The ucsCNL: A Controlled Natural Language for Use Case Specifications

Flávia A. Barros, Laís Neves
Center of Informatics
Federal University of Pernambuco
Recife (PE), Brazil
Email: {fab, lmn3}@cin.ufpe.br

Érica Hori
MV Sistemas
Apply Solutions
Recife (PE), Brazil
Email: ericahori@gmail.com

Dante Torres
Other Ocean Ltd
Newfoundland, Canada
Email: dantegt@gmail.com

Abstract—In general, test generation tools receive as input either requirements or use case specifications. However, in most companies these specifications are written in free natural language (NL), and the lack of standardization may become a problem for the generation tools and testers. A promising solution is to use a Controlled NL (CNL) to write software specifications. We present here the ucsCNL, a tool for the automatic generation of feature test cases based on use case scenarios written in English. The ucsCNL is already in use, and has achieved satisfactory results.

I. INTRODUCTION

Software Testing has grown in importance in recent years, as a way to increase quality and reliability of the final product. Several tools to automate software testing tasks have been proposed, from test generation to its execution. In particular, automatic test generation has been the focus of several works [1], due to the drawbacks of manual test design, which is time-consuming, and not always systematic and precise.

Usually, test generation tools receive as input either requirements or use case (UC) specifications, from which the test cases are derived. This input should be unambiguous, to preserve the quality of the testing process. However, in most companies requirements and use cases are written using free natural language, and the absence of standardization may become a problem both for the generation tool, and for the professionals who manually execute the test suits.

A promising solution is to use a Controlled Natural Language (CNL) to restrict the creation of requirements and/or use cases. A CNL is a subset of natural language that uses a restricted set of grammar rules and a predefined vocabulary, in order to avoid textual complexity and ambiguity [2].

We present here the ucsCNL, a subset of English language designed for the authoring of UC specifications. The ucsCNL counts on two knowledge Bases: a Lexicon, with the domain-specific vocabulary; and a Grammar, used to restrict the sentence constructions allowed for writing the UC specifications.

The ucsCNL is implemented as a plug-in of TaRGeT [3], a tool for the automatic generation of feature (black box) test cases based on use case scenarios. The ucsCNL is already in use, and has achieved satisfactory results. Initial experiments showed a better performance of testers when manually executing test generated from UCs written using the ucsCNL versus UCs written in free English. This is an original work which combines techniques from Software Engineering, Artificial Intelligence and Natural Language Processing.

Section II discusses research in the CNL field. Section III presents the ucsCNL in detail, section IV shows implementation details, and section V brings conclusions and future work.

II. CONTROLLED NATURAL LANGUAGES AND SOFTWARE SPECIFICATIONS

In order to safeguard the quality of the input specifications in the software development process, some companies use a Controlled Natural Language specially designed to address their particular communicative needs. A CNL [2] is a subset of some NL which uses:

1) a domain-specific vocabulary (Lexicon), in order to avoid synonymy (i.e., two different terms referring to the same entity) and lexical ambiguity (i.e., the same term referring to two or more entities in the application domain); and

2) a restricted set of grammar rules, which can be general (e.g., ‘write short and simple sentences’), or more formal, counting on grammar rules to constrain the accepted syntactic structures (in order to avoid structural ambiguity - a sentence being mapped into two or more different syntactic structures).

It is possible to identify two major groups of CNLs, aimed at different purposes. The most basic goal of a CNL is to define a standard to be followed throughout an organization, in order to provide for unambiguous and clear technical documentation (e.g., ASD-STE100 Simplified Technical English [4]). These CNLs are also known as ‘simplified’, or ‘technical languages’, since their Grammar is basically a set of general writing rules (e.g., ‘write short and grammatically simple sentences’, ‘use active instead of passive voice’ [5]).

The second group of CNLs is at the more formal side. In case a mapping from CNL specifications to a formal representation is desired, the CNL must use a precise syntax and semantics. Based on a formal grammar and a controlled lexicon, it is then possible to define a mapping from CNL sentences into a more formal representation (such as First-Order Logic - FOL) - e.g., PENG [6]. These formal representations...
can be used for further processing, such as model checking [7], automatic test generation [8], and so on.

We highlight here the Attempto project1 and works derived from the Attempto Controlled English (ACE) [9], such as the work of [10], which focuses on Requirement specifications.

Finally, [11] presents a tool for modal-based test generation from ‘natural-language-like’ functional specifications. Requirements are represented using a Template Based Natural Language Specification (TBNLS), where each requirement corresponds to a TBNLS representation. These TBNLSs are mapped into a Formal Requirement Language in which requirements are represented as tuples: (start-condition (if), consequence (then), end-condition (until)). Clearly, this is a too restricted grammar for our purposes.

III. A CONTROLLED NATURAL LANGUAGE FOR USE CASE AUTHORING

This section presents the ucsCNL developed for the authoring of use cases specifications. Our CNL is not merely a technical language, since we are interested in the automation of the whole testing process. This way, we designed a more formal CNL, counting on a Lexicon with pre-defined word types and terms, and a Grammar, used to restrict the sentence constructions allowed for writing the UC specifications.

The ucsCNL version presented in this paper was developed for the mobile phone domain. However, our CNL can be adapted to different application domains. In this work, UC specifications consist of three basic fields:

1) Initial Condition, stating the system’s state before the test takes place (e.g., ‘The phone is in data connection screen’);
2) User Action, stating the test steps to be executed by the tester (e.g., ‘Cancel the operation pressing END key’); and
3) System Response, stating the result of an action or the system state after the test was completed (e.g., ‘The operation was successfully canceled’).

The ucsCNL Grammar provides rules for the three UC fields (section III-B). Section III-A presents the ucsCNL Lexicon.

A. The ucsCNL Lexicon

The Lexicon contains the ucsCNL vocabulary, whose terms are classified into 7 different lexical classes (also known as ‘parts of speech’) [12]: noun (representing the domain entities), verb (representing an action or an event that may occur in the domain), determiner (a noun modifier - detailed below), adjective, adverb, preposition, and conjunction.

Determiners include articles (a, the), quantifiers (every, all, some), and numerals (cardinals and ordinals). Pronouns such as ‘this’, ‘that’ or ‘it’ are not allowed, in order to avoid ambiguity caused by pronominal reference (i.e., anaphora).

Determiners, prepositions and conjunctions are closed word classes, that is, they contain a stable (fixed) set of words which do not depend on the application domain. Terms belonging to these categories are already provided by the ucsCNL initial vocabulary, and cannot be added or erased by the user. Only

1http://attempto.ifi.uzh.ch/site/
the signs can be used. Finally, ‘[]’ indicates that all constructions between the signs must occur together. Some sentence formations require a particular word occurring in a given position (rather than any word from a particular class). Words between quotation marks must be literally interpreted by the rules (e.g., ‘and’ in Table I).

We introduce now the acronyms (tokens) used to represent the Lexicon entries: noun = Noun, verb = VB and other tokens (see below), determiner = DT, adjective = ADJ, adverb = ADV, preposition = PP, and conjunction = CJ. Numerals have their own entries: ordinals = OD and cardinals = CD. The main Verb entries are: VB (infinitive form) and VTB (verb to be).

The verb conjugations are represented by adding suffixes to these acronyms: D - past tense; N - past participle; P - non-3rd person singular present; Z - 3rd person singular present. Finally, the gerund form is represented by G suffix.

A verb phrase (VP) is a group headed by a verb and may consist of a single verb, or auxiliary and main verbs, with optional complements and adjuncts (e.g., ‘The agenda has three entries’). Table II brings some rules for verbs.

2) **User Action Grammar:** User Action (UA) sentences are imperative sentences that convey an action to be executed by the tester (for manual test execution). UA sentences may also state how the action should be executed (e.g., ‘Send a message by pressing the SEND key’). This grammar allows imperative sentences in the negative form, as well as subordinate clauses to an imperative sentence (e.g., ‘After creating a message with 100 characters, go to the drafts folder’).

Note that imperative sentences do not have an explicit subject, since UC actions are commands to the tester. This way, the UA rules do not accept an NP in the beginning of the sentence (see Table III).

3) **Initial Condition and System Response Grammar:** Initial Condition (IC) sentences are statements defining the system state before a test is started. In turn, System Response (SR) sentences convey messages stating how the system is expected to behave after some action has been taken by the tester. We observed that IC and SR sentences have similar grammar structures. This way, the ucsCNL uses the same grammar rules for the parsing of both IC and SR fields (Table IV).

The IC and SR fields convey four different kinds of sentences: active and passive voice sentences, existential sentences, and the so-called ‘verb To Be’ sentences. The ‘verb to be’ sentences are simple formations that denote the state of a domain entity, or better describe it (e.g., ‘The imported media file is a music file’).
IV. IMPLEMENTATION DETAILS

The ucsCNL is implemented as plug-in of a larger system for feature (black box) test cases generation, the TaRGeT (Test and Requirements Generation Tool) [3]. TaRGeT was developed as a software product line, and its main purpose is to automate a systematic approach to generate feature test suites from use case specifications written in the ucsCNL.

TaRGeT receives as input use cases scenarios written in CNL, thus aiding the test designers/engineers, who are more familiar with NL sentences than with formal specification languages. The system model and test suites are automatically derived from these UC scenarios.

A. The ucsCNL Advisor

The ucsCNL plug-in is implemented as the TaRGeT CNL Advisor view at the workbench. This advisor gives support to the process of UC authoring, obeying the ucsCNL Grammar and Lexicon (in order to minimize possible mistakes in the testing phase). In order to conform with TaRGeT’s software product line approach, the ucsCNL Lexicon and Grammar bases are represented as XML files.

The CNL Advisor counts on 3 modules: a pre-processing module, a POS-tagger and a context free parser. Initially, the pre-processing module verifies whether all words in the sentence are included in the Lexicon. When a new word is detected, the user receives an error message. If the word belongs to an open class (noun, verb), the user will able to add it to the Lexicon via a simple user interface. When the word belongs to a closed class, the user receives a message advising him/her to modify the input sentence, or to contact the system’s administrator, who has special priority to edit the Lexicon and the Grammar.

Following, the POS-tagger tags all words in the input sentence with their lexical class. Words that belong to more than one class are tagged with all possibilities, and the parsing process will disambiguate the double categorization based on the available grammar rules. The parser receives a list of words with their lexical class, and builds one tree for every possible syntactic structure. Sentences that could not be correctly parsed are returned to the user. In the case of syntactic error, the Advisor indicates the word categories that were expected instead, and gives examples of correct sentences. The Advisor also allows the user to filter the errors, displaying only the user action, initial condition or system response sentences.

B. Empirical tests

Empirical tests were performed to evaluate the impact and the benefits of the ucsCNL for text standardization. Although this is not the unique aim of our CNL, this preliminary test already indicated the benefits of text standardization. This case study was executed in the context of a mobile device testing company. The aim was to compare the gains of manually executing TC suites created using the ucsCNL versus suites written in free natural language. We considered here: (1) the execution time (the ucsCNL performed better in 35% of the TCs); and (2) number of detected defects/bugs (the ucsCNL detected one extra bug, and avoided a false defect accused by the free LN TCs). The results demonstrate a slight advantage of the use of ucsCNL in relation to free natural language. New tests will be performed in order to confirm this advantage, as well as to identify other gains with the use of the ucsCNL.

V. CONCLUSION AND FUTURE WORK

This paper presented the ucsCNL for the authoring of Use Case specifications, from which test cases can be automatically derived by TaRGeT. The main aim of this work was to aid users (in particular, test designers) who are more familiar with natural language descriptions than with formal specifications.

We are currently developing a mechanism for the automatic mapping from ucsCNL sentences into a formal language representation, to automatically generate test scripts from these formal specifications. Another future work is the inclusion of ontology-based semantic analysis to check the coherence of the UC input specifications according to the specific domain application being tackled.

Acknowledgments: This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES www.ines.org.br), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08.

REFERENCES