Motorola NLForSpec: Translating Natural Language Descriptions into Formal Test Case Specifications

Daniel Leitão, Dante Torres and Flávia Barros

Centro de Informática – Universidade Federal de Pernambuco
PO Box 7851 - 50.732-970 Recife(PE), Brazil
E-mail: {dal,dgt,fab}@cin.ufpe.br

Abstract

This paper describes the NLForSpec, a Natural Language (NL) processing tool to translate software test cases descriptions in NL into a formal representation in CSP specification language. NLForSpec is part of a larger project which aims to automate test case generation, selection and evaluation for mobile phone applications. Our tool can be used in the process of update or partially generate requirements documents from test cases (one of the project's main goals). The NLForSpec architecture follows the traditional pipeline NL interpretation approach, counting on a lexicon, a case grammar (to represent semantic information) and a domain ontology. The prototype was tested with a corpus of 100 test cases descriptions, obtaining a performance rate of 91%. This is an original and innovative work.

1 Introduction

The software development process involves three major artifacts: software requirements, code and test cases. Requirements and test cases are often written in some natural language (NL), since this is the stakeholders natural way of communicating. However, NL descriptions may be ambiguous and inconsistent. As a consequence, the interpretation of software requirements and test cases will depend on readers experience. Misinterpretations may introduce mistakes into the code and in the testing phase. One way to minimize these drawbacks is to derive unambiguous formal specifications from NL (requirements and test cases) descriptions, which is not a straightforward task.

Besides ambiguities and imprecision of NL descriptions, we cite yet another problem concerning specifically requirements documents. Regarding software development companies, we identify two common scenarios: (1) well-organized companies maintain requirements, code, and test cases (however, during the project development, the initial requirements documents may become out-of-date); and (2) some companies maintain only code and test cases (defined from non-documented requirements). For companies in the situation (1), it would be helpful to provide for automatic update of requirements documents from more up-to-date test cases (since test cases change more often than requirements). In turn, for companies in situation (2), it would be necessary to (even partially) automatically generate requirements documents from test cases.

We propose here the NLForSpec, a NL Processing (NLP) tool to translate software test cases descriptions in English into a formal representation. This tool addresses the two above-cited problems: (1) by providing a (precise) formal description of test cases, (2) which can be later used as a basis to generate or update requirements documents.

NLForSpec is based on the traditional pipeline NL interpretation architecture. The input NL sentence is parsed and then mapped into case grammar structures (based on thematic roles). Finally, these structures are mapped into representation in CSP (Communicating Sequential Processes) formal language [9]. The implemented prototype was tested with a corpus of 100 test cases descriptions, obtaining a performance rate of 91%. This is an original an innovative work in both NL Processing and Software Engineering areas.

This work is part of the Motorola CIn-BTC research project, a partnership between the Informatics Center (CIn-UFPE) and the Motorola Brazil Test Center. The overall goal of this project is to automate test case generation, selection and evaluation for mobile phone applications. The CSP representations generated by the NLForSpec are used as input by another tool of this major project, responsible for generating requirements descriptions.

In what follows, Sect. 2 presents some state-of-the-art in translation from NL descriptions into formal specifications. Sect. 3 describes the architecture and basic features of the NLForSpec. Sect. 4 presents experiments and results, followed by related work (Sect. 5) and conclusions (Sect. 6).

2 Formal Specifications from NL descriptions

This section briefly describes techniques and systems for translating NL descriptions into formal specifications. In the available literature, we could only identify three systems with similar (although, not exactly the same) goal as our tool: NL-OOPS [14], [6], [10]. This section will comment on the main features of these systems, which will be later compared to the NLForSpec (Sect. 5).These systems were developed within the Symbolic NLP approach, which centers around three major modules: syntactic analyzer, semantic analyzer and discourse/pragmatic analyzer [2].

The syntactic analysis stage aims to determine the input sentence syntactic structure according to a formal grammar. It may also include a pre-processing lexical analysis stage. This module can also be based on a Part-of-Speech (POS) tagger [7], which is commonly used in NLP systems based on the Empirical approach. In contrast to traditional parsers, POS taggers aim to determine the grammatical category of each word in the input sentence, rather than the whole sentence structure. Regarding the above-cited systems, unfortunately, we did not find in the available literature clear information on how they perform the syntactic analysis phase.

The semantic analysis is responsible for creating a representation of the sentence meaning, and it can deploy different formalisms (e.g., First-Order Logics, Semantic Networks, among others). Here, Knowledge Bases (KBs) may be used to explicit represent the objects, concepts, and other entities in a particular domain, as well as the relationships between them. Two of the above-cited systems follow this approach, using Semantic Networks as KBs [14] [6].

The discourse/pragmatics stage covers discourse analysis and intention recognition. Among the reviewed systems, only [10] treats discourse, since it takes as input descriptions of communications protocols, handling this input not as isolated sentences, but rather as paragraphs.

The following section describes the NLForSpec tool in detail, discussing its architecture and main features.

3 NLForSpec: Translating English Descriptions into Test Cases Specifications

As said before, the NLForSpec is part of a larger project, aimed to update requirements documents, among other goals. Our tool is the first step in the translation process from test case descriptions into formal use models, following the Anti-Model-Based Testing Approach [5]. As said, the formal language used in the major project is CSP [9].

Our tool was developed within the NLP symbolic approach, which decomposes the mapping process into a sequence of well-defined tasks (Sect. 2). However, the NL-ForSpec does not count on discourse and pragmatics analyzers, since each sentence in a test case represents an isolated action to be taken. Therefore, there is no need to process discourse fragments, but only isolated sentences.

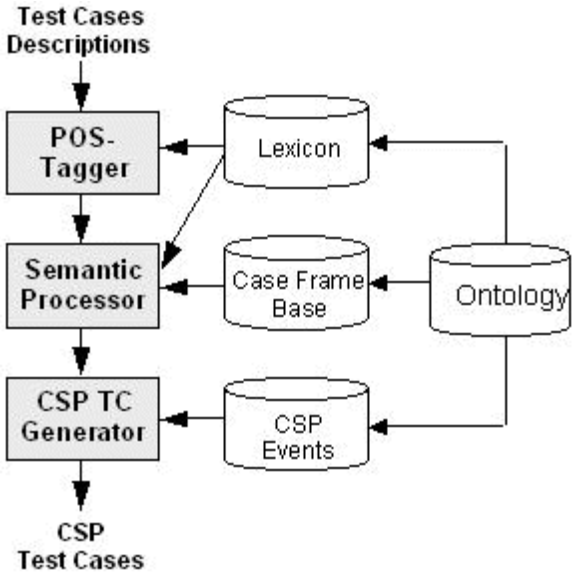


Figure 1. NLForSpec architecture

In what follows, Sect. 3.1 presents the tools overall architecture, and a brief description of the tools knowledge bases and processing modules. Finally the systems process is illustrated by an example.

3.1 Architecture

Figure 1 illustrates the NLForSpec architecture. The rectangles represent the processing modules (Sect. 3.3): POS-Tagger, Semantic Processor and CSP Test Cases Generator. The cylinders represent knowledge bases (Sect. 3.4): Lexicon, Case Frame Base, Ontology and CSP Events.

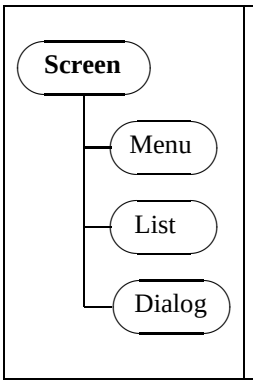
Input test cases are composed by a sequence of sentences of three different types, in the following order: Initial Conditions, Steps and Post Conditions. As said, each sentence in a test case represents an action to be taken. Even the initial and post conditions are here interpreted as actions (e.g., the initial condition “Phone is in idle screen” is translated into “go to idle”). Each sentence is mapped onto one or more CSP events, which compose the CSP process that formally represents the correspondent test case.

3.2 Knowledge Bases

This section presents the tool’s KBs, which are all represented in XML format.

3.2.1 Ontology Base.

The domain entities are represented in the Ontology, which groups them into classes, according to their characteristics. Figure 2 shows a fragment of ontology and its representation in XML format in the domain of mobile phone applications. The ontology represents only specialization relations between classes, in order to ease the addition of new domain entities (since it is just necessary to assign a class to the new entity).




```
<class>
    <name>Screen</name>
    <code>screen</code>
    <subclasses>
        <class>
            <name>Menu</name>
            <code>menu</code>
            <subclasses />
        </class>
        ...
    </subclasses>
</class>
```

Figure 2. Ontology excerpt: graphical and XML representations.

3.2.2 Lexicon.

The Lexicon stores the terms that may appear in the input domain. It is based on the phrasal lexicon approach [4], in which the terms are multi-word phrases. This KB contains three types of terms: (1) Noun, representing a domain entity; (2) Verb, representing an action; and (3) Modifier, representing a modifier that may appear before or after a noun, changing its meaning. Figure 3 presents two lexical terms in the sentence "Select at least 3 messages from inbox folder". As said, nouns represent domain entities. Therefore, each noun entry must contain a tag associating it to one Ontology class.

3.2.3 Case Frame Base.

This KB stores the system's grammar, which is based on the Case Grammar formalism [8]. It comprises a set of case frames with information about the input domain verbs and their thematic roles. These frames represent linguistic semantic knowledge (whereas the ontology represents domain semantic knowledge).

Each case frame corresponds to one verb meaning in the input domain, containing all verbs which share that same

```
<noun>
    <term>inbox folder</term>
    <plural/>
    <class>list</class>
    <model>INBOX_FOLDER</model>
</noun>
```

```
<modifier>
    <term>at least <int/> </term>
    <position>before</position>
    <precedence>last</precedence>
    <model>AT_LEAST.Int</model>
</modifier>
```

Figure 3. A fragment of the Lexicon.

meaning and thematic roles.[1] For instance, the case frame SelectItem (Fig. 4) groups the verbs 'select' and 'choose'.

```
<frame>
    <name>SelectItem</name>
    <verblist>
        <verb>select</verb>
        <verb>choose</verb>
    </verblist>
    <roles>
        <role mandatory="True">agent</role>
        <role mandatory="True">theme</role>
        <role mandatory="False">from-loc</role>
    </roles>
</frame>
```

Figure 4. Case frame base entry.

The Case Frame base also contains a set of restrictions that specifies which ontology classes can be associated to each frame thematic roles. For example, the case frame SelectItem (Fig. 5) is composed by the *theme* role (associated to MenuItem ontology class), and by the *from-loc* role, which is associated to the Menu ontology class.

3.2.4 CSP Events Base.

This KB defines the CSP events (CSP channels and datatypes) used to compose the output CSP representation (Fig. 6). Each channel in this base corresponds to one frame in the Case Frame Base. The datatypes are defined based on the ontology classes, existing one datatype for each class.

[1] Our approach differs from the FrameNet Project [3], in which a case frame contains a set of verbs with the same thematic roles, not necessarily with the same meaning (e.g., 'rent' and 'buy' verbs in the Commerce_buy frame).

```
<restriction name="DTSEL_MENUITEM_MENU">
    <class role="theme">menu_item</class>
    <class role="from-loc">menu</class>
</restriction>
```

Figure 5. Frame restriction example.

```
<channel>
    <name>select</name>
    <casegrammar>SelectItem</casegrammar>
    <datatype>DTSelect</datatype>
</channel>
```

```
<datatype class="list">
    <label>LIST</label>
    <name>List</name>
</datatype>
```

Figure 6. CSP Base Example.

3.3 Processing Modules

The NLForSpec three processing modules are linked together in a pipeline. These modules are detailed in the following sections.

3.3.1 POS-Tagger.

The NLForSpec uses the Stanford POS-tagger [15] to replace the syntactic parsers used in traditional NLP systems. A comparative study among three Web available POS-taggers was performed: Montylingua [11], Stanford POS-tagger and OpenNLP POS-tagger [1]. These taggers were evaluated with 450 test case steps and the Stanford POS-tagger reached the best result.

As said, tagging aims to determine the grammatical category of each word in the input sentence, instead of its whole structure. However, because of lexical ambiguity, the correct determination of each category depends on the category of neighboring words. Therefore, the analysis of each word cannot occur in isolation. The POS-tagger receives a sentence in English, breaks it into tokens, and tags each token with its corresponding grammatical class.

3.3.2 Semantic Processor.

This module maps the tagged sentence into one (or more) case grammar frame in the Case Frame base. Each verb phrase in the sentence is mapped into one case frame. This module analyses the tagged sentence, and applies a set of rules to identify the most suitable case frame to each verb phrase. It then relates the nouns associated to the verb phrase to the corresponding frame's thematic roles. Finally, the Semantic Processor verifies the restrictions that determine whether the nouns related to thematic roles are allowed for the case frame under analysis.

3.3.3 CSP Test Cases Generator.

This module is responsible for mapping each case frame delivered by the previous module into the corresponding CSP event. As said, each event is composed by a channel and its associated datatypes (Fig. 6). This way, the case frame verb is mapped into the event's CSP channel, and the thematic roles are mapped into the corresponding CSP datatypes (that will constitute the channel messages).

3.4 System Execution Flow

Here we present an input-output flow example of all NLForSpec modules. Consider the input sentence "Select at least 3 messages from Inbox". The POS-Tagger receives this sentence as input, and provides a string with its parts-of-speech. The tag *VB* stands for Verb, *DT* for determiner, *NN* for Noun, *IN* for preposition, *JJS* for Adjective, and *CD* for Cardinal.

The Semantic Processor maps the tagged sentence into the corresponding case frame. Initially, this module selects the case frame that contains the verb identified by the POS-Tagger (i.e., 'Select'), and retrieves the ontology classes associated to each noun in the sentence. In this example, the ontology class associated to the noun "message" is *SendableItem*[2], and the "inbox" class is *list*. After that, this module maps the sentence nouns into their respective thematic roles. It also verifies whether there is a restriction associated to the case frame 'Select' containing: (1) the thematic role *theme* associated to the ontology class *SendableItem*, and (2) the role *from-loc* associated to the class *list*. Next, the Semantic Processor identifies the nouns modifiers. In our example, the modifier "at least 3" is associated to the noun "message".

Finally, the CSP Test Case Generator receives the case frame instantiated by the previous module and the modifiers associated to the frame's nouns. This module maps the verb to the corresponding CSP channel, and the nouns and modifiers to the datatypes, delivering the CSP representation of the input sentence.

4 Case Study

This section presents the implemented prototype and experiments results. We worked within the domain of mo-

[2] i.e., an item that can be sent.

1. Input sentence: *Select at least 3 messages from Inbox*
2. Parts-of-speech: *Select/VB at/IN least/JJS 3/CD messages/NN*
3. Case Frame: *Verb: select; Theme: Message; From-loc: Inbox*
4. CSP Event: *select.DTSEL_SENDABLEITEM_LIST.* *(MESSAGE,{AT_LEAST.3}).(INBOX_FOLDER,{})*

Figure 7. NLForSpec Example Flow.

bile phone applications testing. The experiments were performed over a corpus of 100 test cases descriptions from the messaging domain (Sect. 4.2). The obtained results were very encouraging, achieving a performance rate of 91%.

4.1 The Prototype

The tool prototype was implemented in Java, for portability and modularity. The implementation followed the persistent data collections pattern [13]. This way, the Knowledge Bases, represented as XML files, can be easily migrated to another storage form, such as a database.

In order to populate the Knowledge Bases, we randomly selected 450 test cases sentences from the mobile phone messaging application, which were manually analyzed. The Lexicon was filled-in with the nouns, verbs and modifiers identified in these sentences. After that, the Case Frame base was created considering the verbs in the lexicon, and the restrictions for each case frame were defined. It was also necessary to classify the identified nouns according to the Ontology. After that, the CSP Events Base was populated with the channels corresponding to the existing verbs, and the datatypes corresponding to nouns and modifiers.

4.2 Experiments and Results

In order to validate the prototype, we selected another feature from mobile phone application, one not used to populate the KBs. This feature contained 100 test cases.

The experiments results achieved a performance rate of 91%. Regarding the sentences that were not correctly translated into CSP specifications, the following situations were identified in test case pre-conditions:

1. Sentences without a verb, e.g., the sentence “At least 2 messages in Inbox” in test case pre-conditions, instead of “There are at least 2 messages in Inbox”.

2. Ambiguous sentences (which can be interpreted in more than one way), e.g., the sentence “There is one read, unread, and protected message in Message Inbox”. Which messages are in the Inbox: one of each type exclusively or inclusively?

5 Related Work

This section presents the three systems mentioned in Sect. 2, which aim to translate NL descriptions into specification models: [14], [6], and [10].

The NL-OOPS (Natural Language Object Oriented Production System) is a CASE tool that generates OO models from requirements documents in natural language. Its nucleus is the NLP system LOLITA [12], which receives requirements description and generates a semantic network (SemNET) that represents these requirements. A specific algorithm obtains from this network the classes, attributes, associations and operations that will appear on the generated OO model.

Two main drawbacks are identified in the NL-OOPS system: (1) it requires a manual pre-processing of the requirements documents in order to identify ambiguities, inconsistencies and omissions; and (2) for large documents, the corresponding semantic networks are very complex; consequently, the OO model generation task becomes difficult and slow, resulting in inadequate models.

Cyre et al. [6] present a system that generates VHDL models from NL descriptions. This system is composed by four processing modules. The syntactic analyzer parses input sentences based on a grammar of 120 rules. The second module performs the semantic analysis of each sentence and generates its respective conceptual graph (a semantic network). The third module integrates the generated graphs, and the final module generates the VHDL models.

The authors state that all generated models were correct. However, the experiments were based on descriptions containing at most 10 sentences. In our opinion, a description with 10 sentences does not seem to be complex enough to allow a more accurate evaluation of the system.

Finally, we highlight the [10] system, which translates natural language specifications of communication protocols into algebraic specifications. The natural language specifications define action sequences performed by the protocol machine (program). These sentences are translated into an algebraic specification language, called ASL. The published results of this system are good. However, the focus of this work is on communication protocols, which is different from ours, that is focused on test cases descriptions.

6 Conclusion and Future Work

We presented here the NLForSpec, an innovative NLP tool to translate test cases descriptions into formal specifications. This tool integrates a larger system that has as one of its goals the update of requirement documents from more up-to-date test cases. NLForSpec was developed based on the traditional symbolic NLP architecture, consisting of three processing modules (POS-Tagger, Semantic Processor and CSP Test Cases Generator) and four knowledge bases (Lexicon, Case Grammar, Ontology and CSP Events).

NLForSpec achieved a performance rate of 91% in experiments within the domain of Motorola's mobile phone testing. This result indicates that the proposed tool can be successfully integrated into a real industry software development and testing environment. Besides that, the flexibility of our KBs design allows the use of NLForSpec in different domains. For that, it is only necessary to change the KBs content. NLForSpec can also be used to translate NL descriptions into specifications in others formal languages, different than CSP. In this case, the CSP Events Base has to be replaced by other KB that contains information about the new output specification.

As future work, the NLForSpec will be adapted to translate Use Cases (extracted from requirement documents) into CSP specifications. This adaptation represents the first step towards the automatic test cases generation from requirement documents (another goal of our major project). Besides that, we will develop an intelligent user interface to facilitate the update of the NLForSpec KBs. This interface will favor the integration of our tool into the mobile phone testing activities, also allowing experiments with new mobile phone features.

References

[1] The opennlp homepage, 2006. Available at http://opennlp.sourceforge.net/. Accessed on June 03, 2006.

[2] J. Allen. *Natural Language Understanding*. The Benjamin Cummings Publishing Company, Inc, New York, NY, 1995.

[3] C. Baker, C. Fillmore, and J. Lowe. The berkeley framenet project. In *Proceedings of COLING/ACL*, pages 86–90, Montreal, Canada, 1998.

[4] J. Becker. The phrasal lexicon. In *Proceedings of the Conference on Theoretical Issues in Natural Language Processing*, pages 70–77, Cambridge, MA, 1975.

[5] A. Bertolino, A. Polini, P. Inverardi, and H. Muccini. Towards anti-model-based testing. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 124–125, Florence, Italy, 2004.

[6] W. R. Cyre, J. R. Armstrong, M. Manek-Honcharik, and M. Honcharik. Generating vhdl models from natural language descriptions. In *Proceedings of the Euro-VHDL*, pages 474–479, Grenoble, France, 1994.

[7] R. Dale, H. L. Somers, and H. Moisl. *Handbook of Natural Language Processing*. Marcel Dekker, Inc, 2000.

[8] C. J. Fillmore. Frame semantics and the nature of language. In *Annals of the New York Academy of Sciences: Conference on the Origin and Development of Language and Speech*, pages 20–32, 1976.

[9] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[10] Y. Ishihara, H. Seki, and T. A. Kasami. Translation method from natural language specifications into formal specifications using contextual dependencies. In *Proceedings of IEEE International Symposium on Requirements Engineering*, pages 232–239, 1992.

[11] H. Liu. Montylingua: An end-to-end natural language processor with common sense, 2004. Available at: web.media.mit.edu/ hugo/montylingua. Accessed on March 17, 2006.

[12] D. Long and R. Garigliano. *Reasoning by Analogy and Causality: Model and Applications*. Ellis Horwood, Chichester, UK, 1994.

[13] T. Massoni, V. Alves, S. Soares, and P. Borba. Pdc: The persistent data collections pattern. In *First Latin American Conference on Pattern Languages of Programming*, pages 232–239, Rio de Janeiro, Brazil, 2001.

[14] L. Mich. Nl-oops: From natural language to object oriented requirements using the natural language processing system lolita. *Journal of Natural Language engineering*, 2(2):161–187, 1996.

[15] K. Toutanova, D. Klein, C. Manning, and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of HLT-NAACL*, pages 252–259., 2003.