given 10 metre × 10 metre projection screen.*

While the revised requirement is slightly more verbose, it is more explicit and precise, and better communicates its intention between systems and stakeholders.

[6] Note that the definitions can be made formal with the inclusion of appropriate vocabulary and rules; however, the example is left informal to keep it small.

## 6 Conclusion and Future Work

In this paper we presented an alternative to shallow NLP techniques for performing semantic interpretation of natural language requirements specifications within the context of MDE. By combining Cognitive Grammar with Knowledge-based Configuration, we perform a deep analysis of the requirements to provide the following advantages: (1) a complete and precise interpretation of both functional and non-functional requirements; (2) terms and definitions in domain specific glossaries are specified with minimal specialised notations; and (3) interactive feedback of ambiguities, errors, and inconsistencies with possible solutions to help the user refine and clarify the requirements. This allows non-technical stakeholders to play a much greater role in the creation and validation of their requirements.

Future work will extend the prototype with the detection of candidate vocabulary entries to support the process of developing domain specific glossaries. Moreover, core vocabularies will be added to provide more consistent interpretation of common considerations (e.g. time) which will better support the reasoning and consistency checking of requirements specifications. Finally, the results of our approach will be quantified through tests on a larger corpus and compared to those of other solutions; of key concern is the determination of effort taken when using our approach compared to that of others.

## References


[1983]Abbot, R.: Program design by informal English descriptions. Comm. ACM 26(11), pp. 882–894 (1983)

[1997]Ambriola, V., Gervasi, V.: Processing natural language requirements. In: Proc. Automated Software Engineering 1997. pp. 36–45. IEEE (1997)

[2006]Ambriola, V., Gervasi, V.: On the systematic analysis of natural language requirements with CIRCE. Automated Software Engineering 13(1), pp. 107–167 (2006)

[2006]Anandha Mala, G., Uma, G.: Automatic construction of object oriented design models [UML diagrams] from natural language requirements specification. In: Yang, Q., Webb, G. (eds.) Proc. PRICAI 2006. LNCS, vol. 4099, pp. 1155–1159. Springer, Heidelberg (2006)

[2012]Bajwa, I.S., Choudhary, M.A.: From natural language software specifications to UML class models. In: Zhang, R., Zhang, J., Zhang, Z., Filipe, J., Cordeiro, J. (eds.) Proc. ICEIS 2011. LNBIP, vol. 102, pp. 224–237. Springer, Heidelberg (2012)

[1994]Booch, G.: Object-Oriented Analysis and Design with Application. Benjamin/Cummings, Redwood City, CA, 2nd edn. (1994)

[2011]Deeptimahanti, D.K., Sanyal, R.: Semi-automatic generation of UML models from natural language requirements. In: Proc. India Software Engineering Conference (ISEC) 2011. pp. 165–174. ACM (2011)

[2004]Estratat, M., Henocque, L.: Parsing languages with a configurator. In: de Mántaras, R.L., Saitta, L. (eds.) Proc. ECAI'2004. vol. 16, pp. 591–595. IOS Press (2004)

[2003]Harmain, H.M., Gaizauskas, R.: CM-Builder: A natural language-based CASE tool for object-oriented analysis. Automated Software Engineering 10(2), pp. 157–181 (2003)

[1993]Holmqvist, K.B.I.: Implementing cognitive semantics: image schemata, valence accommodation and valence suggestion for AI and computational linguistics. Ph.D. thesis, Dept. of Cognitive Science Lund University, Lund, Sweden (1993)
[2008]Jurafsky, D., Martin, J.H.: Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition. Prentice Hall, 2nd edn. (2008)
[2009]Kleiner, M., Albert, P., Bézivin, J.: Parsing SBVR-based controlled languages. In: Schürr, A., Selic, B. (eds.) Proc. MODELS 2009. LNCS, vol. 5795, pp. 122–136. Springer, Heidelberg (2009)
[2008]Langacker, R.W.: Cognitive grammar: a basic introduction. Oxford University Press, Oxford, New York (2008)
[2013]Letsholo, K., Zhao, L., Chioasca, E.V.: TRAM: A tool for transforming textual requirements into analysis models. In: Denney, E., Bultan, T., Zeller, A. (eds.) Proc. 2013 Conference on Automated Software Engineering (ASE). pp. 738–741. IEEE/ACM (2013)
[2013]Liang, P., Jordan, M.I., Klein, D.: Learning dependency-based compositional semantics. Computational Linguistics 39(2), pp. 389–446 (2013)
[2012]Menzies, T., Caglayan, B., He, Z., Kocaguneli, E., Krall, J., Peters, F., Turhan, B.: The PROMISE repository of empirical software engineering data (June 2012), `http://promisedata.googlecode.com`
[1996]Mich, L.: NL-OOPS: from natural language to object oriented requirements using the natural language processing system LOLITA. Natural Language Engineering 2(02), pp. 161–187 (1996)
[2012]Naradowsky, J., Vieira, T., Smith, D.: Grammarless parsing for joint inference. In: Kay, M., Boitet, C. (eds.) Proc. COLING 2012. pp. 1995–2010. The COLING 2012 Organizing Committee (2012)
[2008]Object Management Group (OMG): Semantics of Business Vocabulary and Business Rules (SBVR), v1.0 (2008)
[1998]Soininen, T., Tiihonen, J., Männistö, T., Solunen, R.: Towards a general ontology of configuration. AI EDAM 12(04), pp. 357–372 (1998)
[1998]Stumptner, M., Friedrich, G.E., Haselböck, A.: Generative constraint-based configuration of large technical systems. AI EDAM 12(04), pp. 307–320 (1998)