SpecNL: Generating NL Descriptions from Test Case Specifications*

Dante Torres, Daniel Leitão, Flávia Barros

Informatics Center – Federal University of Pernambuco
PO Box 7851 – 50.732-970 Recife, PE

{dgt,dal,fab}@cin.ufpe.br

Abstract. This paper describes the SpecNL, a Natural Language Generation (NLG) tool that generates software test cases descriptions in NL from test case scripts specified in CSP formalism. The main aim is to help test engineers to execute the manual tests. SpecNL is part of a larger project which aims to automate test case generation, selection and evaluation for mobile phone applications. This tool was based on the traditional NLG approach, counting on a lexicon, a case grammar and a domain ontology (all represented in XML). The prototype was implemented in Java for portability and extensibility. This is an original and innovative application of NLG in the Software Engineering area.

1 Introduction

Considering the importance of computer systems for the society and industry, techniques to produce reliable software has become increasingly important. A well-known way to achieve software quality and reliability is the Software Testing.

Software Testing is an important and extremely expensive task of the software development process. Studies suggest that testing often takes approximately 50% of the total software development cost [Boehm 1981]. In order to automate and optimize the testing activity, several tools have been proposed to assist the testing tasks, from test generation to its execution.

Regarding automatic test generation, these tools use some script language to represent the generated test cases. Test cases can be seen as a sequence of actions to be (manually or automatically) performed on a system under test. For test that must be manually executed, this representation may constitute a problem to the test engineers, who will be required to well understand the used script language. In this case, it would be of great help to generate test cases in natural language, in order to facilitate the manual test process.

This paper presents the SpecNL, a tool intended to generate description in natural language (NL) from test scripts. The main aim is to help the test engineers to execute the manual tests. To validate the proposed tool, we developed a prototype that re-

* This work is supported by the Brazil Test Center Research Project, Motorola, Inc. The second author is also supported by CAPES Brazilian research agency.

ceives as input test scripts specified in the CSP [Hoare, 1975] formalism, and delivers as output an English description of test case steps. The prototype was implemented in Java and the tool's knowledge bases are represented in XML, to safeguard portability and extensibility.

This work is part of the CIn-BCT research project, under development in partnership with the CIn-UFPE and Motorola. The overall goal of this project is to automate test case generation, selection and evaluation for mobile phone applications.

Section 2 briefly describes techniques and systems for generating NL descriptions from specifications. Section 3 details the SpecNL, presenting its knowledge bases and processing modules. Section 4 shows the implemented prototype, and Section 5 describes some related works. Finally, Section 6 brings conclusions and future work.

2 From Formal Specifications to NL Descriptions

We can identify two major approaches to NL Generation (NLG) [Reiter & Mellish 1993]: shallow and deep generation. The former approach covers two simple techniques, canned text and the use of templates.

NLG systems based on *canned text* maintain predefined output strings in NL. Given an input data, the system selects the most suitable string as output. This technique does not count on any syntactic or semantics analysis. However, it may perform some surface treatment in the output, such as punctuation, capitalization, etc. This technique is very simple, and limited, and therefore not suitable for more complex applications. We did not find, in the available literature, systems for generating NL from formal specifications based on canned text.

Templates, in turn, are more flexible than canned text, since their predefined output strings may contain slots to be filled in with dynamic information. This technique can be used in multi-sentence generation systems with regular output text structure. As example, we cite the ADL project [ADL, 2001], which generates text from system specifications (see Section 5). Although more flexible, templates are still inadequate for applications which demand a more varied output.

The *deep generation* approach, in turn, deploys more sophisticated techniques capable of providing higher quality NL output. These systems decompose the generation process into a sequence of well-defined and specialized subtasks. Dale and Reiter (2000) identified six basic tasks in a complete NLG process: content determination, document structuring, lexicalisation, referring expression generation, aggregation and linguistic realization. As an example of the use of this approach in the generation of NL from formal specifications, we cite the Review system [Salek et al. 1994] (see Section 5). This system, however, does not implement all the above-cited tasks.

As said, techniques within the shallow generation approach are not able to generate higher quality text. However, they offer some advantages, such as the easiness to develop systems and the low computational cost. On the other hand, systems within the deep generation approach offer a better output quality, but the development process is more complex and the computational cost is higher than in the former approach.

In order to safeguard the advantages of both approaches, whereas overcoming their limitations, some researches propose the construction of *hybrid systems* [Reiter 1995].

In general, these systems use deep generation techniques for high-level operations (e.g., content determination or document structuring) and templates for low-level realization [Buchanan *et. al,* 1992] [Reiter *et al.* 1992].

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

3 SpecNL: From Test Cases Specifications to NL Descriptions

As said, our work is part of the CIn-BCT project, which aims to improve the software test process for mobile phone applications. This major project counts on seven correlated sub-projects which, among other tasks, generate CSP test cases specifications from systems use models. Our tool is a back-end for this system, aiming to generate natural language descriptions from test cases specifications.

The SpecNL was developed within the deep generation approach, counting on three modules (Input Specification Processor, Case Grammar Generator and Surface Realizer) and five knowledge bases. However, it may be classified as a hybrid system (from the NLG viewpoint), since it uses linguistic structures (the Case Grammar) to treat the input test case specification, and uses output templates in the Surface Realizer module (see section 3.3).

In what follows, section 3.1 presents the tool's overall architecture; section 3.2 brings a brief description of the tool's knowledge bases; and section 3.3 shows the processing modules.

3.1 SpecNL Architecture

Figure 1 presents the SpecNL overall architecture. The rectangles represent the three processing modules (sect. 3.3): Input Specification Processor, Case Grammar Generator and Surface Realizer. The cylinders represent knowledge bases (sect. 3.2): Lexicon, Case Frame Base, Ontology and Input Specification base.

As said, although SpecNL is based on the deep generation approach, it does not perform all the traditional tasks. The Sentence Planning is not needed here since the output NL test cases steps can be treated as unrelated sentences (each step consists of an action to be performed).

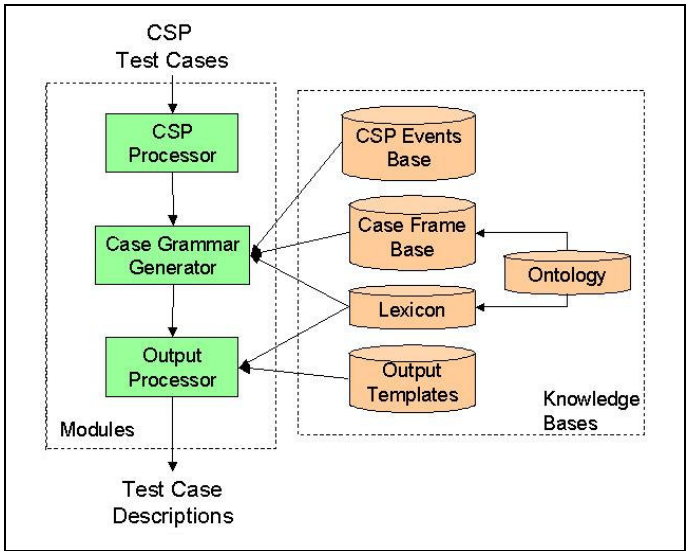

Fig. 1. SpecNL Architecture

The Document Planning and the Content Determination tasks are joined into the Input Specification Processor module, which processes the input specification and identifies the test case sections (in this application, test cases are composed by three sections: initial conditions, steps or post conditions).

In the current version of our tool, the input test cases specifications are represented in the CSP (Communicating Sequential Processes [Hoare 1978]) formal language. As said, this tool is part of a larger system, which generates test cases specifications in CSP. Nevertheless, it is possible to customize our tool to process a different input formal language by adapting the Input Specification Processor and the Input Specification base.

3.2 Knowledge Bases

This section presents the system's Knowledge Bases (KBs). These KBs use the XML format to represent the data.

3.2.1Ontology Base. The application domain entities are represented in the Ontology base, which groups them into classes according to their characteristics. 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). Figure 2 shows a fragment of ontology (and its representation in XML format) in the domain of mobile software user interface.

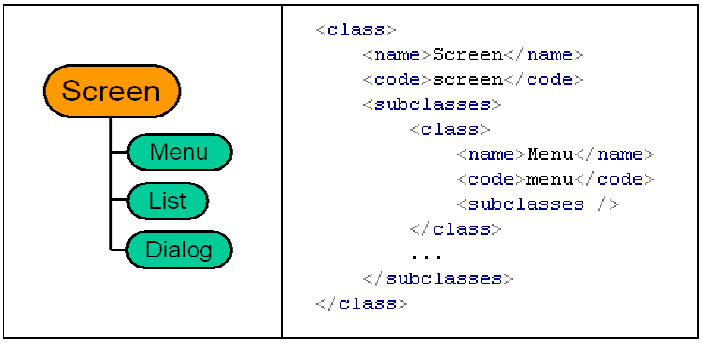

Fig. 2. Ontology excerpt: graphical and XML representations

3.2.2 The Lexicon. The Lexicon stores the terms that may appear in the application domain. It is based on the phrasal lexicon approach [Becker 1975], in which the lexical terms are multi-word phrases. The Lexicon 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. For example, the sentence "Select at least 3 messages from inbox folder" is composed by some of the lexical terms showed in figure 3.

```
<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>
    <numberinflection>plural</numberinflection>
    <article>no</article>
    <model>AT_LEAST.Int</model>
</modifier>
```

Fig. 3. A fragment of the Lexicon

As said, nouns represent domain entities. Therefore, each noun entry contains a tag (*class* tag) associating it to one Ontology class (see figure 3).

3.2.3 Case Frame Base. The system's grammar is based on the Case Grammar formalism [Fillmore 1968], comprising a set of case frames, which contain information about the application domain verbs and its thematic roles. These frames are stored in the Case Frame Base, and they represent linguistic semantic knowledge (whereas the ontology represents domain semantic knowledge).

Each case frame corresponds to one verb meaning in the application domain. As such, each case frame may contain one or more verbs that share the same meaning and thematic roles. Our approach differs from the FrameNet Project [Baker *et al.* 1998], in which each 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). For instance, the case frame SelectItem groups the verbs select and choose (Figure 4).

```
<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>
```

Fig. 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 case frame thematic roles. For example, the case frame SelectItem (Figure 5) is composed by the theme role (associated to MenuItem ontology class) and by the from-loc role, which is associated to the Menu class.

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

Fig. 5. Frame Restrictions example

3.2.4 Input Specification Base. As said, this version of the SpecNL tool works with the CSP formal language. Currently, this base defines the CSP events (CSP channels and datatypes) used to compose the input CSP representation (Figure 6). For each case frame in the Case Frame base, a channel is specified in the Input Specification base. The datatypes are defined based on the ontology classes, there existing one datatype for each class.

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

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

Fig. 6. Input Specification Base example

3.2.5 Output Templates. This base contains the output templates used by the Surface Realizer module to build the output sentence. The templates are specifically designed for a case frame, existing one or more templates for each frame. Figure 7 shows a template example designed for the *SelectItem* case frame.

```
<template info="default">
    <verb> <tense>infinitive</tense> </verb>
    <article agreement="1"/> <role id="1">theme</role>
    of <article agreement="2"/> <role id="2">from-loc</role>
</template>
```

Fig. 7. Output Template example

Besides the fixed strings, an output template may contain slots of the following types: (1) *Article* – indicates that an article may be placed at this position in the template; (2) *Role* – this slot must be filled in with a thematic role (a noun and its modifiers); and (3) *Verb* – represents a slot that must be filled in with a verb in a specific tense and voice. These slots may have additional attributes, such as *id* and *agreement,* to specify agreement between the template elements (e.g., a verb must agree in number with the subject).

3.3 Processing Modules

As seen, the current version of SpecNL uses CSP to formally specify the input test cases. The tool receives as input a CSP test case specification and generates a list of English sentences, each one corresponding to a test case step. The tool's three processing modules are linked together in a pipeline. These modules are detailed in the following sections.

3.3.1 Input Specification Processor. This module receives as input CSP processes representing test cases. Each CSP process is composed by a sequence of events, which are classified as *initial conditions*, *test steps* or *post conditions* (corresponding to the three sections of a test case). The input CSP specification is parsed into a tree structure, which is then searched in order to identify the CSP test case sections (events). In addition, this module also verifies whether the CSP input specification

follows the main project's standard format[1]. As output, this module returns an intermediate representation which associates CSP events and their corresponding sections.

When compared to the tradition NLG systems, we could say that this module performs the Content Determination and the Document Planning tasks.

3.3.2 Case Grammar Generator. This module receives the intermediary representation generated by the previous module, and builds what we call "case grammar structures" based on the Case Frame base and Lexicon. These structures are case frames with their thematic roles filled with terms obtained from the Lexicon. This module also verifies whether the domain entities used in the thematic roles are correct according to the case frame restrictions. This phase can be compared to the tradition NLG Lexicalization task.

3.3.3 Surface Realizer In order to generate the output sentence, the Surface Realizer selects a template in the Output Templates base and fills its slots with the case grammar structure generated by the previous module. Currently, the Surface Realizer is designed to generate one sentence for each case grammar structure. As future work, we are considering to join more than one case grammar structure into one sentence. For that, more complex output templates must be designed, in order to specify the aggregation of case grammar structures.

4 Case Study

This section presents the implemented prototype and some experiments results. This case study was preformed within the domain of mobile phone applications testing. The experiments were based on a corpus of 50 test cases from the messaging domain (sect. 4.2).

The Knowledge Bases were previously populated through an acquisition session performed by another module of the main system. The KBs initial setup was based on 450 test cases sentences randomly selected from the mobile phone messaging application. The Lexicon was filled-in with the nouns, verbs and modifiers identified in these sentences. The nouns were classified according to the domain Ontology. The Case Frame base was filled in with verbs and their associated restrictions. After that, the CSP Events Base was populated with the channels corresponding to the existing verbs, and the datatypes corresponding to nouns and modifiers. Currently, the Lexicon has 120 nouns, 43 modifiers and 27 verbs. The ontology counts on 21 classes.

These KBs can be extended to cover new applications in the messaging domain, as well as an entirely new application domain (e.g., picture application). The KBs update is performed through an intelligent GUI, which verifies the relationships between entities in the KBs (e.g., verbs in the Case Frame base and their corresponding entry in the Lexicon). This GUI is still under development.

---

[1] CSP is a very powerful specification language which allows the design of a vast number of different specification patterns. Therefore, in order to make the overall system feasible, the input CSP specification formats had to be restricted.

4.1 The Prototype

As said, the SpecNL is a back-end of the CIn-BCT project. This tool receives as input CSP specifications and generates the corresponding test descriptions, based on information available from the already populated KBs.

The prototype was implemented in Java, for portability and modularity. The implementation followed the persistent data collections pattern [Massoni *et al.* 2001]. This way, the Knowledge Bases, currently represented as XML files, can be easily migrated to a different storage type/format.

4.2 Experiments and Results

The SpecNL was tested with 50 test cases specifications automatically generated from system use models. The output descriptions were manually analyzed and also used by Motorola test engineers to test messaging mobile phone application. The following problems were identified in the output sentences:

- *Use of articles* – in some cases, the tool uses an inappropriate article before a noun. Here, articles are selected according to a set of rules that consider, among other features, the sentence's corresponding case frame, and the noun's thematic role and ontology class. As future work, we consider to use machine learning in order to select the appropriate article.
- *Misunderstanding of lexical terms* – as said, we are working within the phrasal lexicon approach, allowing multiword terms in the Lexicon. For example, *move to inbox folder* is a lexical term that corresponds to a menu option. However, in some cases, this lexical term may be seen as a sentence, which may generate misleading descriptions. To solve this problem, we are considering using some tag words and quotes to identify lexical terms, as seen in the following sentence: *"Move to inbox folder" option is highlighted.* The tag word used in the sentence was *option*.

The quality of the tool's output is very dependent upon the input. When a correct and complete input test case specification is given, the tool will generate an entire test case description. Otherwise, the output may be incorrect/incomplete due to the lack of information.

5 Related Work

This section presents two tools that aim at mapping specifications into NL descriptions: ADL [ADL, 2001] and Review system [Salek et al., 1994].

The Assertion Definition Language (ADL) project aims to improve the development of tests for system interfaces. ADL is a high-level language used to formally specify the interface's behavior and generate tests and documentation for the interface. ADL specifications consist of First-Order Logics expressions that are composed of three elements: *operators*, translated into verb phrases; *data objects*, translated into

noun phrases; and *functions,* that combine verb phrases (describing the function meaning) and noun phrases (describing how the function parameters participate in the function actions). The documentation of the target system and the documentation of generated test suites are both based on these three elements.

The ADL translator uses a Natural Language Dictionary (NLD), provided by the user, to relate a specification element to its textual representation. The authors state that the NLD is very easy to build and that there is no mechanism to control the introduction of senseless mappings. Clearly, the ADL translator is very simple and not fine-grained. Yet, it does not use the technologies from the NLG field.

Another related work worth to mention is the Review system, which is a subsystem of the Metaview [Findeisen, 1994]. The Review generates NL from SSL (Simple Specification Language) specifications. Besides the SSL specification, the system receives as input a user request, which determines the type of NL report to be generated. The Review system has three major modules. The *Content Selector*, based on the user request, selects the appropriated objects from the SSL specification and orders them according to the user request. The ordered list of objects is sent to the *Sentence Generator* module, which generates initial sentences with the aid of a grammar and a lexicon. The sentences have a fixed syntax according to the input specification object. Finally, the *Global Refiner* module organizes these initial sentences into refined sentences in such a way that they appear as a well-organized text. The consulted bibliography does not detail the knowledge bases representation format and how to update them, which may be a very hard work.

6 Conclusions and Future Work

We presented here the SpecNL, a tool designed to generate test cases descriptions from its specifications. This tool is part of a larger project which aims to automate test case generation, selection and evaluation for mobile phone applications.

SpecNL was developed based on the traditional NLG pipeline architecture, counting on three processing modules and five knowledge bases. A prototype was implemented and tested in the domain of mobile phone message application, showing promising results. We highlight hat this is an original and innovative application of NLG in the Software Engineering area.

Besides the extensions mentioned in section (4.2), we cite as future work: (1) the design of more sophisticated output templates to improve the quality of the generated NL sentences; (2) further tests with a different domain of application; (3) the adaptation of the tool to generate descriptions from Use Case specifications (which is very useful for the software engineers to understand the Use Cases from which the Test Case specifications were automatically generated).

References

[ADL, 2001] ADL project. Assertion Definition Language project website. 2000. Disponível em: http://adl.opengroup.org/index.html. Último acesso: 06 de Março de 2006.

[Baker *et. al*, 1998] Baker, C. F., Fillmore, C. J., and Lowe, J. B. 1998. The Berkeley FrameNet project. In Proceedings of the COLING-ACL, Montreal, Canada.

[Becker, 1975] Becker, J. D. 1975. The phrasal lexicon. In Proceedings of the 1975 Workshop on theoretical Issues in Natural Language Processing (Cambridge, Massachusetts, June 10 - 13, 1975). Theoretical Issues In Natural Language Processing. Association for Computational Linguistics, Morristown, NJ, 60-63.

[Boehm, B. W. 1981] *Software Engineering Economics.* Prentice-Hall, 1981.

[Buchanan *et. al,* 1992] Buchanan B.G., Moore J., Forsyth D., Banks G., and Ohlsson S. Involving patients in health care using medical informatics for explanantion in the clinical setting. In *Proceedings of the 16th Symposium on Computer Applications in Medical Care*, 1992, 500-514. Washington D.C.

[Dale & Reiter 2000] Dale, R., and Reiter, E. Building Natural Language Generation Systems. 1st ed. Cambridge University Press, 2000, ISBN: 0-521-62036-8

[Fillmore, 1968] Fillmore, C. J. 1968. The case for case. In Emmon W. Bach and Robert T. Harms, editors, Universals in linguistic theory. Holt, Rinehart & Winston, New York, pages 1-88.

[Fillmore, 1976] Fillmore, C. J. 1976. 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. volume 280, pages 20-32.

[Findeisen, 1994] Findeisen P. The Metaview system. Technical Report TR96-13, Dept. of Computing Science, University of Alberta, 1994.

[Hoare, 1978] Communicating sequential processes, in Communications of the ACM, vol. 21, no. 8, pp. 666-677, Aug. 1978

[Massoni *et al*., 2001] Massoni, T., Alves, V., Soares, S., Borba, P.: PDC: The persistent data collections pattern. In *First Latin American Conference on Pattern Languages of Programming*, Rio de Janeiro, Brazil, 2001.

[Reiter, 1995] Reiter, E. Nlg vs. Templates. In *Proceedings of the 5th European Workshop on Natural Language Generation (ENLGW-95)*, Leiden, Netherlands, 1995.

[Reiter & Mellish, 1993] Optimizing the costs and benefits of natural language generation . In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-1993)*, pages 1164-1169, 1993.

[Reiter et al., 1992] Reiter R., Mellish C., and Levine J.. Automatic generation of on-line documentation in the IDAS project. In *Proceedings of the Third Conference on Applied Natural Language Processing (ANLP-1992)*, pages 64--71, Trento, Italy, 1992.

[Salek et al., 1994] Salek A. K., Sorenson P. G., Tremblay J. P., and Punshon J. M. The REVIEW system: From formal specifications to natural language. In *Proceedings of the First International Conference on Requirements Engineering*, Colorado Springs, pages 220–229, 1994.