LTS-BT: A tool to Generate and Select Functional Test Cases for Embedded Systems

ABSTRACT

Automation of model-based testing for embedded systems is discussed. The focus is on feature and feature interruption test case generation and selection from behavioral specifications. For this, the LTS-BT tool is presented. The tool has been designed to suit embedded systems by focusing on selected notations for behavior specification and tailored techniques for test case generation and selection. This is motivated by the peculiarities of these systems that challenge cost-effective testing.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Experimentation, Verification

Keywords

Model-based Testing, Generation, Selection

1. INTRODUCTION

Automation of software testing activities can promote reliability and productivity in testing. Recently, a number of tools have been developed for test case generation, focusing on different specification notations such as UML [8]. On one hand, the use of testing tools in industry is highly dependent on the relative costs of incorporating them in specific development and testing processes. In this sense, tailored solutions seems to be more appropriate as well as creating an opportunity to bring the gap between theory and practice. On the other hand, certain kinds of testing such as interruption testing is still not properly covered by current tools.

A feature is a clustering of individual requirements that describe a cohesive, identifiable unit of functionality [12]. Features are combined into applications as the basis of a given device. In this paper, we denote by feature interruptions those scenarios where features can interrupt other features [12]. Such applications operate under severe constraints and they are also driven by user inputs and network events that may trigger concurrent execution. Conventional techniques and tools for desktop-like distributed applications are not directly applicable to the embedded systems ones: goals and challenges are clearly different and the second present new ones. For instance, interruptions can occur at any point of execution. Such interruptions are rarely specified and also they are difficult to anticipate. Moreover, the costs of manual test execution are high, even for reasonable size test suites, demanding effective test case selection techniques that can effectively cope with redundancy.

Generally, embedded systems are developed in an incremental way, i.e., initially one feature is developed and after/in parallel other features are developed and integrated with the former. Embedded systems, such as mobile phones, are composed of a set of features. Then, when features are put together, perhaps undesirable interrelationships with already integrated parts of the system are introduced [14]. This problem must be taken into consideration during a testing phase.

This paper presents a solution for feature and feature interruption test case generation and selection for embedded applications. We focus on embedded applications, composed of a number of features, that are highly interactive, having their flow of execution guided mostly by external input. For this, the LTS-BT (Labelled Transition System-Based Testing) tool has been developed. In its current version, the tool supports test case generation from an annotated Labelled Transition System (LTS). LTSs are largely used as the semantics formalism of specification notations, making it easier for the tool to interoperate with other tools, specially UML modelling tools. Also, LTSs can be easily created from mind maps of the observable behavior of an application. The annotations are basically concerned with giving semantics to transitions according to the general format of embedded applications. The tool also gets specifications in UML sequence diagrams. Two strategies for test case selection are also proposed. These are motivated by the challenges to be faced in both, feature and feature interruption testing. To the best of our knowledge, there are no current tools that directly support feature interruption testing or can be used in a cost-effective way with such purpose. The tool also includes a novel selection strategy to cope with redundancy that is staple for manual testing. This paper is structured as follows. Section 2 introduces the concepts of feature and feature interruption testing. Section 3 presents

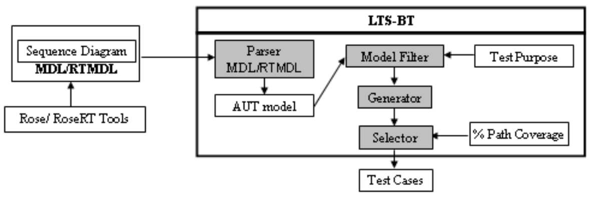


Figure 1: LTS-BT tool architecture

the LTS-BT tool. Section 4 presents a case study. Related works are presented in Section 5. Finally, Section 6 presents conclusion.

2. FEATURE AND FEATURE INTERRUPTION TESTING

Features are usually developed and tested in isolation [3]. This way, feature testing is very important to reduce the number of defects that escape from one phase to another during the test process. Feature testing, even though a functional testing, has some particularities. In the embedded systems applications area, for example, the majority of test cases are manually executed. This fact increases the probability of human error during test cases execution. Therefore, test cases usually have a very small number of steps. Another characteristic of feature testing is the concept of redundancy, specially when two test cases have the same steps but in a different order, distinguishing them. However, these test cases can be considered redundant because the order of steps is not important, for instance, when filling fields of a form and the checking is performed after the data submitting. Therefore, test case generation and selection need further investigation.

Some embedded systems such as mobile phone applications may be composed of a number of concurrent processes and network distributed services, where interruptions in a flow of execution can occur at any time. The possible number of combinations of allowed interruptions at different points of a flow of execution is huge: there are many scenarios where interruptions can occur. One example of an interruption in this context is when the user is composing a text message and a call arrives in its device, causing an interruption of feature "call" in feature "to send text messages". Considering that any interruption can occur at any time, there are infinite possibilities of occurrences. This makes the specification of each possibility unfeasible by using conventional notations and strategies. As a consequence and also due to the lack of a suitable test model, test case generation and selection is compromised. A definite challenge is to define a cost-effective way of specifying interruptions by composing features and properly select key test cases among the potentially infinite set of possible test cases.

3. LTS-BT TOOL

LTS-BT is a test case generator and selector tool. It takes as input sequence diagrams or LTS behavioural models with annotations and generates test cases. Also, the tool selects test cases according to a test purpose and a coverage percentage criteria.

In summary, the input formats are: (i) **Behavioral model** - it can be Sequence Diagrams Rose(MDL)[1] and Rose-RealTime(RTMDL)[2] formats) or LTS (TGF and AUT formats); (ii)**Test Purpose** - targeting transitions, including the generalizing "*" transition; (iii) **Path Coverage Percentage** - determines the desired test cases coverage degree relative to the model. The test purpose and path coverage percentage is chosen by the tester in the LTS-BT tool's GUI.

The test cases generated by the LTS-BT tool are abstract, i.e., they describe behaviors in terms of input/output interactions between the tester and the System Under Test (SUT), besides initial conditions. As an alternative output, the tool maps the LTS to a table to facilitate the manual execution. The table shows the initial conditions, user actions and expected results. The test case format is an LTS that can be easily translated into a specific language such as TTCN[3].

For feature interruption testing, one behavioural model for each feature to be considered is given as input to the tool. Some of the features can cause interruptions in the flow of the others. Features and Interruptions are specified as in [7] and translated into LTS models. By default, interruptions can happen at any execution point of a given feature. Special treatments such as interruptions not allowed at certain points and alternative behaviour can also be specified. The tool automatically build the behavioural model by composing features according to the specification. From this model, test cases are generated and selection strategies can be applied.

3.1 Architecture

The architectural design can be seen in Figure 1. For each functionality, a module was created. In next Subsections, each module is described.

3.1.1 Parser MDL/RTMDL

The LTS-BT tool works internally with Aldebaran (AUT) format[4]. Then, this module execute the derivation. This is based on the general procedure to generate flow graphs from sequence diagrams presented in [2]. As UML sequence diagrams have information about different objects of an application, and we are interested in functional testing, the messages sent between internal objects are not considered. The behavioral model in the AUT format is automatically derived from sequence diagrams (MDL/RTMDL)[5].

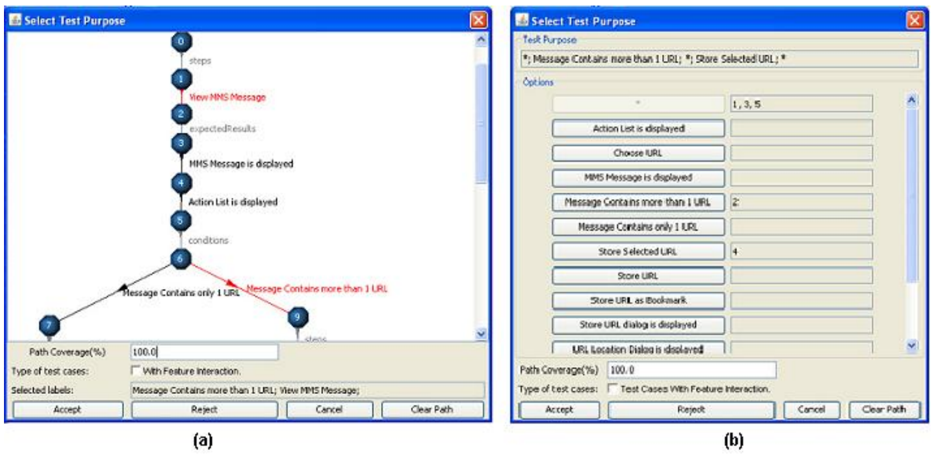
3.1.2 Model Filter

A test purpose targets test case generation at a particular functionality. This strategy has been widely considered in the literature, but the notations for test purpose specification are little used in practice due to the difficulty of applying them. In this sense, LTS-BT uses a simple notation based in [10].

In the LTS-BT tool, for each LTS behavioral model,the set of all possible transitions is displayed in the GUI, i.e., the conditions, steps and expected results. The user chooses the test purpose in order to filter the LTS behavioral model.

The test purpose notation is a sequence of transitions. In this sequence, the "*" (asterisk) can appear at the beginning of the sequence or between transitions. The sequence

[1]http://www-306.ibm.com/software/awdtools/developer/rose/index.html
[2]www.rational.com/ products/rosert/index.jsp
[3]http://www.ttcn-3.org/
[4]http://www.inrialpes.fr/vasy/cadp/man/aldebaran.html#sect6



Path Coverage(%) 100.0
Type of test cases: ☐ With Feature Interaction.
Selected labels: Message Contains more than 1 URL; View MMS Message;

Test Purpose
*; Message Contains more than 1 URL; *; Store Selected URL; *

Options

*	1, 3, 5
Action List is displayed	
Choose URL	
MMS Message is displayed	
Message Contains more than 1 URL	2
Message Contains only 1 URL	
Store Selected URL	4
Store URL	
Store URL as Bookmark	
Store URL dialog is displayed	
URL Location Dialog is displayed	

Path Coverage(%) 100.0
Type of test cases: ☐ Test Cases With Feature Interaction.

Figure 2: LTS-BT tool screens - Test Purpose

always ends with either an Accept transition (meaning that the user wants all test cases that comply with the test purpose) or a Reject one. In the sequence, the asterisk means any transition. There are two ways to define a test purpose: either by clicking on transitions of the LTS graphical view (Figure 2(a)) or by clicking on buttons that represent transitions (Figure 2(b)).

3.1.3 *Generator*

Having the LTS behavioral model, the next step is to obtain the test cases. As we are considering functional test cases, the generated test cases are composed by user action, system response and initial conditions (if they are in the model). Then, each LTS path (initial state until final or visited state) is a test case. In order to do it, it is necessary to identify all paths from the LTS, to assure that all functionalities can be tested.

A path can be obtained, using the Depth First Search method (DFS), by traversing an LTS starting from the initial state. Test cases are generated according to a coverage criteria, for instance, transitions coverage, i.e., all transitions are visited at least once. Since we are considered functional testing, total coverage is a reasonable and feasible goal, to guarantee a thoroughly investigation of the feature functionalities [13].

3.1.4 *Selector*

This module aims to reduce the number of test cases in situations where full coverage is unfeasible and yet we aim to covering the model as much as possible. For this selection, the user must pass a percentage of coverage as a selection criterion. This strategy is inspired by the work of Lin and his colleagues [11]. The strategy reduces the number of redundant test cases according to a degree of similarity between them. The similarity is calculated by observing the number of identical transitions, i.e., states "from" and "to", and transition label are the same, and the average between paths length [4].

4. CASE STUDY

In this section, we use a quick case study to illustrate the use of LTS-BT for feature and feature interruption test case (TC) generation and selection. The kind of testing is selected by the user. This case study has been developed using version 1.0 of the LTS-BT Tool.

This case study is divided in two subsections. The first one presents feature test case generation and selection from a sequence diagram (SD) while the second one shows feature interruption test case generation and selection.

4.1 Feature test case

The focus of this test is to check all functionalities of a feature. These can be specified by a set of sequence diagrams. For the sake of space, we consider only a sequence diagram here. The scenario of this case study is "A user is viewing a message containing a URL and can go to it". The feature *Embedded URL in Messages* follow this scenario.

We use the SD of Figure 3. The LTS behavioral model is obtained from the sequence diagram. We chose the option to generate and select the test cases choosing the test purpose (TP) by clicking on a graph. The TP is *;"View text message";*;"Message contains more than 1 URL";*" and also the path coverage percentage is 100% (Figure 2(a)). The test case generated is shown in Figure 4. Even though the path coverage percentage was 100%, the use of a test purpose decreased the number of test cases, since only one test case attended the test purpose.

4.2 Feature interruption test case

In a typical feature interruption testing, the focus is on TCs that can uncover defects in the scenarios where one feature interrupts another feature, assuming that the features where thoroughly tested separately. The behavior model is composed of number of features and the specification of the possible interruptions. For the sake of space, we consider only one interruption here. In this case study, the behavior model has been specified as use cases and then converted into the AUT input format as in [7].

The scenario of this case study is "A user is viewing a message containing a URL and can go to it, but at any time the phone can receives a call". Two features contemplate this scenario: *Embedded URL in Messages* and the *Incoming Call*. Considering that the behavioral model has an incoming call interruption when the tester is reading the message containing a URL (Figure 5 portrays this scenario) and using the TP "*", LTS-BT generated 12 feature interruption test cases.

TCs with a specific purpose were generated. In order to test a scenario where a call arrives when the user is viewing

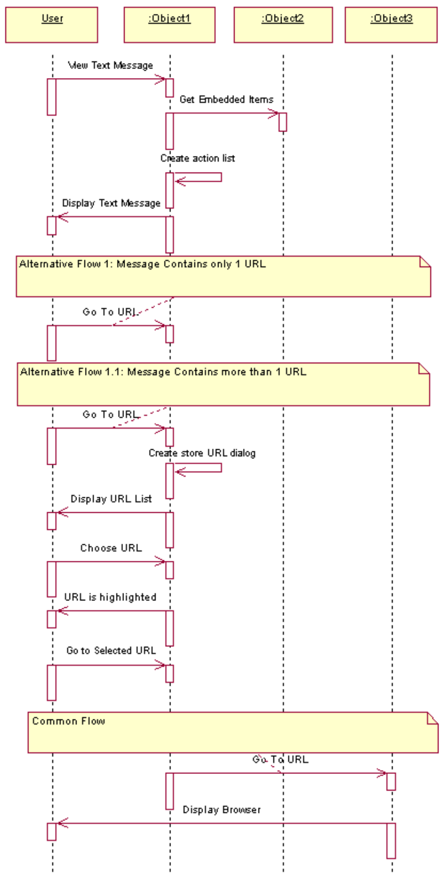


Figure 3: SD - Go to URL contained in a text message

the message we used the "*;Display Text Message;*;Incoming Call;*". In this case, LTS-BT generated 4 feature interruption TCs.

Note that the use of TPs is particularly interesting for feature interruption testing, since, in practice, the number of possible TCs is usually too big or infinite. TPs make it possible to achieve a reasonable coverage of a given interruption also focusing on certain functionalities.

4.3 Test Selection based on Similarity

For the sake of space, we can only present some statistics about the effectiveness of the use of this strategy in three case studies. According to the percentage of test cases to be select, the tool automatically selects the less similar ones. Table 1 presents the results obtained. We fix the percentage on 50%.

In general, similarity based selection discards less transitions than random testing and therefore provides a better coverage of the model. More detailed comparisons are presented in [4].

```
conditions: Message Contains more than 1 URL

 | Steps              | Expected Results     |
  ------------------------------------------
 | View Text Message  | Display Text Message |
  ------------------------------------------
 | Go To URL          | Display URL List     |
  ------------------------------------------
 | Choose URL         | URL is highlighted   |
  ------------------------------------------
 | Go to Selected URL | Display Browser      |
  ------------------------------------------
```

Figure 4: TC generated from a SD (Figure 3)

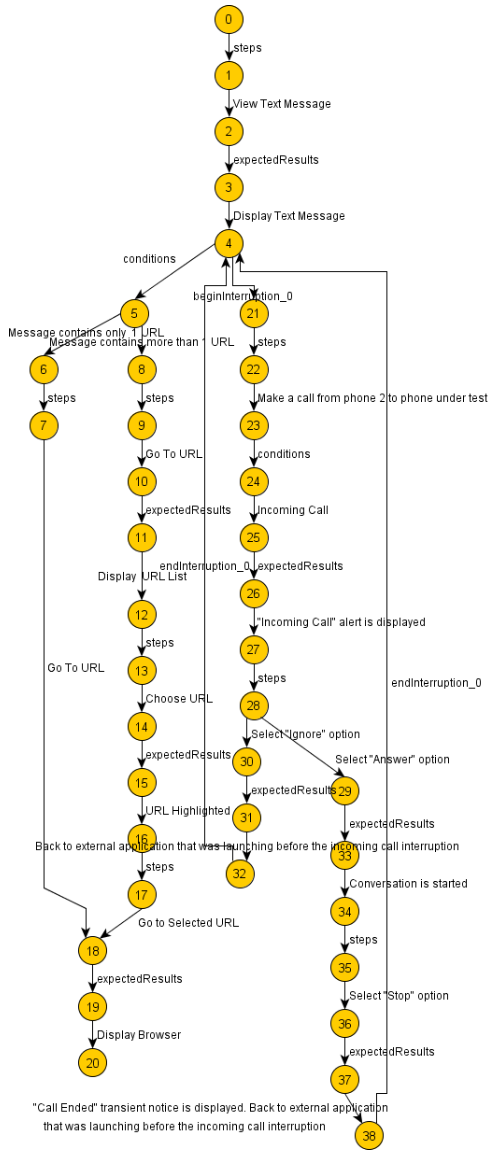


Figure 5: Scenario with an interruption

Table 1: Results

	AP1	AP2	AP3
Random Test Selection (%)	24.93%	36.10%	27.22%
Similarity Based Selection (%)	1.23%	11.98%	1.70%
Total of Test Cases	24	66	153

5. RELATED WORKS

Figure 2 presents distinguishable model-based testing tools (PTK [1], TGV/UMLAUT [9], TestMaster [6], and the ARCHETEST [15]) that can be considered as alternatives to the use of LTS-BT in the embedded systems domain. The main advantages of the LTS-BT tool are: (i) the input and output notations are LTSs that can be derived from and to a number of different notations, including UML, improving the chances of integrating the tool in different contexts/test processes; (ii)

Table 2: Comparing LTS-BT tool with other tools

	TestMaster	TGV/UMLAUT	LTS-BT	PTK	ARCHETEST
Input	EFSM	UML Diagrams IOLTS	Sequence Diagram Annotated LTS	HMSC MSC Sequence Diagram	High-level use-case class diagram
Output	Test Script Language (TSL)	IOLTS	Table Format LTS	TTCN SDL	Test Script
Feature Interaction	No	No	Yes	No	No
Selection techniques	-	Test purpose	Test purpose Similarity	-	-
Automatic test case generation	Yes	Yes	Yes	Yes	Yes
Abstract Test Case	Yes	Yes	Yes	No	Yes
User interface	Yes	Yes	Yes	Yes	Yes

this is the only tool that directly supports feature interruption testing - this can probably be done by using the others, but artifacts are very likely to become cumbersome and unpractical; (iii) selection techniques, especially the similarity strategy is a unique contribution that is key for the embedded systems domain where the costs of test execution are high while high coverage is crucial; and iv) test cases are necessarily abstract to manual testing and a user interface is staple for making it possible for domain experts to view and refine the test suites.

6. CONCLUSION

Automation of feature and feature interruption testing for embedded systems is proposed and a tool is presented. The tool is targeted at the reality of most embedded systems companies, since the test cases can be derived from different specification languages, specially sequence diagrams and also it focus on the specific, yet general format of these applications.

One of the main advantages in using the presented tool is related to the reduced size of the LTS, since LTS-BT combines the feature behavioral model with the interruption model, in the specified points where the interruption may occur. Moreover the several techniques used in the test case selection and generation provides a reliable coverage in both the LTS and test cases, providing a test suite suitable for scenarios where time and money constraints make the execution of all the test cases an unfeasible or undesired task (considering applications with a large LTS). Research on feature testing is practically nonexistent in academia. Feature interruption testing has been more thoroughly investigated. Nevertheless, suitable and effective tools are still needed.

LTS-BT has been experimentally used in real case studies in cooperation with Motorola testing teams. The goal is to assess its adequacy as well as cope with its evolution. These case studies has shown that the tool can promote productivity and reliability in test case generation. As further work, the tool needs to incorporate test case generation in the TTCN format and also cover a complete test process by interoperating with other tools.

7. REFERENCES

[1] Baker, P., Bristow, P., Jervis, C., King, D. J., and Mitchell, B. Automatic generation of conformance tests from message sequence charts. In *SAM* (2002), pp. 170–198.

[2] Binder, R. *Testing Object-Oriented Systems: Models, Patterns, and Tools.* 1999.

[3] Calder, M., Kolberg, M., Magill, E. H., and Reiff-Marganiec, S. Feature interaction: a critical review and considered forecast. *Comput. Networks 41*, 1 (2003), 115–141.

[4] Cartaxo, E. G., Neto, F. G. O., and Machado, P. D. L. Automated test case selection based on a similarity function. In *Model-based Testing 07 (Motes'07)* (Bremen, Germany, September 2007), Lecture Notes in Informatics. To appear.

[5] Cartaxo, E. G., Neto, F. G. O., and Machado, P. D. L. Test case generation by means of uml sequence diagrams and labeled transition systems. In *IEEE International Conference on Systems, Man, and Cybernetics 07 (SMC'07)* (Montreal, Canada, October 2007), IEEE. To appear.

[6] Clarke, J. M. Automated test generation (from a behavioral model).

[7] de Figueiredo, A. L. L., Andrade, W. L., and Machado, P. D. L. Generating interaction test cases for mobile phone systems from use case specifications. *SIGSOFT Softw. Eng. Notes 31*, 6 (2006), 1–10.

[8] Hartman, A., and Nagin, K. The agedis tools for model based testing. In *ISSTA '04: Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis* (New York, NY, USA, 2004), ACM Press, pp. 129–132.

[9] Ho, W. M., Jquel, J.-M., Guennec, A. L., and Pennaneac'h, F. Umlaut: An extendible uml transformation framework. In *Automated Software Engineering* (1999).

[10] Jard, C., and Jéron, T. Tgv: theory, principles and algorithms, a tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems. *Software Tools for Technology Transfer (STTT) 6* (2004).

[11] Lin, J.-C., and Yeh, P.-L. Automatic test data generation for path testing using gas. *Inf. Sci. 131*, 1-4 (2001), 47–64.

[12] Lorentsen, L., Tuovinen, A.-P., and Xu, J. Modelling feature interactions in mobile phones. In *Feature Interaction in Composed Systems (ECOOP 2001)* (Budapest, Hungary, 2001), pp. 7–13.

[13] McGregor, J. D., and Sykes, D. A. *A Practical Guide to Testing Object-Oriented Software.* Addison-Wesley, 2001.

[14] Metzger, A. Feature interactions in embedded control systems. *Comput. Networks 45*, 5 (2004), 625–644.

[15] Williams, C. E. Towards a test-ready meta-model for use cases. In *Workshop of the pUML-Group held together with the UML 2001 on Practical UML-Based Rigorous Development Methods - Countering or Integrating the eXtremists* (2001).