TaRGeT: a Model Based Product Line Testing Tool

Felype Ferreira[1], Laís Neves[1], Michelle Silva[1] and Paulo Borba[1]

[1]Informatics Center – Federal University of Pernambuco (UFPE)
Recife – PE – Brazil

{fsf2,lmn3,mcms,phmb}@cin.ufpe.br

Abstract. *To assure software quality and reliability, software testing is a technique that has grown in importance over the years. However, testing can increment development costs in about 50% of the project's total cost. TaRGeT was created as an alternative to reduce these costs by proposing a systematic approach in which test suites can be automatically generated from use case specifications. TaRGeT has been used in different contexts and can be extended by other clients to be customized according to their specific needs.*

1. Context and Motivation

Software testing has been one of the most used techniques for achieving software quality and reliability. Despite the fact that it is an important software development activity, testing software systems can be expensive. Studies suggest that testing often takes approximately 50% of the total software development cost [Ramler and Wolfmaier 2006]. So it is important to figure out a way to automate and improve the productivity of testing activities, ranging from test design to execution.

One of the approaches for automatic test generation is Model-Based Testing (MBT). In order to automatically generate test cases, MBT accepts two main inputs: a formal model of the System Under Test (SUT) and a set of test generation directives, which guide the generation process. The bottleneck of MBT is often the creation of the SUT's formal model, which requires engineers to have formal modeling expertise.

TaRGeT was designed to minimize this bottleneck by automating a systematic approach for dealing with requirements and test artifacts in an integrated way, in which test cases can be automatically generated from use cases scenarios written in natural language, which are used to automatically derive the system model that is transparent to the users. For this, TaRGeT defines a template for the use cases in which it is necessary to provide information about the user action, the system state and the system response for each step of the use case. It is possible to reuse the steps when creating alternative and exception flows.

The tool also provides a GUI for guiding the user to generate test cases. TaRGeT's test case format contains information about the use cases and requirements related to the test, as well as the steps and its expected results and the test initial conditions. The user can describe the system inputs in the text of the test. It is important to highlight that automatic execution is not part of the scope of TaRGeT and the test cases that it generates are designed for manual execution.

TaRGeT's main benefit is a significant increase on productivity, for example, instead of having people to maintain and create a large number of test cases, with TaRGeT it is possible to concentrate efforts on properly describing a much smaller number of use cases and then have the tool generate several test cases from each use case.

2. Main Features

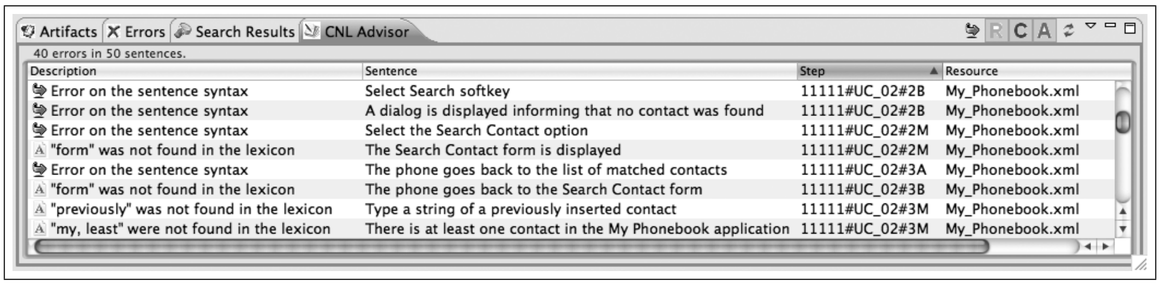
The main functionality of TaRGeT is the automatic generation of test suites from use case scenarios written in natural language. In addition, other tool's important features are listened in the following subsections.

2.1. Controlled Natural Language Verification

The use case specifications used as input in TaRGeT's test generation process are written in natural language (NL), in order to facilitate the tool's usage. However, NL descriptions may be ambiguous and inconsistent. As a consequence, the interpretation of software requirements and test cases will depend on readers experience. TaRGeT's Controlled Natural Language (CNL) was created as an optional feature to improve the process of text verification and to minimize possible mistakes into the code and in the testing phase.

A CNL is a subset of an existing natural language. Basically, a CNL defines some writing rules and a restricted vocabulary in order to avoid authors to introduce ambiguities into their texts and also to define a standard to be followed throughout an organization. TaRGeT's CNL is implemented by the **CNL Advisor** view at the workbench, as showed in Figure 1. The CNL contains a vocabulary, the CNL Lexicon, where the words are stored in different grammatical categories and a CNL Grammar that recognizes the use case documents sentences.

The current implementation only supports documents written in American English. However, this can be extended to other languages because the CNL's parser receives as input XML files containing the lexicon and the grammar rules. So it is only necessary to provide new versions of these files according to a specific language. TaRGeT's CNL recognizes and processes the sentences from the user action, system conditions and system response fields in the use case document template.



40 errors in 50 sentences.

Description	Sentence	Step	Resource
Error on the sentence syntax	Select Search softkey	11111#UC_02#2B	My_Phonebook.xml
Error on the sentence syntax	A dialog is displayed informing that no contact was found	11111#UC_02#2B	My_Phonebook.xml
Error on the sentence syntax	Select the Search Contact option	11111#UC_02#2M	My_Phonebook.xml
"form" was not found in the lexicon	The Search Contact form is displayed	11111#UC_02#2M	My_Phonebook.xml
Error on the sentence syntax	The phone goes back to the list of matched contacts	11111#UC_02#3A	My_Phonebook.xml
"form" was not found in the lexicon	The phone goes back to the Search Contact form	11111#UC_02#3B	My_Phonebook.xml
"previously" was not found in the lexicon	Type a string of a previously inserted contact	11111#UC_02#3M	My_Phonebook.xml
"my, least" were not found in the lexicon	There is at least one contact in the My Phonebook application	11111#UC_02#3M	My_Phonebook.xml

Figure 1. CNL Advisor

There are two categories of errors in TaRGeT's CNL implementation: **lexicon and grammar faults**. The CNL Advisor displays both types of errors and additional information about the sentence, use case step and imported document where the error was found, as showed in Figure 1.

Lexicon faults are displayed when a word present in a sentence from the use case document is not found in the lexicon. To correct this type of error the user can add unknown words in the lexicon or replace it by a synonym present in the lexicon. Words like features and application names and acronyms should be written between quotation marks to not be considered by the parser, like in the following: *Start the "Microsoft Word" application, The "GPS" is turned on.*

Grammar faults will be displayed when all words in a sentence are recognized by the lexicon and the sentences is not grammatically correct. To correct this type of fault the user should try to re-write the sentence in accordance to the CNL grammar.

TaRGeT's CNL helps to avoid spelling errors like *"Scrol down the page"* and grammatical errors as *"Go to the Inbox folder and selects a message"*. In addition, it restricts the vocabulary and avoid ambiguities in the generated tests.

2.2. Test Case Selection

One advantage of the usage of TaRGeT is the possibility to generate a great number of test cases only writing few use cases specifications. However, due to time or budget restrictions, it is often not possible to execute all generated tests. TaRGeT provides some filters that allow the user to select the test cases according to different criteria, such as requirements, use cases, test purpose and similarity of test cases. The generated test suite is the result of these filters combination. If no option is chosen, the tool generates test cases for all possible scenarios. The following items describe the test selection filters:

- **Requirements and Use Cases Filter**: through this filter, the user is able to generate test cases that cover specific requirements and/or use cases.
- **Similarity Filter**: with this filter, the user can select test cases based on similarity [Gadelha et al. 2009]. The tool discards the most similar test cases based on the percentage informed by the user. For instance, it is possible to restrict the test suite by removing the 30% most similar test cases. For this, the user must set the scale to 70%, indicating that the generated test suite will contain the 70% most distinctive test cases.
- **Test Purpose Filter**: this filter allows the user to select test cases using test purposes. A test purpose is an abstract description of a subset of the specification, allowing choosing behaviors to test and, consequently, allowing reducing the specification exploration. A test purpose is composed of a sequence of steps. The operator * can also be used and means any sequence of steps. For example, if the test purpose is defined as [*;4M], it means that the tool shall select all test cases finished on 4M.

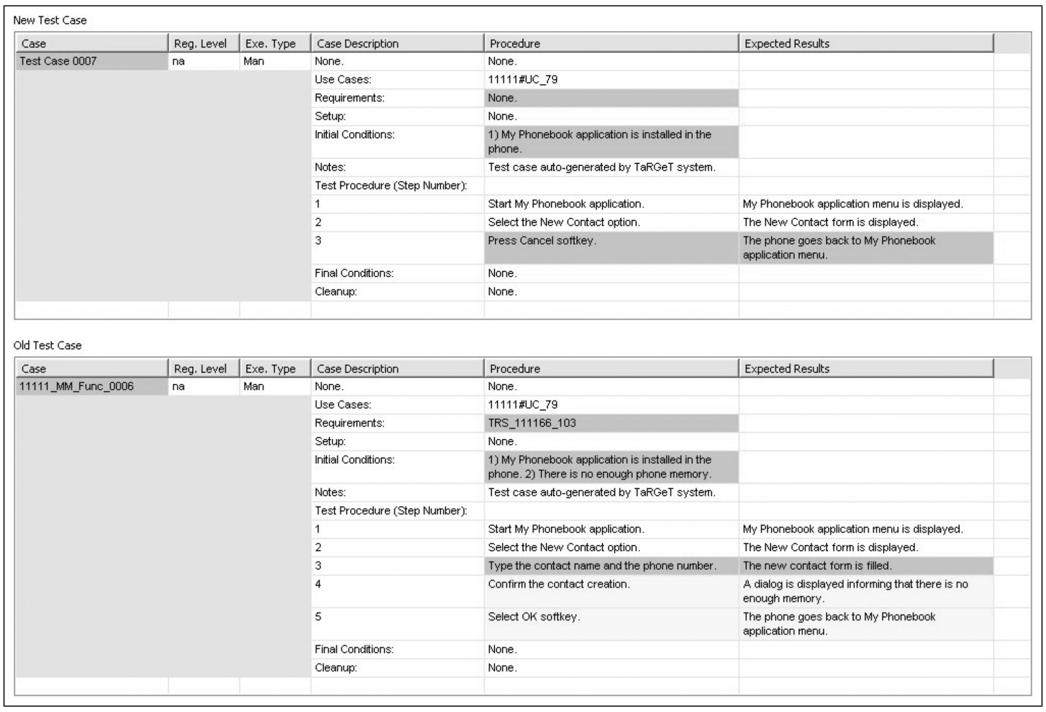
2.3. Test Suite Consistency Management

When requirements evolve and it is necessary to regenerate the tests, information added in the test suite, like test's description and objective, and the selection of tests that have been performed before are lost. In addition, these modifications performed in the use cases can generate new tests with id's that were already in use by other tests in an old suite. TaRGeT Consistency Management (CM) was conceived to solve this problem and to maintain the consistency of different test suites generated over the time. The algorithm has two main steps: comparison and merging.

In the comparison step, the new test suite generated after the modifications made in the use cases is compared with an old one in order to find a relationship between new and old test cases. This functionality is implemented by comparing the steps of the test cases using the Levenshtein's word distance algorithm [Levenshtein 1966].

When the comparison is done, the system returns to the user a list of new and old test cases and the percentage of similarity between then. By selecting two tests, the user can visualize the steps that are different in each one, as show in Figure 2. The system automatically associates tests that have a similarity percentage grater than 95%. However, it is possible to change these associations manually and to configure the percentage of automatic association. Tests that have been added to the suite are marked as new because they don't match any old tests.

The second step occurs when the associations between tests are done and it is time to generate the test suite. In this stage the system performs a merge between corresponding new and old tests, keeping any information that the user has added in the suite and preserving the tests selection. Besides, TaRGeT assigns id's that are not in use to the tests marked as new, assuring the consistency of the resulting test suite.


New Test Case

Case	Reg. Level	Exe. Type	Case Description	Procedure	Expected Results
Test Case 0007	na	Man	None.	None.	
			Use Cases:	11111#UC_79	
			Requirements:	None.	
			Setup:	None.	
			Initial Conditions:	1) My Phonebook application is installed in the phone.	
			Notes:	Test case auto-generated by TaRGeT system.	
			Test Procedure (Step Number):		
			1	Start My Phonebook application.	My Phonebook application menu is displayed.
			2	Select the New Contact option.	The New Contact form is displayed.
			3	Press Cancel softkey.	The phone goes back to My Phonebook application menu.
			Final Conditions:	None.	
			Cleanup:	None.	

Old Test Case

Case	Reg. Level	Exe. Type	Case Description	Procedure	Expected Results
11111_MM_Func_0006	na	Man	None.	None.	
			Use Cases:	11111#UC_79	
			Requirements:	TRS_111166_103	
			Setup:	None.	
			Initial Conditions:	1) My Phonebook application is installed in the phone. 2) There is no enough phone memory.	
			Notes:	Test case auto-generated by TaRGeT system.	
			Test Procedure (Step Number):		
			1	Start My Phonebook application.	My Phonebook application menu is displayed.
			2	Select the New Contact option.	The New Contact form is displayed.
			3	Type the contact name and the phone number.	The new contact form is filled.
			4	Confirm the contact creation.	A dialog is displayed informing that there is no enough memory.
			5	Select OK softkey.	The phone goes back to My Phonebook application menu.
			Final Conditions:	None.	
			Cleanup:	None.	

Figure 2. Test Comparison in Consitency Management

3. Architecture

The TaRGeT implementation is based on the Eclipse RCP architecture [RCP 2010] and it is developed as a software product line [Clements and Northrop 2002] to answer customers particular needs with a significant reduction of the effort and cost required to implement product variations. Figure 3 shows how the dependences between the implemented plug-ins are organized.

The main responsibilities of the application are distributed in four distinct basic plug-ins.

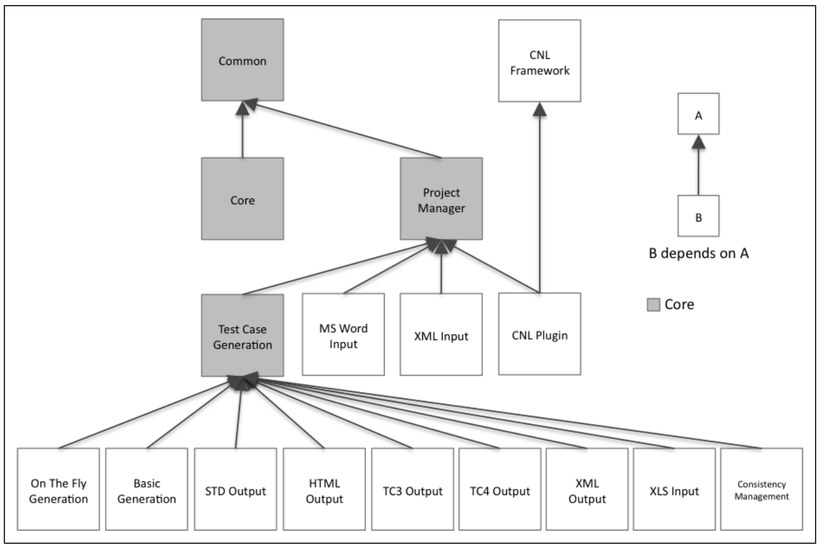


Figure 3. Dependencies between plugins

- TaRGeT Core plug-in: responsible for system starting up and setting up the workspace and perspective of the RCP application.
- Common plug-in: implements basic entities to represent use cases documents and test case suites. Beside that, it contains parsers for different input formats and provides support to new implementations for new input formats.
- Project Manager plug-in: contains operations and exceptions to handling projects, test case generation algorithm, basic GUI components to be extended in the implantation and support for implementing variabilities to make TaRGeT compatible with different formats of input use case documents.
- Test Case Generation plug-in: generates test case suites in different formats and provides support to extend TaRGeT with new implementations for different output formats.

4. Brief Comparison with Other Related Tools

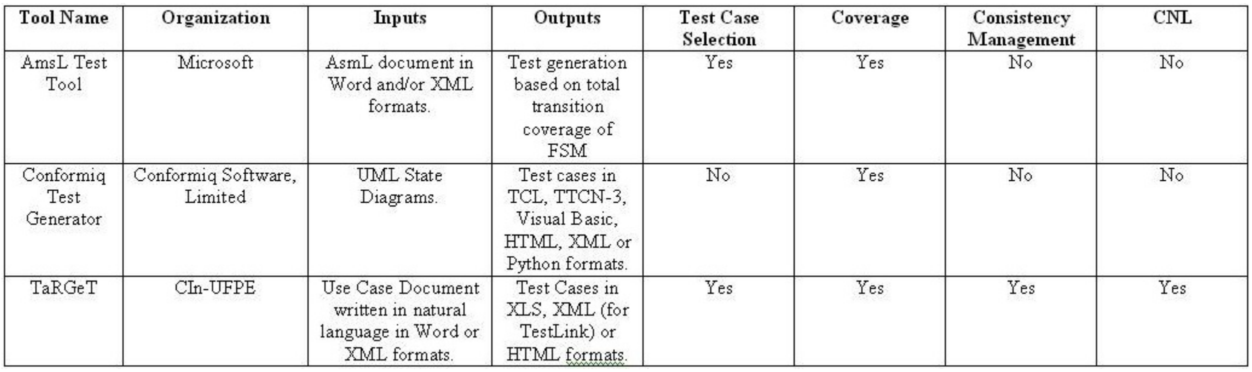
There are many model based testing tools like TaRGeT. In this section we chose two tools to make a brief comparison with TaRGeT: AsmL Test Tool and Conformiq. The Table 1 summarizes the main differences among the tools.

AsmL Test Tool – it was developed by Microsoft and uses AsmL (Abstract State Machine Language) that is an executable modeling language which is fully integrated in the .NET framework and Microsoft development tools [Barnett et al. 2004].

Conformiq [Conformiq 2010] – is a tool for automatically designing and creating test cases. It uses as input system models in Java and UML compatible notation, mathematically calculates sets of test cases and exports the test cases in the format chosen by the user that can be TCL, TTCN-3, Visual Basic, HTML, XML or Python.

5. License

TaRGeT is an open source software whose code and documentation are covered by the MIT license [MIT 2010]. The complete license text and the tool can be found at [TestProductLines 2010].



Tool Name	Organization	Inputs	Outputs	Test Case Selection	Coverage	Consistency Management	CNL
AmsL Test Tool	Microsoft	AsmL document in Word and/or XML formats.	Test generation based on total transition coverage of FSM	Yes	Yes	No	No
Conformiq Test Generator	Conformiq Software, Limited	UML State Diagrams.	Test cases in TCL, TTCN-3, Visual Basic, HTML, XML or Python formats.	No	Yes	No	No
TaRGeT	CIn-UFPE	Use Case Document written in natural language in Word or XML formats.	Test Cases in XLS, XML (for TestLink) or HTML formats.	Yes	Yes	Yes	Yes

Table 1. Comparison among tools.

6. Acknowledgements

We would like to thank National Institute of Science and Technology for Software Engineering (INES) [INES 2010] and Motorola Brazil in a cooperation project for partially supporting this work, CNPq and FACEPE for partially funding it, grants 573964/2008-4 and APQ-1037-1.03/08. In addition, we thank all research team members, especially Sidney Nogueira, Emanuela Gadelha, Dante Torres and Gustavo Cabral, who collaborated with this project.

References

Barnett, M., Grieskamp, W., Nachmanson, L., Schulte, W., Tillmann, N., and Veanes, M. (2004). Towards a tool environment for model-based testing with asml. *Microsoft Research*.

Clements, P. and Northrop, L. (2002). *Software Product Lines: Practices and Patterns*, page 563. Addison-Wesley.

Conformiq (2010). Conformiq test generator. http://www.conformiq.com/products.php.

Gadelha, E., Machado, P., and Neto, F. (2009). On the use of a similarity function for test case selection in the context of model-based testing. *Software Testing, Verification and Reliability Journal*. Wiley.

INES (2010). National institute of science and technology for software engineering. http://www.ines.org.br.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*.

MIT (2010). MIT License. http://www.opensource.org/licenses/mit-license.php.

Ramler, R. and Wolfmaier, K. (2006). Economic perspectives in test automation - balancing automated and manual testing with opportunity cost. *Workshop on Automation of Software Test ICSE*.

RCP (2010). RCP FAQ. http://wiki.eclipse.org/RCP_FAQ.

TestProductLines (2010). Tests generation, selection, prioritization and processing product line. http://twiki.cin.ufpe.br/twiki/bin/view/TestProductLines/TaRGeTProductLine.