A TECHNIQUE FOR TESTING HIGHLY RELIABLE REAL-TIME SOFTWARE

Slide Presentation
Biography(s)

Abstract

The engineering of software systems that must be highly reliable is very difficult, and support tools and techniques are clearly needed. We are developing a technique and an associated tool set that uses executable specifications based on Annotated Ada (Anna) for software testing in hard real-time environments. Our initial tool, the test range oracle tool (TROT), supports the creation of simple test oracles that check the correctness of equation execution; future tools will have expanded capabilities. TROT was designed using commercial-off-the-shelf and reuse software to fit in with an on-going software development process and does not interfere with the software under test.

1.0 INTRODUCTION

Automating the verification and validation testing of software systems remains a problem area in software engineering. Testing software that must be highly reliable requires a large number of test cases. Analyzing and determining the correctness of these large number of test cases is very difficult and time consuming. Software engineers must often rely on engineering judgment or use hard to define success criteria which are often visually compared with actual program output. These approaches introduce many opportunities for human error, and are difficult to repeat or use in regression testing. Tools that will help engineers automatically determine the correctness of program output during testing would be helpful [1]. Such tools are generally called "automated test oracles." An oracle is a tool or technique that will determine the correctness of execution results for input-process-output operations. As the TROT approach shows, the correct execution of software during testing can be automatically determined from a set of executable formal specifications.

A formal specification language is one that defines "what to do" requirements in a clear

and unambiguous fashion. Specifications express information about a program that is not normally part of the program, and often cannot be expressed in a normal programming language [3]. Formal specification languages are usually mathematically based, because English based requirements can be misinterpreted. Formal specifications have been advocated for use by the software engineering community. Specifications are commonly considered part of the up-front development process, to be used during requirements and design. The practical use of formal specification methods has been slow or non-existent due to the difficulty of learning a technique and the cost of developing specifications [5]. The benefits and successful application of these specification techniques must be demonstrated before they will be used in industry. An approach to incorporating formal methods is to incrementally introduce them into the typical software engineering project. In this paper, we will show how formal specification techniques can be effectively used to support software testing. Using formal specifications to aid testing is not new, having been proposed by [6, 7], however, this is a new and practical application of them to the high reliability real-time domain.

In this paper we examine the creation of automated test oracles, developed using a formal specification language, and associated assertions about the software under test. Our tool was designed to fit in with an on-going software development process. We demonstrate this technique in a development environment where code and English requirements already exist--where numerous testing cycles have been completed and testing must be done repeatedly on a non-interference basis, i.e., no software instrumentation or in-line self-checking code is allowed. A prototype self-analyzing test case tool has been produced that supports testing of the real-time system without significantly interfering with the software being tested. Our technique is based on the practical use of formal specifications that appear as comments embedded within Ada code. We use the Annotated Ada (Anna) design-based specification language and the associated tool set, combined with a real-time test bed that incorporates a bus logic analyzer. The Anna-Ada code takes the form of special test scripts that form the test oracle. This system, while evolutionary in nature, is being developed to test aspects of an actual production system. For our purposes, the system under test has many numeric equations and exists in the embedded avionics system control domain, such as, spacecraft, aircraft, and boosters. Specific projects where this technique is being applied are not relevant or mentioned by name. The TROT tool, while based in Ada, is used to test a system developed in a language other than Ada.

This paper first introduces the Anna language and supporting tools with high level design information for the TROT system components. After the tools are introduced, the TROT testing system approach is outlined. The paper also outlines example applications of TROT. And, since the tool is a prototype, future work and expansion of

the TROT tool is outlined. Finally, the conclusions and results from this work are provided.

2.0 ANNA SYSTEM AND LANGUAGE DESCRIPTION

The supporting language for the TROT tool is the Anna language. The Anna language has a supporting tool set that was developed at Stanford University. Although the Anna tool set itself is a prototype, it currently has sufficient capabilities to be used as part of our system. Since Anna is written in Ada, the tool set can be made to run on numerous hardware platforms; we use a Silicon Graphics computer system. Anna is a language extension to Ada that allows the formal specification of the intended behavior of programs [3, 4]. In TROT, the Anna language is used to specify information that is derived from a requirements document. A support environment is available in the Anna language which provides a fairly robust tool set. The tool set is capable of transforming Anna constructs into executable programs.

Of the languages considered for the TROT system, Anna was chosen for the following reasons.

- Anna supports Ada.
- Anna's tool set can be used to produce executable code from specification.
- Reusable Anna software and support tools/environment are available on the INTERNET.
- Anna supports design-based specifications.

Anna adds requirement specification capabilities to the Ada language. This is achieved by adding the following three basic extensions to the Ada language:

1. Annotations that generalize the Ada concept of constraint;
2. Virtual code (a program that is separate from the "main" program, but that describes the main program); and
3. The capability to define all aspects of Ada program interfaces (above the standard Ada checks).

For TROT, however, only annotations and virtual code are used. These extensions provided by the Anna language constructs add capabilities that are over and above the normal expressive power of Ada. Using statement annotations, we can define program specifications and conditions that must be true at a given point in the program. In

addition to Anna-unique constructs, an Anna program can contain any Ada language construct that is semantically legal.

The Anna tool set includes the following tools that were used in support of TROT:

- Anna Program Verifier - Chromatic Theorem Prover
- Anna Syntax checker - Check structure of Anna constructs
- Transformer - Anna-to-Ada code generator that creates self checking Ada
- Anna Debugger - Debugs Anna language constructs
- Specification Analyzer - Semantics checking for consistency of constructs and libraries

These tools support the generation of executable Anna-Ada programs. Annotations are converted into Ada code that can be compiled by a normal Ada compiler. The version of code with the Anna-Ada code inserted into it is called a "self-checking" program, because each time an annotated code sequence is encountered, it is checked for correctness to the associated code. The tools also support Anna syntax and semantics checking as well as the debugging of Anna programs. The tool set works within the Ada language environment.

2.1 ANNOTATIONS AND VIRTUAL CODE

Code annotations and virtual code are essentially comment fields within Ada programs. Annotations appear as "--|" followed by Anna language extensions. Virtual code appears as a
"--:" followed by legal Ada code. Other systems have used comment annotations to embed specification information within code [1 & 6]. Annotations and virtual code are not compiled by a normal Ada compiler, and so do not normally interfere with the code they are annotating, but they can be interpreted by the Anna tool set to generate Ada code that does execute. Figure 2.1-1 gives an example of some simple Anna code annotations. Code annotations can appear in numerous places within a program.

```
Read (Test_Var)

--|   for all Test_Var  =>

--|      (Input <= 0.0 and Test_Var = 0.0) and

--|      (Input <= (Test_Var * Test_Var) + 0.0000000000001) and
```

```
--|      (Input >= (Test_Var * Test_Var) - 0.0000000000001)
```

Figure 2.1-1 Anna Specification Code

2.2 ANNA LANGUAGE CONSTRUCTS

There are a large number of Anna constructs. This section introduces some Anna constructs to aid in understanding what the Anna language is and can do. For a complete treatment of the Anna language see [4].

1. => - left-side/right-side separator symbol for the collection operation, a.k.a. yields

2. for all - Universal quantifier

```
-- Names of months do not contain the letter x

--|   for all X : MONTH range JAN .. DEC  =>

--|      for all index_month : 1..MONTH'WIDTH =>

--|                                   MONTH'IMAGE(X)(index_month) /= 'X';
```

3. exist - Existential quantifier

```
-- there exists a least natural number

--|   exist X : NATURAL;  for all Y : NATURAL  => Y _ X
```

In addition to these non-Ada constructs, there can be virtual types, variables, and procedure declarations in Anna.

3.0 THE TROT SYSTEM

The TROT system that we developed has two major components: the test environment and the analysis environment (see figure 3.0-1). Each of these has several major sub-components.

Figure 3.0-1 Major Components and Sub-components of the TROT System

In the TROT approach, the software under test is executed on a test bed system that

has low-level CPU monitoring capability. For this prototype, CADRE's Software Analysis Workstation (SAW) system was used to monitor the CPU while a second computer was used to control the CPU. This test environment is made up of hardware and software components that are detailed later. Trace files are generated in the test environment and transferred to a separate workstation analysis environment for post-processing by the TROT test case software. The TROT test cases are created by a tester using Anna specification and Ada code. TROT test cases are run through the supporting Anna tool set to produce a program capable of automatically analyzing the SAW trace files for correctness, based on requirements and associated inputs that were given to the software under test.

The TROT test case software is made up of differing sub-components. The foundation of the tool set is the Anna language and its supporting tool set. Using Anna and Ada code, a test case is constructed from English requirements. The TROT test cases are passed into the Anna tool set and then an Ada compiler to produce an executable test case. These test cases then make a pass-fail judgment on the SAW trace file results and produce test documentation.

3.1 TROT SUPPORTING TECHNOLOGIES AND SYSTEMS

The TROT system has several supporting computers and software systems. As shown in figure 3.1-1, these are comprised of two IBM compatible computers, the CPU of the system under test, and the supporting software.

Figure 3.1-1 Software Analysis Workstation and Support Systems

The Personal Computer Test Set (PCTS) allows basic memory load and CPU control functions over the target processor. We have implemented this on an IBM compatible PC with customized software and connections to the CPU under test, but many vendors of embedded CPUs offer similar systems as part of an embedded computer development environment, and these could be substituted in place of the PCTS.

The computer under test is a 16-bit processor with hardware interfaces to a surrounding avionics environment. We have the first level of the hardware avionics, interfaced to the CPU. There is also a computer system that interfaces with the avionics hardware that allows a system simulation to the computer and software under test.

The SAW [2] is on a second IBM compatible PC. The SAW is a commercial off-the-shelf system that is user modifiable (both hardware and software). The SAW has an interface that allows it to monitor the memory reads and writes to the CPU under

test. The SAW traces and traps computer activities without significant interference with the software under test. A system of hardware probes driven by configurable software components, monitors computer activity.

The tester configures the SAW to trap the execution of a particular sequence of memory addresses that correspond to the instructions being executed by the CPU under test. This forms a trace that corresponds to the assembly language operation codes and CPU activity. The trace file can be saved as a separate output file by the SAW. These trace files are inputs to the TROT software test case.

The SAW, with its execution trace results, produces the following types of information:

- executed assembly language statements via a disassemble and non-intrusive hardware logic probe on the CPU bus;
- timing numbers (time tags each statement because the SAW's clock is faster than the system under test);
- memory addresses accessed and changed (memory inputs and output results); and
- computer status information.

The trace file is retained as an ASCII file for post processing by the TROT test oracle. A series of runs and associated traces can be made and retained in this fashion. 3.2 TROT Test Case

For this version of TROT, we used CADRE's SAW to monitor the CPU under test, however, the use of tools that prove trace file capability could be easily integrated as a replacement for the SAW. We have investigated the use of tools like ITCN's C-TAC or all-digital computer simulations like TARTAN's interpretive computer simulator (ICS). We are currently planning a future version of TROT to use TARTAN's ICS.

The last major component of the TROT system is the TROT test case itself. An executable TROT test case is a program produced from an Anna specification and supporting Ada libraries. The TROT test case starts from a standard template of Ada-Anna code. The following defines the structure of a typical TROT program template. It has a body that:

- initializes files used in processing;
- reads in needed files;

- provides access to the variable being tested;
- provides numeric conversion utilities;
- provides the base Anna requirement that is being tested;
- provides a reporting mechanism; and
- handles exceptions raised by the Anna logic.

A TROT test case is first coded in Anna and Ada using this standardized template. A tester implements English requirements in Anna. The tester also identifies as part of the Anna specification, the code variables that are under test. The variable-name under test appears in the trace file as part of what the SAW disassemble provides. These are what TROT uses to gain access to the data values associated with these variables.

The logic flow of a TROT test case is shown in Figure 3.2-1.

Figure 3.2-1 TROT Logic Flow

The main part of the TROT process is in step 3. The rest of the logic is mostly file manipulation and reporting logic. The TROT flow allows a variable computed by the software under test and accessed from the SAW trace file, to be converted into a numeric representation acceptable to the workstation where TROT is executing. The conversion of a value from one format to another is needed because the internal representation of numbers in the system under test is different from that in the workstation, and the workstation's format of numbers and the associated calculations offers a higher, more accurate representation of the number. Conversion is needed for floating point and integer numbers. Once a value has been converted, it is then compared to Anna based annotations. These automatically determine if the specified conditions of a variable are met. The compare capabilities are a normal part of what the Anna tool set and language provide. For variables that pass the annotation's specifications, a nominal report is issued at the end of TROT test case processing. However, if an annotation is violated, an Anna exception is raised. These are handled by an exception handler that traps the violation and issues a failed test report. Thus, if a test case works, a nominal report is issued indicating that all test conditions passed. And, if an exception is processed a special report is available. Nominal use of TROT is not to have any Anna exceptions, because this means a requirement is not being met.

In our implementation, the following areas are Ada reuse packages that all testers use:

- Target-to-workstation numeric conversion routines;
- File Open-Close routines;
- SAW trace file access logic; and
- Reporting and exception processing.

These, in addition to the template approach, minimizes the coding a tester must do to the test related Anna-Ada code, variables, and assertions.

4.0 APPLICATION AND USAGE OF TROT

The following section is a brief overview on the use of TROT. The TROT system supports:

- The use of formal specifications to do equation analysis (accuracy, range, and resolution);
- An automated “comparator” function of required-to-actual results;
- Processing of multiple input runs;
- Reporting results and summary information (for example, failure/success statistics and metrics); and
- Testing on a non-interference basis with the system under test.

Figure 4.0-1 Simplified Software Project Life Cycle

A simplified software project life cycle is shown in figure 4.0-1 with parallel test processes where the tester creates TROT assertion test cases and then uses them in testing. Conceptually, TROT can be viewed as a form of N-Version programming [8], where by two or more versions of the software are created. However, in the TROT system, one version is "how-to-do" code and the other is "what-to-do" specifications. In TROT N-version, code which is used in the test process does not impact the timing and memory of the system under test. N-version programming has been advocated as contributing to the production of fault-tolerant and reliable software[8]. N-version testing of software establishes a basis for automated comparison and thereby increases confidence in the software under test. However, N-version programming does not guarantee that both versions of software do not contain duplicate errors [8], but with the use of formal specifications, this risk can be reduced. A more rigorous development

process using a formal specification will help avoid certain errors.

Figure 4.0-2 Typical TROT Test Cycle

Figure 4.0-2 outlines a typical TROT test flow process. Before testing, a tester must decide what area of code will be tested. This decision drives both the creation of the Anna test case assertions and then execution of the SAW. First, a tester creates the TROT test case in Anna and Ada. Once created, the Anna test case is entered into the Anna tool set. The Anna tool set checks things such as syntax and some semantic structures and produces a "self-checking" Ada program that can be used to process the SAW trace files.

After an executable TROT test case is ready, the SAW computers are brought up and initialized. This initialization takes place by using the PCTS to load a binary image of the software under test into the actual target processor's memory. Once loaded, the PCTS is used to configure the target computer's initial state. Initial conditions are determined by the testing that is to be done; currently this is a manual process. After the target CPU is configured, the tester brings the SAW up and configures it. The SAW is configured with a file that defines the trace and CPU under test. Once the SAW is active, the PCTS is used to start the software under test and at the appropriate trigger point, the SAW captures the defined trace information.

By doing multiple target CPU initializations and SAW set ups, a tester can generate a series of test runs and associated trace files. These allow for multiple test sessions. Once captured, trace files are transferred to the system where the TROT test case is to be executed. In our case, we ran the TROT test case on a Silicon Graphics computer due to its improved performance (over the IBM) and the support for Anna and word size, but there is no reason not to use other systems, if they support Ada.

Finally, the trace files are processed by the TROT test case. This is accomplished by simply running the TROT executable file. It produces a test results report which includes specific identification of any failed constraints. Failures require engineering analysis to determine cause and corrective action. The entire package of TROT test cases and results constitutes a complete package of test documentation that can be retained (as required) by the project software development and test plans.

4.1 EXAMPLE

An English requirement states:

"SQUARE_ROOT shall be a single argument entry algorithm which calculates the positive square root of the input argument, accurate to 13 decimal bits. For inputs of zero or less, the output shall be 0."

A part of a simplified TROT test case for this is shown in Figure 4.1-1.

```
Read (Test_Var)
--|   for all Test_Var  =>
--|      (Input <= 0.0 and Test_Var = 0.0) and
--|      (Input <= (Test_Var * Test_Var) + 0.0000000000001) and
--|      (Input >= (Test_Var * Test_Var) - 0.0000000000001)
```

Figure 4.1-1 Anna Specification Code

The result Test_Var is read from the SAW trace. It has been computed by the software under test. In our prototype, the software under test is executed on a 16-bit processor and suffers from the limitations of 16-bit accuracy problems. The TROT test case software is executed on a 32-bit processor and uses Ada typing to guarantee more accuracy. So it is possible to check the accuracy requirement to the actual accuracy delivered by the target computer's square root algorithm.

When the TROT software is executed, it accesses the input to the square root function and the result from the function. These are used by the TROT test case to determine if the stated annotations are violated. If they are, an Anna exception is raised and then handled within the TROT test case.

During analysis of this example, we found that the square root function had only been partially tested because a human tester overlooked a data input value. This error was found and corrected during analysis using the Anna prototype specification. The problem here was that the tester had only one data value less than zero, but in fact needed to have values of zero and less than zero. Also, the determination of accuracy had been partially left to human judgment and not fully automated, and therefore was doubtful. In this example, the TROT test case seems to perform better than a human

expert.

4.2 TROT PRELIMINARY RESULTS

To date, TROT has been used to test initialization routines and some simple math utilities. The TROT test case automatically showed that a selected series of variables had been correctly computed by the software under test and that the variables were within required accuracy values. The reported results needed virtually no analysis other than printing them out and then visually checking them to ensure there were no exception processing messages.

4.3 CURRENT TROT LIMITATIONS

The current version of TROT is a prototype. Many aspects of the system currently are simplistic in nature or involve human interaction that is being automated. A spiral life cycle model has been followed where each prototype of TROT builds on information learned and problems solved from a previous cycle iteration. Examples of current limitations are defined below.

- The current TROT test case system does not have complete access to compiler information such as tables, arrays, etc.
- Cross reference dictionary between requirement-based names and code-based names will aid in test case generation from the English requirements.
- The current versions are limited to single equations or simple code sequences. Expansion to cover module and object level assertions is desirable.
- Additional information about path, timing, and logic flow with program sequences will enhance module level testing.
- Simplistic control using the PCTS and SAW trace information complicates TROT testing and analysis.

We have also experienced problems and design limits with the Anna tool set which have limited what we can do and required workarounds, however, that is not unexpected since the Stanford tool set is a prototype. For production use, a more robust version is planned.

5.0 CONCLUSIONS AND SUMMARY

The TROT system is a feasible, non-interfering, Ada-based test system, which can be used for automated verification that an implementation meets requirements using formal specifications. This project shows the usefulness of formal specification languages like Anna to support testing of required equation results and accuracy. TROT testing support is accomplished in a real-time environment without interfering with the software using a variety of available reuse and commercial components. This initial version of TROT is designed only to test variable range and resolution requirements of equations and math functions. It does this without incurring any of the overhead associated with normal Ada type enforcement. And, it can be used to analyze the accuracy and numeric representation of test results on a higher fidelity machine than the system under test.

It will be possible to extend this work. Expansion will include accommodating other requirement structures and integration with initial phases of development so that test information can be generated at the earliest phases of software development. TROT is being integrated into an on-going test and inspection environment. It should prove especially useful in verifying and validating software systems that must be highly reliable and that are subjected to a lot of testing during their life cycle. Expansion of TROT into areas such as engineering components for reuse and regression testing will be a natural outcome of this work.

Although our tool set is developed using Ada, the technique that we introduce does not depend on source language or target processor. The technique and tool set can be applied to any system where statement execution and resultant information are available. In fact, we apply our technique and tools to a project which is implemented in a language other than Ada and based on a non-standard computer processor platform. Also, our test system uses commercial and reuse components to conduct test activities. Only minimal customization will be needed to implement a system such as ours for other project domains. An interesting outgrowth of this work will be systems or software that can incorporate requirements, code, and test information into a single related package that can be used and reused.

6.0 REFERENCES

1 Bieman and Yin, "Designing for Software Testability Using Automated Oracles", Proceedings International Test Conference, pp 900-907, September 1992

2 CADRE, Workstation Environment Software, Manual Package - 02584-04033,

CADRE Technologies Inc. 1988.

3 David C. Luckham, Friedrich W. von Henke "An Overview of Anna, a Specification Language for Ada", IEEE Software, IEEE, 1985, pp 19-22.

4 David Luckham, An Introduction to Anna, A Language for Specifying Ada Programs, Programming with Specification, Springer-Verlag, 1990 .

5 David S. Rosenblum, "Towards a Method of Programming with Assertions", Proceedings of the 14th Int. Conference on Software Engineering, ACM Press, pages 92-104, May 1992

6 D. J. Richardson, O. O'Malley, C. Tittle, "Approaches to Specification Based Testing", Proc ACM SIGSOFT 89 TAV3, ACM no 8 vol 14, Dec. 1989, pp 86-89.

7 D.J. Richardson, S. Aha & T. O'Malley, "Specification-based Test Oracles for Reactive Systems", TBD - Proc. 14th IC on SW Eng, Melborne, Australia, IEEE 1992, pp 11-13.

8 J. Knight, P. Ammann, Testing Software using Multiple Versions, SPC publications, June 1989.

9 K. Olender, J. Bieman, "Using Algebraic Specifications to Find Sequencing Defects", Proceedings of Int. Symposium of Software Reliability Engineering (ISSRE 93), pp 226-232, Nov. 1993.

Contact:
Jon Hagar
Martin Marietta Astronautics Company
P.O. Box 179, M/S H0512
Denver, CO 80201
Voice: (303) 977-1625
Fax: (303) 977-1472
INTERNET: hagar@den.mmc.com

or

Dr. James M. Bieman
Colorado State University
Computer Science Department

Task Plan
®

Fort Collins, Colorado 80523
(303) 491-5792
Internet: bieman@cs.colostate.edu