Developing a TTCN-3 Test Harness for Legacy Software

Joseph C. Okika
International Institute for Software Technology
The United Nations University
Macao
joseph@iist.unu.edu

Anders P. Ravn
Department of Computer Science
Aalborg University
Denmark
apr@cs.aau.dk

Zhiming Liu
International Institute for Software Technology
The United Nations University
Macao
Z.Liu@iist.unu.edu

Lokesh Siddalingaiah
Department of Computer Science
Aalborg University
Denmark

ABSTRACT
We describe a prototype test harness for an embedded system which is the control software for a modern marine diesel engine. The operations of such control software requires complete certification. We adopt Testing and Test Control Notation (TTCN-3) to define test cases for this purpose. The main challenge in developing the test harness is to interface a generic test driver to the legacy software and provide a suitable interface for test engineers. The main contribution of this paper is a demonstration of a suitable design for such a test harness. It includes: a TTCN-3 test driver in C++, the legacy control software in C, a Graphical User Interface (GUI) and the connectors in Java. Our experience shows that it is feasible to use TTCN-3 in developing a test harness for a legacy software for an embedded system, even when it involves different heterogeneous components.

Categories and Subject Descriptors
D.2.5 [Testing and Debugging]: Testing tools

General Terms
Experimentation, Documentation and Languages

Keywords
TTCN-3, embedded system, legacy software, test harness

1. INTRODUCTION
Testing software for embedded systems has always been a major concern, as such software directly controls machines. Finding an appropriate technique to test such software without changing the code and disrupting its usage is important since the system is certified as a whole. A concrete example is the control software for a diesel engine, where the certification covers a complete system and is based on tests of the software with specific hardware sensors, actuators, buses and processors.

The Nature of the Control Software
A marine diesel engine is composed of several parts ranging from mechanical, electrical to electronic as shown in Figure 1. The engine series has different units such as the Cylinder Unit, Injection System, Turbo Charger System etc. All these are connected together to form the complete engine which can be controlled remotely from the control room or through a bridge (knobs that are adjusted manually).

A marine diesel engine is controlled using an engine control unit (ECU). This is an electronic device, basically computers, and some networks that read several sensors and uses the information to control the actuators for example fuel injection. The ECU allows an engine’s operation to...
be controlled in great detail, allowing greater fuel efficiency, better power and responsiveness, and much lower pollution levels than earlier generations of engines.

The ECU is connected to several units of the engine through a system bus. A schematic view of the ECU is shown in Figure 2. The ECU controls and monitors the flow of signals in this setup. Essentially, each unit has a sensor and actuator through which control and monitoring is effected.

**Current Test Procedures**

In order to develop a useful test harness, it is important to look at the current test procedure as used in the case study. The manual test procedure for an ECU is done by connecting a computer to a physical board. The physical board has the ECU and the sensors terminal. The connection between the board and the computer terminal is through the use of modbus [3]. During operation, input data in text format is passed to the ECU to emulate sensor readings. The changes made to the data are converted back to a text format. This enables a test engineer to read values exchanged and to check that the values are valid.

This setup has a test engineer who is permanently stationed to effect/monitor the exchange of text values. Currently, a test engineer uses a number of special scripts and documents to test a new release. These tools are developed more or less ad hoc, e.g., using spreadsheets and word processors. Although there is a thorough and documented test procedure, it is clear that it is very time consuming and highly dependent on the internal knowledge of the test engineers, therefore an alternative solution is very desirable and should be devised.

**Toward a TTCN-3 Solution**

A novel solution should certainly give a better documentation of the knowledge of the test engineers. Thus there is a need for a language to write test cases and expected outcomes of tests. Testing and Test Control Notation version 3 (TTCN-3) is a standardized specification and testing implementation language which is extended from previous versions of TTCN called Tree and Tabular Combined Notation. TTCN-3 is being gradually introduced in the embedded system community, but the TTCN-3 capability is yet to be fully explored. This may be due to the previous versions of TTCN which were mainly targeted at telecommunication applications.

We believe TTCN-3 can be used to effectively test embedded systems because of the following features:

- **Concurrency:** TTCN-3 allows the specification of dynamic and concurrent test systems. It supports creation of multiple test components and their interfaces. This answers the question of how to handle concurrent processes which is prominent in embedded systems, although our legacy software contains only one control loop.

- **Deadlock Handling and Timed Invocations:** TTCN-3 guards against deadlocks through the use of exception handling and time out. This means that a time-out period is set for call operations. If no reply or exception is received within the specified period, TTCN-3 will generate a timeout exception.

- **Real-time Operations:** TTCN-3 allows operations to be performed at real-time. The platform adapter of TTCN-3 implements the model of time to be used during the execution of TTCN-3 functions.

The challenge here is how to develop a test harness which incorporates a TTCN-3 test driver with the legacy software, and a graphical user interface that allows engineers to run the tests. This test harness shall replace the current setup which involves manually configured test drivers and several script formats with a single test script driver and a single test script format. This promises a more efficient test procedure and a potential for easier test reconfiguration for new enhanced versions of the control software.

**1.1 Related Work**

TTCN-3 is applied in all kinds of black box testing for both reactive and distributed systems. Notable examples outside the telecommunication industry include mobile, Internet, and CORBA systems. As [13] noted, the special advantage of the generic testing language TTCN-3 is that it has wide coverage of different kinds of software systems. Furthermore, it is based on series of standards which means that there is very little risk of obsolescence.

Among the applications of the TTCN-3 to embedded systems, is the testing of an Automatic Teller Machine (ATM) developed by Jens Grabowski and others in [10]. They give a detailed introduction to TTCN-3 using this ATM example. The ATM system allows a customer to withdraw money from his account. The ATM hardware gives a customer access to the ATM system. The bank database is needed to check and update the credit balance of a customer. They considered a complete and successful money withdrawal process and abstract from the hardware interactions like putting the credit card into the ATM hardware, typing the PIN number or delivering money, by modeling such interactions in the form of messages. This work gives a detailed explanation of the implementation of TTCN-3 based systems.

The framework in [15] goes through several stages to automate the testing of XML/SOAP based web services. These include; generation of test data structure, test data, and test behavior from XML to TTCN-3 followed by compilation to executable tests. The test cases are compiled from TTCN-3 to Java. The compiled test cases are executed using TTCthree [5]. This gives an insight on how to capture test behavior in a TTCN-3 module.

J. Zander et al. [17] describe an approach to derive executable tests from UML 2.0 Testing Profile (U2TP) diagrams automatically. The executable test cases were derived within the TTCN-3 and transformations are made
from the source U2TP meta-model to the target TTCN-3 meta-model. The interesting part of the approach is its plug-in to the Eclipse Platform [2]. This work explains how to plug-in the test driver with the system being tested.

In testing of web services similar to [6], TTCN-3 has been found useful as seen in [16]. The authors looked at the language and platform independence characteristics of web services that bring difficulties in testing and proposed a distributed testing process based on TTCN-3. The test cases were not executed, but it gives an in-depth description of the use of TTCN-3.

Furthermore, the possibility of generating TTCN-3 test cases from UML Testing Profile (a language for designing, visualizing, specifying, analyzing, constructing and documenting the artifacts of test systems) has been demonstrated in [14].

1.2 Overview
The rest of the paper is organized as follows. Section 2 describes the test harness and the concepts of TTCN-3. Section 3 describes how the test harness is designed. Section 4 presents the implementation and use of the test harness, and Section 5 concludes.

2. A TEST HARNESS AND TTCN-3
Testing is the process of exercising software to verify that it satisfies specified requirements and to detect errors. It is the process of analyzing a software item to detect the differences between existing (which may include bugs) and required conditions, and to evaluate the features of the software item [1]. Furthermore, as noted in [18], software testing is a process in which a software system’s dynamic behavior is observed, recorded and analyzed so that properties of the system can be inferred from the information revealed by test executions.

2.1 Test Harness
A test harness is a program which sets up a software environment in which tests will be run, and any other software with which the software under test interacts when under test. It calls a test case (a set of inputs, execution preconditions, and expected outcomes developed for a particular objective).

A test harness collects test cases into suites and provides an environment for them to be run as a batch and the results to be displayed or stored for future reference. In this paper, the test harness is organized into three major components:

1. A TTCN-3 test driver integrating the compilation, execution of test cases and logging of relevant information.
2. A user interface where the selection of test cases to be executed and other relevant commands are issued.
3. The software under test (SUT). Here it is the control software with a global data table containing the data exchanged at different phases of its operation.

2.2 Core TTCN-3 Concepts
We illustrate here, the elements of TTCN-3. The syntax of the TTCN-3 core language is similar to other programming languages (e.g. C, Java). The standard reference is [7].
before the definition of test cases. TTCN-3 uses templates to represent these data types. For instance, the messages exchanged during the running of the control software are defined in the following datatype.

```csharp
type record GlobalIO {
    charstring TYPE, // I/O Type
    hexstring IO, // I/O nr.
    hexstring LINKED_IO, // Linked I/O no.
    integer sval, // Data
    boolean DAV, // Data changed
    boolean FAIL, // I/O
    boolean Disabled, // I/O has been disabled
    boolean Cablebreak, // I/O has cablebreak
    charstring TAGNAME // I/O
}
```

and then an instance or object of it is defined as a template:

```csharp
template GlobalIO GlobalIOTemplate := {
    "IO_TYPE_DIG", '8001'H, '010E'H, 0, FALSE, TRUE, TRUE, TRUE, TRUE, "ZS4712B@Eng. local stop switch (alarm cut off)"},
    "IO_TYPE_DIG", '8001'H, '1012'H, 0, FALSE, TRUE, TRUE, TRUE, TRUE, "ZS4712B@Eng. local stop switch (alarm cut off)"
}
```

**Components and Ports.** Having defined the data to be exchanged and the templates for the values, the next is to define the test components. Test components are test processes that are executing independently. A test component is an instance of component type definition. In our case, we identify one major test component; the test harness component. The other component; Main test component (MTC) is automatically created by the test system. The definition of the test harness component is listed below. Communication between test components and the test system interface is achieved via communication ports. Test component types and port types, denoted by the keywords component and port, are defined in the definition part.

```csharp
type component ThComponent {
    port ThPort thp;
    const float T_GUARD_DEFAULT1 := 5.0;
    timer T_GUARD1 := T_GUARD_DEFAULT1;
    var address sut_addr := {
        host1 := PX_SUT_IP_ADDR1,
        portField1 := PX_SUT_PORT1
    };
    var default compDefaultRef1;
}
```

Messages and signatures are used for communication over the communication ports by means of message exchange and procedure calls. Hence, the ports are message-based, procedure-based or mixed (i.e. message- and procedure-based) and they are directional. Each port may have an in-port (for input), out- (for output) or inout- (used for both input and output).

**Test Behavior.** A test behavior in TTCN-3 is defined within functions, altsteps and testcases. Altsteps are function-like descriptions that are used for structuring test component behavior. Altstep has special semantics used to define an ordered set of alternatives. The result of a test case execution is a test verdict. Below is the altstep and part of the test case definition of the Main at function. The responsibility of this function is to read specific data from a data table which signals the start of another cycle.

```
altstep DefaultAltstep1() runs on ThComponent {
    [] thp.receive {
        setverdict(pass);
        stop;
    };
    [] thp.getcall {
        setverdict(fail);
        stop;
    };
    [] thp.getreply {
        setverdict(fail);
        stop;
    };
    [] thp.catch {
        setverdict(fail);
        stop;
    };
    [] T_GUARD1.timeout {
        setverdict(inconc);
        stop;
    };
}
```

```
testcase Main_at() runs on ThComponent system GioInterface {
    preambleSetup1();
    log("--- sending parameters");
    thp.send(triggerMsg) to sut_addr;
    thp.receive(StartRunTemplate) from sut_addr;
    setverdict(pass);
    postambleRelease1();
}
```

**Test Configuration.** TTCN-3 allows the specification of concurrent test configurations. A configuration consists of a set of inter-connected test components with well-defined communication ports and an explicit test system interface which defines the borders of the test system. See Figure 4.

### 3. DESIGN OF A TEST HARNESS

This section illustrates the detailed design of a test harness and the different components involved in the procedure.

The general structure of a TTCN-3 Test System is depicted in Figure 5. A TTCN-3 Test System is made up of a set of interacting entities which manage the test execution: interpret or execute the TTCN-3 code. They realize the communication with the SUT, implement external functions and handle timer operations. The following are the definitions of these entities as found in the standard documents [8], [9].
TTCN-3 Executable (TE) interprets or executes the compiled TTCN-3 code. This component manages different entities: control, behavior, component, type, value and queues, entities which are the basic constructors for the executable code.

- Component Handler (CH) handles the communication between test components. The CH API contains operations to create, start, stop test components, establish connections between test components (map, connect), handle the communication operations (send, receive, call and reply) and manages the test case verdicts (result of tests). The information about the created components and their physical locations is stored in a repository within the execution environment.

- Test Management (TM) manages the test execution. It implements operations to execute tests, provide and set module parameters and external constants. The test logging is also realized by this component.

- Coding/Decoding (CD) encodes and decodes types and values. The TTCN-3 values are encoded into bitstrings which are sent to the SUT. The received data is decoded back into the TTCN-3 values.

- System Adapter realizes the communication with the SUT. The communication operations; send, receive, call, getcall, reply, used to interact with the SUT, are defined and implemented by the System Adapter.

- Platform Adapter implements the timers and the external functions. Timers are platform specific elements and have to be implemented outside the test system. The Platform Adapter provides operations in order to handle timers; create, start, and stop. External functions (whose signature is specified in the TTCN-3 specification) are implemented also in the Platform Adapter [9].

We composed the various heterogeneous components for the development of a test harness for a control software based on Object Oriented Design [11]. Our architecture emphasized the system’s overall software components as consisting of three basic components; viz. User interface component, the TTCN-3 tester component and the control software component. The TTCN-3 tester component is responsible for providing all the functionality for testing the control software. Interaction with the test engineer is provided through the Graphical User Interface (GUI) as a set of operations which are connected to the control software component.

Since our major concern is a test harness for the engine control software, we had to implement some components of the test system, the user interface and link it to the legacy control software. The major subcomponents of these components are listed below.

1. GUI Component: The GUI component is responsible for reading and setting buttons, updating displays and selecting test cases that let the user interact with the system. This component is implemented using Java.

2. TTCN-3 Component: TTCN-3 component is responsible for providing all the functionality for testing the control software. Functionality is provided through the GUI as a set of operations. TTCN-3 component has EcuAdapter with its classes described below:

   The EcuAdapter is provided for the communication between TTCN-3 modules and the System Under Test. The classes belonging to the EcuAdapter are EcuSA, CD_ecu, SocketListener, Utilities and Main as highlighted in Figure 6. The functionality of these classes are:

   - **ecuSA**: This class implements the main function of the EcuAdapter.

   - **CD_ecu**: This class implements the encoding and decoding functions needed to convert the TTCN-3 value representations into a bitstring and vice versa. The two functions that provide these facilities are tciEncode and tciDecode.

   - **SocketListener**: Implements a UDP port listener function which runs an infinite loop listening for exchange of data in the form of UDP packets. The runSocketListener provides this functionality.

   - **Utilities**: This implements the functions required for sending, receiving and synchronization processes.
of data along with binding function for the socket. It also provides additional functions required by the CD_ecu for data conversion.

3. ECSoftware Component: The ECSoftware component contains two major components; the GlobalIO and the control software ECU functions (Main at being of main focus). This is the system that is being tested and it is already fully developed.

In order to link the control software, we need the MainSut, SocketListener and ResponseGenerator which serve the following purposes:

- MainSut: This wraps the functions and the global data table where the data to exchanged are defined.
- SocketListener: Implements the UDP listener function similar to the ECU Adapter’s Socketlistener for the control software (SUT).
- ResponseGenerator: Implements the functionality to respond on receipt of request message from the test script via ECU adapter.

3.1 Heterogeneous Components

We wrote the test case using TTCN-3 by specifying the behavior of the Main at function of the control software. This test case is compiled and executed using the OpenTTCN [4] tool. We implemented the system under test adapter that connects the TTCN-3 tool to the control software. This is done in Microsoft Visual C++ 6.0 - the source language. The graphical user interface is implemented in Java and it connects the TTCN-3 test driver and the control software under test. See Figure 7.

Java is used because it gives us a higher productivity and easier adaptation of the GUI. Furthermore, the GUI does not need to satisfy demanding performance constraints.

3.2 Operation

A test case is executed by first starting the test system, selecting the test case to execute and executing it. The control software function is started immediately the test system is started. All these are done through the graphical user interface. As test cases are executed, the results are displayed while at the same time logging test execution information to a specified log file.

3.3 Results

The procedure described has been shown to be effective as the correct test verdict was generated from the executed test case. However, additional empirical data for developing test scripts would be useful to gain more confidence in our procedure. This effort is ongoing.

4. CONCLUSION

We have implemented a test harness which incorporates a TTCN-3 test driver with the legacy software, and a graphical user interface that allows engineers to run the tests. These include, a TTCN-3 test driver in C++, the legacy control software in C and a Graphical User Interface (GUI) and connectors linking these components in Java. We implemented the SUT adapter that connects TTCN-3 tool to the control software and wrote some test cases using TTCN-3.

We consider the approach and the design relevant for
many legacy control softwares. A point in favour of the approach is that TTCN-3 is platform independent and mature. This is evident in the testcase that we executed through the test harness. The test case is now documented rather than being an internalized knowledge of the test engineer.

This feature is of particular interest to the company engineers. Indeed, the possibility of evolving the prototype into a production quality test harness is currently being considered. For this to be realized, the test engineers must write up test cases for all the functionalities to get a complete test suite for the control software. Clearly, additional domain knowledge about the operations of a marine diesel engine is required. It may also require a more abstract model of the control software, as presented in the systematic approach to testing embedded and real-time systems in [12].

Currently a new project with a student intern is developing systematic TTCN-3 test scripts for the engine control.

We also consider the experience reported in this paper to be useful to engineers interested in using TTCN-3 for testing legacy systems and designing test harness for new generations of embedded system.

A challenge during the implementation was how to easily integrate system consisting of heterogeneous components. For instance, we had to write Java Native Interface code to interface with C++ functions. Obviously, this requires the knowledge of the different languages and how to link them to achieve the desired integration.

A possible alternative could be automatic code generation where the knowledge of the language of the native codes is not a prerequisite. The different language elements can be modeled and codes can be automatically generated. A UML tool that supports mixture of heterogeneous components would be of help in this context.

5. ACKNOWLEDGMENTS

Many thanks to Adegboyega Ojo for his continued support and helpful comments. We are very grateful to Jean Laustsen of Alpha Shipmate A/S for posing the problem and all the time he has spent explaining it to us.

6. REFERENCES