A MODEL DRIVEN TESTING FRAMEWORK FOR SCA APPLICATIONS

Francis Bordeleau (Zeligsoft, Gatineau, QC, Canada; francis@zeligsoft.com).
Toby McClean (Zeligsoft, Gatineau, QC, Canada; toby@zeligsoft.com);

ABSTRACT

One of the key objectives of the Software Communication Architecture (SCA) is the decoupling of applications (waveforms) and platforms, which aims to enable the porting of waveforms from one radio platform to another. The work done in the last decade has resulted in the development of technologies and commercial products that have significantly reduced the complexity and risk associated with the development of SCA-compliant. However, in spite of the fact that testing accounts for a large percentage of the development effort in embedded systems and that the SCA introduces new testing challenges, the key aspects of SCA application testing have not been seriously addressed yet by the industry.

In this paper, we present a model-based testing framework that focuses on the testing of SCA applications. The proposed framework adopts a scenario-based approach to testing. A key aspect of the SCA testing framework is the grouping of components and applications with their associated test cases. These test cases can be used to test components and applications in different platform configurations and on different platforms. In this context, test cases constitute reusable assets that ensure consistent testing in all deployment contexts in which an application (or a component) is used. This type of packaging offers many key advantages that allow for reducing the risk and cost associated with component and application portability.

1. INTRODUCTION

The Software Communication Architecture (SCA) has been developed by a consortium of US companies under the sponsorship of the US DoD to serve as a standard for the development of interoperable Software Defined Radios (SDRs). One of the key objectives of the SCA is the decoupling of applications (waveforms) and platforms, which aims to enable the porting of waveforms from one radio platform to another. This decoupling is achieved by the introduction of a middleware layer composed of a POSIX Real-Time Operating System (RTOS), a CORBA ORB and an SCA Core Framework (CF), which is responsible for the deployment and configuration of applications on the platform and for the management of the applications running on the radio and of the radio platform itself.

The work done by defense contractors and system integrators in the US and around the world, and by commercial vendors involved in the SCA market, has resulted in the development of technologies and commercial products that have significantly reduced the complexity and risk associated with the development of SCA-compliant products (waveform applications and radio platforms). In particular, commercial SCA development tools and Core Frameworks are now available and have proven their value.

However, in spite of the fact that testing accounts for a large percentage of the development effort in embedded systems and that the SCA introduces new testing challenges, the key aspects of SDR testing have not been seriously addressed yet by the SDR industry. While SCA compliance testing of Core Frameworks and SDR platform is covered by the JTRS Testing Application (JTAP) the testing of SCA applications has only been superficially addressed, primarily through the introduction of basic capability to test the SCA interfaces of application components in commercial SCA development tools such as Zeligsoft CE™. No real efforts have been invested in the development of solutions to support the testing of waveform applications with respect to their behavior and quality of service (QoS).

From an application testing perspective, the component-based software nature of the SCA specification and the platform portability objective of SCA introduce interesting challenges:

- Testing an application in the context of different deployments on a given platform
- Testing an application on different platforms
- Testing an application deployed using different component implementations
- Testing different radio application configurations

To ensure waveform application portability and product interoperability, a standard compliance test suite can be developed and test suites can be associated with applications and packaged/delivered together. This way, application test suites can be reused in a standard way to ensure proper application testing from host to target, in different platform configurations and on different platforms. The development and reuse of such application test suites can lead to significant savings in cost and effort associated with the application porting effort. The development of a standard compliance test suite is particularly important in large programs such as JTRS, ESSOR and WINTSEC, in order to ensure product quality and interoperability.

Model-based techniques are now broadly used in the SDR industry to develop applications and systems. The use of models allows for significantly improved productivity by enabling developers to work at a higher-level of abstraction, and by leveraging generation techniques to produce optimized code and complete sets of descriptors files. The use of models enables the use of model validation techniques to discover errors early in development cycles, and supports model transformation techniques to integrate complementary tools in an overall development process where different models are used for developing different aspects of the system.

Model-based techniques can also be used to support the testing aspect of the development process. Different models, like UML Sequence Diagram (SD), State Machine (SM) and Activity Diagram (AD), can be used to specify different testing aspects. These diagrams can be augmented with annotations, textual descriptions, tabular notations and code for completeness. Modeling techniques can also be used to capture execution sequences to facilitate execution monitoring and diagnosis (i.e. comparison of execution traces with test specification).

In this paper, we present a model-based testing framework for the SCA. The proposed framework adopts a scenario-based approach to testing. In particular, we show how by including scenario descriptions using UML Sequence Diagrams in the SCA SDR model we are able to derive scenario based test suites. The scenarios can describe unit tests for a particular component in the waveform, or they can be used to capture scenarios at the system level. Furthermore, we are able to build the system testing incrementally by growing subsets of the components (and stubs), starting with sub-assemblies of a few components and stubs, and finishing with the complete set of components belonging to the waveform. Since these models are independent of the platform, we are able to start testing on a host platform and systematically migrate to the target platform.

The rest of the paper is structured as follows. Section 2 introduces the key objectives and characteristics of the SCA Testing Framework. Section 3 focuses on the test creation phase. It discusses the main issues that must be addressed and the solution to these issues. Section 4 describes test execution and result analysis. Section 5 discusses the integration of QoS in the testing framework. Finally, section 6 provides a summary of the paper and highlights key benefits of the testing framework.

2. SCA APPLICATION TESTING FRAMEWORK

The main objectives of the proposed SCA testing framework are:

- Support the development of SCA application test suites that are independent of specific radio platforms and that can be used (and reused) to test components and applications on different platforms and in different platform configurations
- Be applicable at different levels of component-based application testing, from component (unit), to application (component assembly), to deployed application
- Support the creation and execution of SCA application test cases and the analysis of test results
- Provide support for the introduction of the QoS aspect throughout the testing process

The SCA testing framework we propose leverages model-based techniques to define reusable test cases independently of specific programming languages, and it uses a scenario-based testing approach to focus on collaboration/interaction between application components. The framework builds on the important body of technical material developed over the last decades around modeling notations and scenario-based techniques, like UML Sequence Diagrams and Message Sequence Charts (MSC).

A key aspect of the SCA testing framework is the grouping of components and applications with their associated test cases. As illustrated in Figure 1, these test cases, which are packaged and delivered with the components and applications, can be used to test components and applications in different platform configurations and on different platforms. In this context, test cases constitute reusable assets that ensure consistent testing in all deployment contexts in which an application (or a component) is used. This type of packaging offers many key advantages that allow for reducing the risk and cost associated with component and application portability.

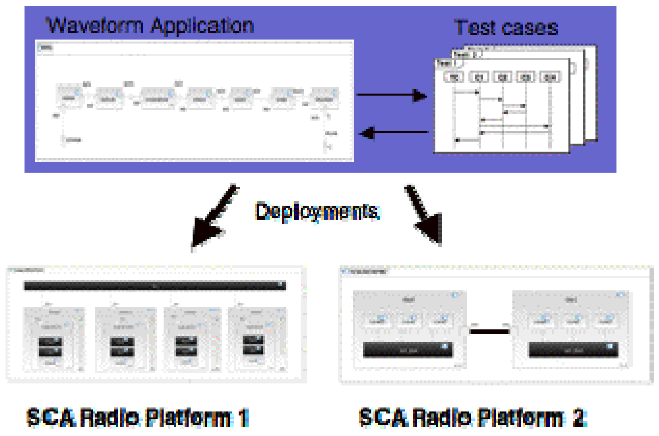

Figure 1 Packaging of waveform applications and test cases

The proposed testing framework can be used to develop standard compliance test suites for waveform applications that need to be ported on a set of different platforms. This approach has the potential to provide major benefits to large programs like JTRS, ESSOR and WINTSEC —ensuring waveform portability and product quality and interoperability.

As an example, ETSI has developed a conformance test suite for the TETRA waveform application. Similar test suites could be developed for SCA waveforms.

3. TEST CREATION

The objective of the test creation phase is to develop the test suite required to test a component or an application in an incremental and iterative fashion. The test suite is composed of a set of scenario test cases. Each scenario describes a specific sequence of actions that need to be tested. The test suite must contain both positive and negative scenarios, which respectively represent scenarios that the system must be able to properly execute and scenarios that the system must reject.

SCA applications and platforms can be modeled as a set of interacting components using UML diagrams. Model analysis techniques can then be used to validate the correctness of models and model transformation techniques can then be used to automatically generate code. Zeligsoft CE is an example of a commercial tool that supports the development of SCA applications and platforms. In a similar way, modeling techniques can be used to specify and validate test scenarios, and automatically generate necessary test artifacts.

Figure 2 illustrates the SCA test creation process. First, the test scenarios are specified by the test designer. Scenarios are specified in terms of pre and post conditions, triggering and resulting events (or messages), sequence of messages, and constraints. The proposed framework uses UML Sequence Diagrams (SD) as a basis for scenario description. SDs are used to describe the collaboration between the components of an application in the context of specific scenarios

Once specified, test scenarios can be validated for consistency. Then, model transformation techniques are used to automatically generate the set of test cases associated with the scenarios, as well as the test drivers (responsible for executing the different test cases) and the set of SCA descriptor files (required to load and execute the test cases).

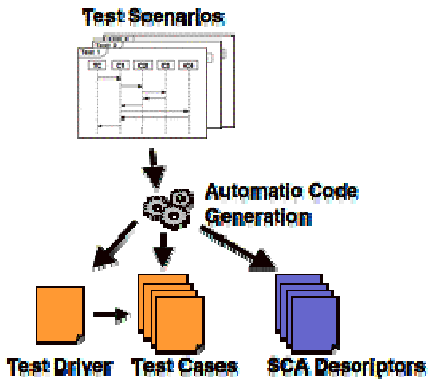

Figure 2 Test Creation Process

The key issues that must be addressed by the overall testing framework include the following:

- Scalability of Sequence Diagram to specify large and complex scenario sets
- Handling of data associated with scenarios
- Specification of scenario setup and cleanup

Scalability of Sequence Diagram (SD) to specify large and complex scenario sets

Because embedded systems like SCA systems are associated with large sets of complex scenarios, scenario description notation must explicitly deal with scalability issues. Those issues relate to two different scenario specification aspects: description of individual scenarios and description of scenario sets.

The description of individual scenario involves dealing with long end-to-end communication sequences, scenario alternatives and exception cases. To facilitate the design and maintenance of such scenarios, one solution consists in decomposing large end-to-end scenarios into sub-scenarios. Sub-scenarios can then be composed together in larger scenario and reused in different contexts. Thus, to support the description of complex scenarios, a scenario description notation must provide a mechanism for hierarchical composition of scenarios (or sub-scenarios) and a set of basic control operators such as sequencing, looping, parallel composition, and conditional branching. Also, additional concepts such as timers and interrupts are required in the context ofreal-time systems.

The UML 2.0 specification [3] allows addressing the scalability issues by combining providing the required concepts and composition operators in Sequence Diagrams. Also, large scenarios composed of sub-scenarios can be described using Interaction Overview Diagrams.

The description of scenario sets involves dealing with large number of scenarios. To facilitate the design and maintenance of such scenario sets, scenarios can be grouped into subsets of related scenarios and scenario relationships can be explicitly captured using graphical representations. Important scenario relationships include sequential composition, trigger/triggered-by, conditional branching, mutual exclusion, parallel composition, and preemption.

To deal with the description of scenario sets, conventional UML notation can be used. Related scenarios can be grouped together in packages and scenario relationships can be captured in class diagrams where individual scenarios are represented by a stereotyped classifier and inter-scenario relationships are represented by stereotyped associations.

Handling of data associated with scenarios

The data issues relate to two different aspects: the specification of data in scenario descriptions and the creation of data sets to drive the execution of scenarios.

The first aspect relates to the need to augment basic message-based scenario description with constraints at different levels, including constraints on the values of parameters in messages, on looping and alternative conditions. To address this aspect, SDs are annotated with constraints.

The second aspect relates to the need to control the configuration of components involved in a scenario, as well as scenario parameters and initial message parameter values. There are two aspects of data specification required: data plane and control plane.

Specification of the test data for the data plane is typically done with a data source component that reads a file with the stream of data to be processed by the application. Thus, the specification of the data for the data plane is one or more files that are loaded by the data source component.

The specification of data for the control plane will be specific values for the scenario parameters. A test case would then comprise specific values for each of the scenario parameters. The challenge with the specification of data for the control plane is the number of possible combinations that arise. One approach for dealing with this combinatorial explosion is to use a technique called covering arrays. Covering arrays is a t-wise testing technique of systems whereby each scenario parameter is associated with a domain of values, and the covering array provides a mechanism for automatically deriving a minimal set of test cases for the t-wise interaction of parameters. For parameters whose domain is not easily enumerable, such as integer parameters for example, equivalence partitioning can be used to define the set of values for the parameter. If there are multiple data source files that are to be tested then the data source becomes another parameter in the covering array (whose domain is the set of files).

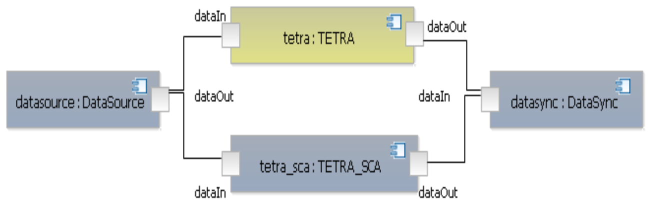

Figure 3 Testing of TETRA waveform

As illustrated in Figure 3, the verification of the data plane can be done by using a golden waveform. The golden waveform may be defined in a Simulink model or as a C implementation of the data processing algorithm. When the stream of data is fed to the application under test in a scenario it is also fed to the golden waveform. The resulting streams of data coming from the application under test and the golden waveform are then compared to ensure that the application is correctly processing the data. This testing of the data plane is done in parallel with the execution of the scenarios for the control plane.

Specification of scenario setup and cleanup

A key issue in the development of a scenario-based testing framework (approach) relates to the specification of the setup and cleanup scenarios that are associated with every test scenario. The setup scenario is required to bring the system to its precondition state for the execution of the scenario to be tested. The setup scenario is responsible for initializing all of the components in the application so that they satisfy the pre-conditions of the scenario. Conversely, the cleanup scenario is responsible for releasing any resources acquired by the scenario.

4. TEST EXECUTION AND RESULT ANALYSIS

One of the main benefits of the proposed testing framework is the reusability of the test cases in the context of different system configurations, i.e. systems configured with applications deployed on different platform devices or systems configured with different sets of applications.

The proposed testing framework provides for the specification of application deployment. Because test scenarios specified independently of any deployment, they do not need to be modified for specific deployments. Code generation from test scenario models takes care of generating code that is specific to a specific deployment.

The same test cases can be executed in different deployment on a platform, and test results may be compared with respect to QoS.

The test execution and evaluation process is illustrated in Figure 4. First, the waveform application to be tested is deployed on the target platform. Then, application test cases are executed. The result of test case execution (identified as Test Output in Figure 4) is compared with the test case specification (identified as Test Input in Figure 4), and an execution evaluation is produced.

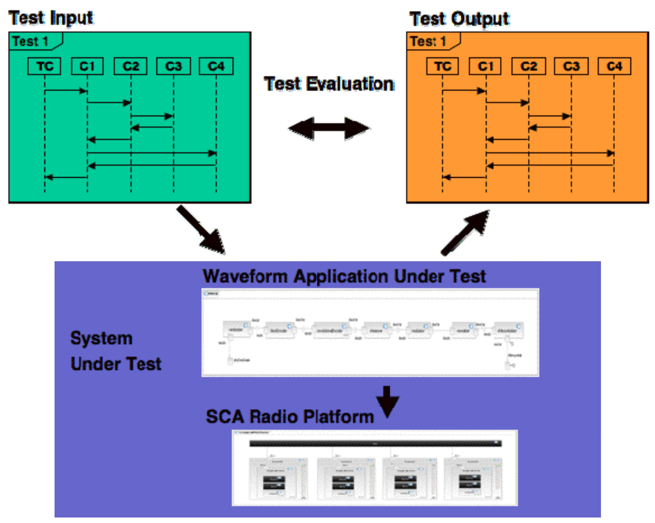

Figure 4 Test Execution and Evaluation

To capture and monitor test execution, component code must be properly instrumented. The current SCA specification doesn't provide all the necessary interfaces to support application testing and monitoring. Using model-based techniques and proper code generation technique, code instrumentation can be automated and optimized to minimize its impact on code execution. In particular, component ports must be instrumented to allow the send and receive messages exchanged between components to be captured.

5. INCLUDING QOS IN TEST MODELS

In order to properly test applications, it is insufficient to test only the functional aspect of the applications. QoS must also be tested. For this reason, it is necessary to introduce the QoS aspect in the SCA testing framework.

To address QoS, key concepts of the MARTE Profile for UML [2] can be used. By adding QoS property annotations and scenarios to the traditional model of an SCA SDR, one is able to generate a more complex testing infrastructure. Combining these additions with the separation between waveform and platform inherent in the SCA, it is possible to test early and test often. Automating the generation, execution and analysis of these tests — also known as Model Driven Testing (MDT) — enables agility for SCA SDR developer.

Using the QoS property annotations in the scenarios and SCA waveform and platform, we are able to integrate system aspects into the generated tests and test suites. For example, the latency of a connection between two devices or nodes in the platform is annotated in the model. This latency annotation can be used to generate “delays” in the behavior of a generated stub. In the paper we explore how other traditional non-functional property annotations can be used to generate more complex stubs in the tests and test suites.

6. SUMMARY

In this paper, we presented a model-based testing framework for SCA applications. The main objectives of the proposed SCA testing framework are:

- Support the development of SCA application test suites that are independent of specific radio platforms and that can be used (and reused) to test components and applications on different platforms and in different platform configurations
- Be applicable at different levels of component-based application testing, from component (unit), to application (component assembly), to deployed application
- Support the creation and execution of SCA application test cases and the analysis of test results
- Provide support for the introduction of the QoS aspect throughout the testing process

The resulting testing framework adopts a scenario-based testing approach and uses UML notation at its core. One of its key aspects is the grouping of components and applications with their associated test cases, which can be used to test components and applications in different platform configurations and on different platforms. This approach results in the creation of reusable assets that ensure consistent testing in all deployment contexts in which an application (or a component) is used.

The proposed testing framework can be used as to develop standard compliance test suites for waveform applications that need to be ported on a set of different platforms. This approach has the potential to provide major benefits to large programs like JTRS, ESSOR and WINTSEC —ensuring waveform portability and product quality and interoperability.

10. REFERENCES

[1] Object Management Group (OMG). UML Testing Profile , Version 1.0 (formal/2005-07-07). 2005.

[2] Object Management Group (OMG). A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, Version1.0 (ptc/2008-05-23). 2008.

[3] Object Management Group (OMG). Unified ModelingSuperstructure, Version 2.1.2 (formal/2007-11-02). 2007.