AGEDIS CONSORTIUM

AGEDIS

MODEL BASED TEST GENERATION TOOLS

Alan Hartman

hartman@il.ibm.com

www.agedis.de

Model Based Test Generation Tools

OVERVIEW

This report attempts to list the main tools for model based test generation in both the academic and the commercial arena. The aim of the report is to position the AGEDIS tools [1] in the context of what is currently available in both industry and academia.

WHAT IS A MODEL BASED TEST GENERATOR?

Arguably all software testing activity is model based, since any test case must be designed using some mental model of the application under test. In recent years the use of explicit models for software development (most notably the use of UML for object oriented analysis and design) has expanded greatly. The use of these models for the generation of test cases in the IT industry is still in its infancy, although a significant part of the telecommunications, aerospace, and micro-electronics industries have been experimenting with models for verification and test generation for over a decade.

We will define a **model based test generator** as an automated process which accepts two main inputs:

- A formal model of the software under test, and
- A set of test generation directives which guide the tool in its generation.

In many tools, the test generation directives are not supplied explicitly to the tool, but are "hard-wired" into the structure of the test generation tool. The output of a test generator is a set of test cases, which include a sequence of stimuli to the system under test, and the expected responses to those stimuli, as predicted by the model.

The process of model based test generation may be carried out manually or with bespoke tools created for the explicit purpose of generating a test suite for a single application. A good example is discussed in Robinson's paper [26]. The focus of this report is on *generic* model based test generation tools.

We distinguish between test generators and **model based input generators**, which do not output the expected response of the system under test. The models accepted by an input generator are models of the input sequences accepted by the system under test, but they do not model the expected behavior of the system. In order to use such tools for the automation of testing, the user must also provide an oracle for the system behavior. Two examples of model based input generators are Telcordia Technologies' AETG [27] and Maurer's DGL [16].

We also distinguish between test generators and **test automation frameworks**. A test automation framework accepts manually created, automatically generated, or pre-recorded test sequences and runs the sequences without human supervision. Typical examples of such tools are Mercury's WinRunner [17], Rational Robot [23], and Telelogic's Tau Tester [29].

Another class of tools which we do not include in this paper is the set of **modeling tools**. Tools in this category include Rational Rose [24], Objecteering [21], Poseidon [8], Together Control Centre [34], Statemate Magnum [12],AutoFocus [20], SimuLink [32], SCR [11], and Telelogic Tau [29][28].

These tools serve as the creators of input to model-based test generators, and do not in general have the capacity to generate test cases.

COMMERCIAL TOOLS

TVEC

T-VEC Technologies has a Test Vector Generation System [33] which accepts models of the requirements and behavior of a system in a proprietary language called T-VEC Linear Form. The product also comes with a graphical environment for creating SCR models and automatically translating them into the Linear Form. Translators exist for two other specification languages namely SCR [11] and MATRIXx [31].

The test generator generates test cases based on domain testing theory. It generates test cases by analyzing the logical expressions in the specification model, and creating test cases that reach extreme values in the decision path (a kind of branch coverage of the specification). The test generation directives appear to be implicit in the algorithm used to generate test cases.

The test cases produced by T-VEC include expected outputs.

The T-VEC system also has an interface for translating the abstract test cases into executable test programs (test driver generation).

The tool seems to have been used mainly in the aerospace industry. It does not appear to have any support for UML or other widely used modeling languages.

CONFORMIQ TEST GENERATOR

Conformiq Software Ltd. has announced a new product which is due for release in autumn 2002. The literature on their website [2] indicates that their tool will be a true model-based test generator according to the definition of this paper. The tool accepts UML state diagrams as the model of the system under test, together with support for real time properties. The main product of the generator's analysis is a behavior simulator which may be used in three different ways:

Batch mode: To generate test cases using the TTCN notation. These test cases may be run against the unit under test at a later time. The test cases contain expected results and verdicts.

Interactive mode: Running the test generator in synchronized mode with the unit under test. The behavior of the system is verified online.

Trace mode: A trace of system activity is run against the test generator and its output is verified as conforming to the specification model (or not).

The literature on the website is not specific about the coverage directives used by the tool, but it indicates that the tool includes a measure of the coverage of the system achieved by the test.

REACTIS

Reactis [25] is a tool produced by Reactive Systems Inc – a company whose focus is on tools for the development of embedded systems. It accepts models in the SimuLink and StateFlow modelling language [32].

The model is compiled and a model simulator is produced which can be used for manual or random test generation. Reactis also has an automatic test suite generation capability and the coverage directives are related to syntactic and structural coverage of the model (branches, states, transitions, conditions etc.)

The test cases contain expected results, but there does not appear to be any facility for translating and executing the test cases in a test harness.

JCONTRACT

The Parasoft Corporation makes a number of unit testing tools for Java (Jtest, and Jcontract) and other development environments. The tool Jcontract [22] enforces the use of Design by Contract, and uses the contract as a model for predicting the behavior of a class. Design by Contract (like Unitesk) involves the use of pre- and post-conditions, assertions, and invariants in the comments at the top of the class under test.

Jcontract and Jtest together provide automatic test generation and execution, testing the paths through the pre-conditions, and verifying that the assertions, invariants, and post-conditions of each method hold. The technology used for creation of the oracle is a form of dynamic symbolic execution of the code under test.

As with most commercial tools, the coverage criteria for the test generator are not given explicitly by the user, however the tools do come with code coverage measurements for the test cases it generates.

TAU TTCN SUITE

Amongst the Telelogic products is a suite of tools for the creation, simulation, and manipulation of SDL models. The Telelogic Tau TTCN Suite [28] webpage describes an ability to generate TTCN test cases from an SDL model. No details are given concerning the use of testing directives, but TTCN test cases certainly included expected results derived from the behavior specified in the SDL model.

TESTMASTER

The Teradyne Corporation produced a model based testing tool called Test Master (see [30]). This tool had a proprietary graphical language for specifying the system under test. The tool produced test suites by exhaustive traversal of the finite state machine used in the specification. Test Master had no special features for translating abstract tests to executable scripts.

The tool is not currently supported or sold.

UNITESK

The UniTesK (Unified Testing and Specification Toolkit) produced by ISPRAS (Institute for System Programming of the Russian Academy of Sciences) is on the border between a commercial and an academic tool (see [15]).

The model is specified in either J@VA or C@++, which are specification languages designed for Java and C++ code respectively. The specifications are in the form of pre- and post-conditions, and are coded as comments in the classes and methods to be tested.

As in the T-VEC tool, the test cases are generated by applying branch coverage of the specification of the post-condition, and this test directive seems to be implicit in the test generation process.

Since UniTesK is close to the source code, the test drivers are created as part of the test generation process, and not in a separate abstract to concrete translation phase.

The tool has been used by NorTel in testing parts of the kernel of a real time operating system.

PROPRIETARY TOOLS

GOTCHA-TCBEANS

GOTCHA-TCBeans is one of the ancestors of the AGEDIS test generation and execution tool suite. The GOTCHA-TCBeans tools are described in two papers [6][7].

The modeling language used by GOTCHA is an extension of Murphi [5]. The Murphi language is extended by the addition of test generation directives which allow for the specification of arbitrary projections of the state space to be used as coverage criteria (see [7]).

The test cases include expected outputs, and the test translation framework is provided by TCBeans.

The tools have been used within the IBM Corporation and the AGEDIS consortium, but are not publicly available.

UCBT-SALT

UCBT and SALT are tools produced by the IBM Software Engineering Center for system testing and API testing respectively.

The UCBT (Use Case Based Testing) tool accepts UML use cases and creates test suites in either a textual format or as input to the TCBeans execution engine. The user provides a set of use cases together with the inputs and expected outputs, and a diagram explaining the interactions between use cases. UCBT minimizes the number of test cases required to achieve the required level of functional coverage.

The SALT (Specification and Abstraction Language for Testing) tool, requires the user to give a specification of the software under test in a declarative proprietary language. The tool derives a set of test cases based on a fault model for the Boolean expressions used in the specification. The user may also specify code fragments and a test case template, in which case, SALT will generate executable test cases.

Both tools have been used within the IBM Corporation, and with selected IBM customers, but are not publicly available.

ASML

AsmL is the Abstract State Machine Language [18]. It is an executable specification language based on the theory of Abstract State Machines. The current version, AsmL 2 (AsmL for Microsoft .NET), is embedded into Microsoft Word and Microsoft Visual Studio.NET. It uses XML and Word for literate specifications.

There is a test generation tool which works with this specification language, and Microsoft researchers have reported on its characteristics in [10]. Their test generation algorithm appears to be total transition coverage of the derived finite state machine. The tool is not commercially available.

PTK

Ptk is tool developed by Motorola Labs in the UK for the generation of conformance tests from message sequence charts (MSC) and protocol data unit (PDU) specifications. It is described in a paper by Baker, Bristow, Jervis, King, and Mitchell [2].

Message sequence charts define a set of acceptable behaviors of the unit under test, rather than the entire set of behaviors described in finite state machine models. To translate message sequence charts into executable test cases is thus making explicit all the sequences and data needed to execute the behavior described in the sequence chart rather than choosing a subset of the sequences described by a state machine model.

The Ptk tool interprets the message sequence chart, making explicit test sequences and observations based on the externally controllable and visible aspects of the MSC, while hiding the internal actions and retaining the order that they impose on the execution sequence. Ptk also provides message data based on the PDU specifications, and creates several test cases from a single MSC based on the non-deterministic behavior described in the MSC.

The output of Ptk is either TTCN or SDL test suites, but it also has a code generator which can be customized to produce scripts in other languages.

ACADEMIC TOOLS

SPECTEST

SpecTest [9] is an automatic test case generator from George Mason University. It accepts models written in SCR or UML, and generates test cases based on a choice of two coverage criteria, with a further two planned for implementation.

The tools are not yet available for downloading.

MULSAW

The MulSaw project [19] at MIT incorporates two tools for the generation of test cases for Java programs, one tool (TestEra) accepts input in the Alloy modelling language, and the other (KORAT) accepts input in the Java Modelling Language (JML). The JML is similar to the language used by UniTesK, in that it places pre-conditions and post-conditions in a Java-like syntax as comments at the head of a Java method.

The KORAT test cases are generated to cover all instances of the method preconditions, and provide expected results based on the post-conditions.

The tools are not available for experimentation.

TOSTER

TOSTER (**T**he **O**bject-oriented **S**oftware **T**esting **E**nvi**R**onment) [37] is a test generation and execution system produced by the Warsaw University of Technology. It incorporates technology for mapping the information in UML diagrams to the source code of an application. It also generates and runs test cases based on expected results derived from the UML state diagrams. There appear to be two test generation algorithms, but no explicit testing directives.

TGV/CADP

TGV (Test Generation with Verification) [13] is a test case generator developed at the IRISA and VERIMAG laboratories. It is an ancestor of the AGEDIS test generation tools.

It accepts input in the form of a LOTOS, SDL, or IF specification model. The output of the test generator is in the TTCN format, and contains the expected results of the test. The test generation directives may be in the form of FSM models of the test purposes. The tool has been used extensively in experimental and industrial situations, mainly in the telecommunications industry.

The test generator is incorporated in the CADP package and may be downloaded from the website at [36].

TORX/CADP

The TorX is an architecture for test generation and execution from the University of Twente. Within this architecture there is a test generator which accepts test purposes in a similar format to TGV. The modeling languages supported by TorX are LOTOS, PROMELA, and SDL.

The TorX test generator is also incorporated in the CADP package [36].

COW SUITE

The Cow Suite tools and methodology was recently discussed in a paper by Basanieri, Bertolino, and Marchetti [3]. Their tools derive test cases from UML sequence diagrams and use case diagrams. Their test generation algorithm is essentially exhaustive enumeration of all possible use cases and message sequences. However they combine this exhaustive enumeration with a weighted selection strategy for test suites. The user is required to create a diagram containing a hierarchical organization of the UML diagrams together with a weight function indicating the "functional importance" of each diagram. From this diagram, the test selection strategy can then choose the most important set of test cases, or satisfy a functional coverage requirement based on the weights in the diagram.

The Cow suite tools produce a text file containing a description of the selected test suite, but they do not yet provide any features for executing the test suite. It is also unclear from the paper whether or not the tool provides expected results for each test case.

REFERENCES

[1] AGEDIS Consortium, **A**utomated **G**eneration and **E**xecution of Test Suites for **DI**stributed Component-based **S**oftware.

[2] Baker P., Bristow P., Jervis C., King D., Mitchell B., **Automatic Generation of Conformance Test From Message Sequence Charts**, 3rd SAM Workshop - "Telecommunication and Beyond The broader applicability of SDL and MSC", Aberystwyth, UK. 24-26th June 2002.

[3] Bassanieri F., Bertolino A., and Marchetti E., **The Cow Suite approach to planning and deriving test suites in UML projects**, Proc. Fifth International Conference on the Unified Modeling Language - the Language and its applications UML 2002, LNCS 2460, Dresden, Germany, September 30 - October 4, 2002, p. 383-397.

[4] Conformiq Software Ltd., Conformiq Test Generator.

[5] Dill D., Murphi description language and automatic verifier.

[6] Farchi E., Hartman A., Pinter S. S. **Using a Model-based Test Generator to Test for Standard Conformance**, IBM System Journal - special issue on Software Testing Volume 41(1) (2002) Pp 89 - 110.

[7] Friedman G., Hartman A., Nagin K., Shiran T., **Projected State Machine Coverage for Software Testing**, Proceedings of ISSTA 2002 International Symposium on Software Testing and Analysis (July 2002).

[8] Gentleware AG, Poseidon for UML.

[9] George Mason University, Spec Test - An automatic test case generator.

[10] Grieskamp W., Gurevich Y., Schulte W., Veanes M., **Generating Finite State Machines from Abstract State Machines**, Proceedings of ISSTA 2002 International Symposium on Software Testing and Analysis (July 2002).

[11] Heitmeyer, C., Naval Research Laboratory, SCR Tool.

[12] I-Logix, Statemate Magnum.

[13] IBM Center for Software Engineering, UCBT and SALT.

[14] IRISA Laboratory, TGV.

[15] ISP RAS Red Verst, UniTesK.

[16] Maurer P. M., Data Generation Language.

[17] Mercury Interactive, WinRunner.

[18] Microsoft Research, Abstract State Machine Language.

[19] MIT Software Design Group, MulSaw.

[20] Munich University of Technology, AutoFocus.

[21] Objecteering Software, Objecteering UML Modeller.

[22] Parasoft Corporation, Jcontract.

[23] Rational Software, Rational Robot.

[24] Rational Software, Rational Rose.

[25] Reactive Systems Inc, Reactis.

[26] Robinson H., Finite State Model-Based Testing on a Shoestring.

[27] Telcordia Technologies, AETG Web Service.

[28] Telelogic, Tau TTCN Suite.

[29] Telelogic, Tau/Tester.

[30] Teradyne, Test Master.

[31] The MathWorks, MATRIXx graphical CASE environment.

[32] The MathWorks, Simulink and StateFlow.

[33] T-VEC Technologies, Test Vector Generation System.

[34] TogetherSoft, Together Control Centre.

[35] University of Twente, TorX.

[36] Vasy Inria Rhone Alpes, Caesar/Aldebaran Development Package (CADP).

[37] Warsaw University of Technology, Toster.