Software Architecture Testing for Wireless Network Applications in Component Based Real-Time Embedded Communication System

A proposal for the degree of Ph.D. at George Mason University

By

Jing Guan

Master of Engineering, Electrical and Computer Engineering
University of Maryland at College Park, MD, 1999

Bachelor of Science, Electrical Engineering
Northwestern Polytechnical University, China, 1997

School of Information Technology & Engineering

Spring Semester 2011
George Mason University
Fairfax, VA

Committee:

_______________________________    Jeff Offutt, Dissertation Director

_______________________________    Jeremy E. Allnutt

_______________________________    Hassan Gomaa, Chair, Department of Computer Science

_______________________________    Daniel A. Menasce, Senior Associate Dean,
School of Information Technology & Engineering
# TABLE OF CONTENT

1. Problem Statement .......................................................................................................................... 3
   1.1 The Statement ............................................................................................................................ 4
2. Background ....................................................................................................................................... 5
   2.1 Wireless Radio Technology ......................................................................................................... 5
   2.2 SDR ............................................................................................................................................... 8
   2.3 Embedded Software Testing ........................................................................................................ 10
3. Issues and Related Work .................................................................................................................. 11
   3.1 Issues in wireless embedded systems testing ............................................................................. 11
   3.2 Related Work ............................................................................................................................... 12
      3.2.1 Component-Based Embedded Software Testing ................................................................. 12
      3.2.2 Software Communications Architecture ............................................................................. 13
      3.2.4 Wireless Network Protocol Analysis & Testing ................................................................. 21
4. Proposed Research Approach ......................................................................................................... 22
   4.1 General Properties to Be Tested at the Software Communication Architecture Level ................. 24
   4.2 Software Communications Architecture Key Requirements ..................................................... 26
   4.3 Graphical Representations of Architectures ............................................................................... 27
      4.3.1 Component Port Interaction Graph ..................................................................................... 27
      4.3.2 Component Port Behavior Graph ....................................................................................... 29
   4.4 Test Criteria .................................................................................................................................. 33
   4.5 Test Case Generation ................................................................................................................... 33
   4.6 Validation Method and Prototype Tool ....................................................................................... 35
      4.6.1 Experiment Design .............................................................................................................. 35
5. Unique Contributions ....................................................................................................................... 39
6. Research Schedule ......................................................................................................................... 39
7. Planed Papers .................................................................................................................................. 40
   Papers from this researched will be prepared and submitted for ICST, Milcom, and TSE .......... 40
8. References ......................................................................................................................................... 40
LIST OF FIGURES

Figure 1: Analog and Digital Radio .............................................................................................................. 6
Figure 2: SCA software architecture detailed view .................................................................................... 14
Figure 3: CF interfaces in UML ........................................................................................................................ 15
Figure 4 shows the relationship between the OS, the application and the OE .............................................. 16
Figure 5: Relationship between SCA components ......................................................................................... 16
Figure 6: Stubs and Skeletons in CORBA ........................................................................................................ 18
Figure 7: Connection of components via Ports ............................................................................................... 20
Figure 8 illustrates these functions as components in SCA context .............................................................. 21
Figure 9: Network services in SDR architecture .............................................................................................. 21
Figure 10: SCA components interaction .......................................................................................................... 25
Figure 11: Connection of components via Ports .............................................................................................. 28
Figure 12: A CPIG for multiple components .................................................................................................... 29
Figure 13: Two components in a CPBG ............................................................................................................... 30
Figure 14: A CPBG Example .............................................................................................................................. 31
Figure 15: Waveform Components IP Packet Path .......................................................................................... 34
Figure 16: The Prototype Tool .......................................................................................................................... 38

LIST OF TABLE

Table 1: Testing Requirements and Graphs .................................................................................................... 33
Table 2: Tentative Schedule .............................................................................................................................. 39
1. Problem Statement

The increased complexity of embedded real-time systems leads to increasing demands with respect to software engineering, which increases the importance of an efficient management of emergent properties such as maintainability, portability and testability. Due to complexity, the development of such systems is very expensive. To ease the problem, component-based modeling is getting more attention in the embedded systems domain [10]. Component software technology for maximizing software reuse helps to overcome the extreme complexity of embedded software and reduce development cost. Software Defined Radio (SDR) developed with Software Communications Architecture (SCA) is an example of such an effort.

SDR refers to reconfigurable or reprogrammable radios that can have different functionality with the same hardware. Because the functionality is defined in software, a new technology can easily be implemented on a radio by updating the software running on it. So a radio can be built to meet the need for continuously changing technology. In an SDR, multiple waveforms can be implemented in software using the same hardware. One software defined radio can communicate with many different radios, with only a change in software parameters. This increases interoperability among different military units, emergency units, and coalition forces. Also new technologies can be adapted quickly, easily, and with a much lower cost.

SCA is a common, well-defined open architecture. It is used to build radios that support operations in a wide variety of domains without losing the ability to communicate with each other. It can help radio vendors improve interoperability by providing the ability to share waveform software between radios, and reduce development time through software reuse. This architecture also facilitates scalability and technology insertion.

As an emerging technology in embedded software development, the SCA presents a new paradigm, and it affects the entire embedded software development cycle including analysis, specification, design, implementation, verification, validation and maintenance. However, it introduces a new environment filled with system software concepts to embedded communication engineers, such as object oriented programming, portable operating system interface, and middleware using Common Object Request Broker Architecture (CORBA). The more complex the system is, the more difficult it is to understand its behavior. As a result, it is harder to verify the system’s correctness [11]. How to increase observability and controllability in testing this type of real-time embedded system presents a number of practical difficulties. There is an urgent need to address this issue in SDR development.
Evaluating and testing SDR at the architecture level will not only allow tests to be created in a systematic way so as to increase the confidence level for such a complex system, but also make it possible to get desired test support functionality included into the system design, therefore substantially reducing the costs of any problems and errors. Currently, there is a lack of integration testing techniques for SDR software at the architecture level. The proposed research is in the area of SCA-based testing to generalize testing techniques at this level.

1.1 The Statement

Testability for real-time embedded software is always a crucial problem due to the low observability and controllability of the embedded systems. Gaining better control of controllability and observability over SCA-based embedded communications software is critical to the overall quality of software defined radio systems. Errors may severely impact the software in ways that are costly to fix and causes delays and failures when deploying systems in the real world. Currently, there is a lack of formal testing methods for testing SCA-based SDR. The few research techniques that exist are either limited in scope or known to depend heavily on simulation methods and test equipment.

The objective of this research is to provide a new architecture-based testing approach to help SDR testing become more controllable and observable. So it will help embedded communications engineers have more confidence about SCA-based embedded software.

Problem Statement:

SCA-based SDR development brings new challenges to traditional embedded communication engineering. In particular, the current method of testing architecture-based software defined radio is inadequate. We need stronger techniques to test this type of component-based embedded communication systems.

Traditional testing techniques do not work effectively for SDR since they cannot detect many of the faults made by the use of SCA architectures in development. On the other hand, the existing SCA architecture compliant testing techniques only focus on architecture related interfaces rather than SDR component interaction analysis.

To solve the problem, we need to consider the unique characteristics of this type of system, which combines new software concepts with advanced radio technology in complex real-time embedded systems. The proposed research project will address the problem of effectively testing SCA-based software defined radio software.
Thesis Statement:
This thesis will address the problem of testing SCA-based SDR by inventing formal test criteria for software architecture level integration testing in component-based embedded communication systems. The research will include techniques to design test values and automate the tests with test scripts while increasing controllability and observability of wireless network applications in the system.

2. Background

This section gives basic definitions and background information on wireless radio technology, software defined radio and software testing.

2.1 Wireless Radio Technology

Wireless technology has changed the way the world communicates. Wireless devices like cellular phones, Bluetooth headsets, and GPS systems have been seen everywhere around us. Wi-Fi Internet is now common in homes, restaurants and businesses. Wireless technology has undergone rapid innovation in providing solutions for fast, easy and inexpensive information access.

In wireless devices, radio is used to exchange information. The information is carried by radio waves. They are electrical and magnetic fields that travel through space in the form of a wave. Waves have two important characteristics that can change. One is the strength of the wave, also called amplitude. The other is frequency, which determines how often the wave occurs at any point. Electromagnetic waves that have a certain frequencies are called radio waves.

A communication system sends information from one place to another. This type of system is formed when the information is sent on a radio wave in such a way that it can be recovered at the other end. The process of impressing the information is known as modulation. In order to modulate a radio wave, either or both of the two basic characteristics of the wave need to be changed: the amplitude and the frequency. Amplitude modulation (AM) means changing the amplitude of a wave in a way corresponding to the information to send. On the other hand, if changing the frequency of a signal instead of the amplitude, the resulting modulation is called frequency modulation (FM). There are other types of modulation techniques that can be viewed as variations of AM and FM.
Radios

A radio performs a variety of functions in the process of converting voice or data information to and from an RF signal. These functions include processing the analog RF signal, waveform modulation/demodulation and processing of the baseband signal. The processing of the analog RF signal consists of amplification and de-amplification, converting to and from intermediate frequencies (IF), RF up-conversion and down-conversion and noise cancelling. Waveform modulation and demodulation depends on the waveforms used in that radio. This part generally includes error correction and interleaving of the signal. The baseband signal processing part adds the networking protocols and routes the signal to the output devices [1].

Analog radios perform these functions on analog signals. Using analog signal processors and IF filters, the signal is processed at different frequencies by a chain of functional blocks. Digital radios transform the analog signal to a digital signal at some point in the chain with an analog-to-digital convertor (ADC), and process the signal using digital signal processors (DSP). The digital signal is converted back to analog by a digital-to-analog convertor (DAC) and transmitted through the antenna [33].

Figure 1 shows the functional blocks in a dual-mode radio that has both analog and digital processing.

![Figure 1: Analog and Digital Radio](image-url)
Radio forms the basis of all the mobile and portable communications systems we use today.

**Wireless Standards**

Wireless systems follow various standards for radios to communicate with each other. The following are some examples.

Advanced Mobile Phone Service (AMPS) is a first generation (1G) cellular standard. It is an analog standard that allows low power enabling portable operation.

Time Division Multiple Access (TDMA), Code Division Multiple Access (CDMA) and Global System for Mobile Communications (GSM) are the second generation (2G) standards. TDMA divides a single channel into a number of time slots, with each user getting one out of every few slots. CDMA is a spread spectrum technology, which means spreading the information contained in a particular signal of interest over a much greater bandwidth than the original signal. Because of the wide bandwidth of a spread spectrum signal, it is very difficult to jam, interfere with or identify. GSM is the world’s leading and fastest growing mobile standard. The signal coding is very similar to TDMA. User authentication is provided by Subscriber Identity Module (SIM) cards. By inserting the SIM card into any GSM phone, the user is able to receive calls at that phone, make calls from that phone or receive other subscribed services [33].

The third generation (3G) standards include Wideband Code Division Multiple Access (WCDMA), Enhanced Data rates for GSM Evolution (EDGE) and Universal Mobile Telecommunications System (UMTS). WCDMA spreads user information bits over an artificially broadened bandwidth, by multiplying them with a pseudo-random bit stream running several times as fast. EDGE is developed to deliver further enhancements in data capability over the core GSM. This will achieve the delivery of advanced mobile services such as the downloading of video and music clips, full multimedia messaging, high-speed color internet access and e-mail on the move. UMTS is one of the major new 3G mobile communications systems being developed within the framework defined by the International Telecommunication Union (ITU). UMTS builds on the capability of current mobile technologies by providing increased capacity, data capability and a greater range of services using an innovative radio access scheme and an enhanced, evolving core network.

The list of wireless and communications standards continues to grow at an unprecedented pace – Bluetooth, WiMAX, cdma2000, ZigBee, and RFID. All of the above systems require different sets of terminals and base stations, and they are incompatible with each other. It is almost impossible for mobile
hardware suppliers to offer end users the flexibility to support all of the standards without integrating so many circuits into a unit that it becomes unwieldy and power hungry.

One way to keep step with wireless and communications advances is through software. This is why SDR is drawing more attention from communications engineers. Traditional radios use hardwired analogue circuits to interpret signals. The circuits enable the radio to operate in a specific range with a specific modulation. SDR moves large parts of the analogue circuit into software, which makes the radio programmable and able to operate in different modulation types used by mobile network standards.

An SDR terminal can use any of the above technologies if it has the required software. If a new standard is developed, or if the user wants to switch to a system that is not present on the handset, the new software can be downloaded remotely and wirelessly. SDR base stations can dynamically update themselves rather than system providers replacing the base station throughout the entire country. For example, a Smartphone using software radio would theoretically be able to jump between the different types of modulation used by Wi-Fi, Wimax and 3G. Future radios with advanced software programmable encoding and modulation will bring wireless networks unprecedented flexibility and performance.

The flexibility with SDRs is a key to the future of wireless communication systems. Prior generations of wireless devices relied on highly customized, application-specific hardware with little emphasis on adaptation to new standards. This design approach generally yielded power- and performance-optimized solutions at the expense of flexibility and interoperability. Over the past decade, SDRs have evolved and spawned related technologies such as cognitive radios (CRs), to give the current generation of communication systems flexibility, reusability, and adaptability. Wireless device developers, as well as service providers and end users, can upgrade or reconfigure SDRs and CRs continually as new versions of wireless standards are released. Furthermore, SDR/CR devices can adapt continually to changes in the spectral or network environment, including modulation schemes, channel coding, and bandwidth. They can be used to establish and maintain ad hoc networks.

2.2 SDR

A software defined radio makes the main characteristics of a communication device reconfigurable with software rather than with hardware alterations. The waveform modulation/demodulation functions are defined in software. Software defined technology offers advantages such as improvements or enhancements without altering the radio hardware, terminals that can cope with the unpredictable dynamic characteristics of highly variable wireless links, efficient use of radio spectrum and power, and
The US Government has led SDR development with the American military’s Joint Tactical Radio Systems (JTRS) program [35]. The JTRS program is a key U.S. DoD transformational program to support the U.S. DoD objective for information superiority on the battlefield and to meet current and future war-fighters’ needs. JTRS is required to provide interoperability across all geographical and organizational boundaries and aims to provide a family of digital, programmable, multiband, multimode, modular radios to alleviate communications interoperability problems. JTRS will be able to transmit voice, data, and video while operating in frequency bands from 2 MHz up to 55 GHz, satisfying ground tactical, maritime, airborne and space based communications requirements.

The military is not the only agency that needs interoperability. As numerous agencies, both domestic and international, have responded to various natural and man-made disasters around the world, communications between the different responding groups has often been hindered by the fact that different communications systems rarely work with one another because the frequency and radio protocols are different. SDR provides an ideal solution to this problem. A centrally deployed base station could be used to receive the transmissions of one agency and reformat and rebroadcast them on the frequencies of the other responding agencies. Since the system would be reconfigurable, as new agencies arrive or depart, the SDR can be rapidly changed to support the required services. When the disaster is over, the system can easily be stowed and re-deployed at a later time when required.

Academia is improving SDR technology via research, using SDR as a flexible tool for radio experimentation, and teaching SDR design methodologies to a new generation of communications engineers. GNU Free Software Foundation (FSF) has an on-going open-source SDR project. GNU Radio [45] is a collection of software that, when combined with minimal hardware, allows the construction of radios where the actual waveforms transmitted and received are defined by software. The project development is open to anybody who wants to contribute. The difference between a digital radio and an SDR is that a digital radio is not reconfigurable. Although a digital radio has software running on it, the functionality of the components cannot be changed on air. New technology insertion is not available either.
2.3 Embedded Software Testing

An *embedded system* is a special-purpose computer system built into a larger device. Usually there is no disk drive, keyboard or screen. Broekman and Notenboom [29] define embedded systems as a generic term for a broad range of systems covering, for example, cellular phones, railway signal systems, hearing aids, and missile tracking systems. They specify that all embedded systems have a common feature in that they interact with the real physical world, controlling some specific hardware. The term embedded system can encompass a variety of devices and systems.

These features greatly affect software testability and measurability in embedded systems. The term *testability* in software testing can be considered from various viewpoints [25, 28, 30]. While some consider the architectural viewpoints [31, 32], few describe techniques for more effective design for testability at the architectural level [24, 26]. However, how to increase testability is commonly identified as an important goal in software testing research [27].

In embedded software systems, two main viewpoints of testability are considered from the architectural viewpoint: controllability and observability [28]. To test a component, we must be able to control its input, behavior and internal state. To see how this input has been processed, we must be able to observe the component’s output, behavior and internal states. Finally, the system control mechanisms and observed data must be combined to form meaningful test cases.

Embedded software often generates output for hardware to function rather than for users to interact with, so the observability is significantly low. It is easy to control software input values that are entered from a keyboard. But an embedded program that gets its inputs from hardware is more difficult to control. Testability is an important factor in embedded real-time systems, especially when the systems have additional properties that increase the complexity.

How to effectively test real-time embedded software has been studied in many research papers. A testing method based on the logic coverage criterion has been applied to embedded, real-time control software deployed in a safety-critical application in the transportation industry [7]. This industrial case study compared the normal testing at a company (manual functional testing) with criteria-based testing. Some results showed that the test cases generated to satisfy the coverage criterion detected major safety-critical faults that were not detected by functional testing.

Criteria-based testing is the most technical and mathematical part of software testing. Much of the work involves creating abstract models and manipulating these models to design high quality tests. Ammann and Offutt’s book [2] is organized around four structures and uses them to teach criteria-based test design.
This structure greatly simplifies the testing process, which allows test design to be abstracted and carried out efficiently, and also separates test activities that need different skill sets and knowledge. Because the approach is based on these four abstract models, they call it the Model Driven Test Design process (MDTD) [14].

Test coverage criteria are categorized into four mathematical structures: graphs, logic expressions, input domains, and syntax descriptions. Test criteria on graphs define test requirements to visit or tour specific nodes, edges or subpaths. Graph coverage criteria define four test criteria on graphs. In prime path coverage, tests must tour each prime path in the graph.

One of the most efficient methods to test software testing is to automate test execution. It has been observed that unit and integration testing, if done manually, can be very expensive and time consuming. This is one reason why testing has to be automated. Various testing tools can perform testing in a few hours and produce the testing reports.

3. Issues and Related Work

Issues in SDR empowered wireless communications embedded systems are discussed first in this section, then related work will be discussed.

3.1 Issues in wireless embedded systems testing

Testing SDR empowered wireless communications embedded systems involve multiple complicated issues. First, embedded systems generally have low testability. Second, SCA introduces a new environment to the embedded communications field, filled with system software concepts such as object oriented programming, portable operating system interface, and middleware using Common Object Request Broker Architecture. The more complex the system is, the more difficult it is to understand its behavior [44]. As a result, it is harder to verify the system’s correctness. Informal test methods and manual testing techniques cannot meet the need for software reuse and reconfiguration with open architectures. How to increase observability and controllability in testing this type of real-time embedded systems presents a number of practical difficulties. Third, wireless radio network has complex functionalities, based on a variety of network protocols, including routing, multicast, IP packets forwarding and Quality of Service (QoS). Most research in the area of wireless networking has been conducted using simulation software, which has unrealistic hardware, propagation, interference, and
mobility models [22]. Thus, there is a significant gap between SDR network waveform development and realistic studies of SDR network behavior.

3.2 Related Work

There are a number of areas that are related to the research described in this proposal. The subsections that follow discuss these areas in detail. The first subsection discusses the studies performed to increase testability in component-based communications embedded system. Following this is a detailed description of software architecture designed to facilitate engineering practice in component-based communications embedded system and it covers a formal definition of software communication architecture. Finally, wireless network protocol analysis and testing, the key functionality of communications system under research, is discussed.

3.2.1 Component-Based Embedded Software Testing

Teemu [4] presented a study on design for testability in component-based embedded software based on two large-scale companies in the European telecom industry. He discussed the Design for Testability (DFT) solutions to support test automation from two European telecommunications companies, working on similar large scale component-based embedded systems. Their techniques to support effective test automation were discussed. A common communication protocol provides support for implementing reusable test components. Especially in the case of embedded systems, a good host test environment enables efficient software testing. When this environment matches the target system as much as possible, efficient host testing is possible. One way to support testing is to use an operating system that is supported on both the target hardware and in a host-testing environment, as simulated on a desktop.

Including support for test automation as a first-class feature allows more effective analysis of the system, including analysis of long running tests and deployed systems, and enables efficient field-testing. Effectively implementing this requires possibilities for dynamic configuration of test functionality during execution. Abstracting test cases from the implementation minimizes the effects of internal system changes to the test cases. This mostly applies at the system testing level, as in earlier testing phases it is often necessary to observe more detailed properties of the system.

Teemu’s methods addressed test automation and the different techniques to make this more effective at the architectural level. But they were still limited to regular functional testing to fulfill system requirements instead of designing a formalized and abstract structured testing model.
3.2.2 Software Communications Architecture

SCA is a software architecture specifically designed for communications devices. Many existing architectures are not designed for communication systems, thus they do not provide interfaces that could be directly mapped to radio components.

When the JTRS JPO was established to acquire a family of affordable, high capacity, tactical radio systems that can provide interoperable wireless mobile network services, the need for an open architecture emerged. By building a common open architecture, JTRS can improve interoperability by providing the ability to share waveform software between radios and reduce development and deployment costs. In view of its potential applicability across a wide range of communications domains, JTRS JPO named this architecture the Software Communications Architecture [1].

SCA is applicable across a wide range of communications domains. It supports interoperability through well defined interfaces. The reusability potential for the software components of an SCA implementation is quite high. SCA also supports plug-and-play, which means the design rules have been crafted so that hardware and software modules from different suppliers will work together when plugged into an existing system [5]. The SCA defines the partitioning of functions into groups, which may subsequently be allocated to components. It has been published to meet the following goals [36]:

**Common Open Architecture:** The use of an open, standardized architecture promotes competition, interoperability, technology insertion, quick upgrades, software reuse, and scalability.

**Multiple Domains:** The JTRS family of radios must be able to support operations in a wide variety of domains, including airborne, fixed, maritime, vehicular, dismounted and handheld.

**Multiple Bands:** A JTRS radio can replace a number of radios that use a wide range of frequencies, and it can interoperate with them.

**Compatibility:** JTRS radios must be able to communicate with legacy systems to minimize the impact of platform integration.

**Upgrades:** The JTRS architecture must enable technology insertion, so that new technologies can be incorporated to improve performance, and to build radios that meet the need for continuously changing technology.
**Security:** Security is a very important aspect of military radios. The architecture should provide the foundation to solve issues like programmable cryptographic capability, certificate management, user identification and authentication, key management, and multiple independent levels of classification.

**Networking:** The JTRS radios should support legacy network protocols, for the purpose of seamless integration. The architecture should also support wideband networking capabilities for voice, data and video.

**Software Reusability:** As with any other software architecture, the JTRS architecture should allow the maximum possible reuse of software components.

The SCA structure is composed of an application layer and an operation environment (OE) layer. The software architecture detailed view is shown in Figure 2. The OE consists of a Core Framework (CF), a CORBA middleware and a POSIX-based Operating System (OS). Since the SCA uses the CORBA middleware, application programs are basically composed of CORBA objects that conform to the SCA core framework. The SCA core framework is composed of the specification of interfaces and a domain profile. A domain profile is composed of XML descriptor files that describe the hardware and software configuration information of a SCA system domain.

![Figure 2: SCA software architecture detailed view](image)

```plaintext
Application component    Application component    Application component    Application component

Core Framework IDL ("Logical Software Bus" via CORBA)

CORBA ORB & Services (Middleware)    CF Services & Application

Operating System

Network Stacks & Serial Interface Services

Board Support Package (Bus Layer)

Hardware Bus
```
3.2.2.1 Core Framework

The OE specifies the services and interfaces that the applications use from the environment. The interfaces are defined by using the CORBA Interface Definition Language (IDL), and graphical representations are made by using UML [19]. The CF describes the interfaces, their purposes and their operations. It provides an abstraction of the underlying software and hardware layers for software application developers. An SCA compatible system must implement these interfaces. The interfaces are grouped as Base Application Interfaces, Framework Control Interfaces and Framework Services Interfaces. Figure 3 shows the relationships between CF interfaces in UML.

![Figure 3: CF interfaces in UML](image)

The Base Application Interfaces are used by the application layer. They provide the basic building blocks of an application. The interfaces in this group are: Port, PortSupplier, LifeCycle, TestableObject, PropertySet, Resource and ResourceFactory.
The Framework Control Interfaces provide the control of the system. The application layer can reach the OS through these control interfaces. The interfaces in this group are: Application, ApplicationFactory, DomainManager, Device, LoadableDevice, ExecutableDevice, AggregateDevice and DeviceManager.

The Framework Services Interfaces provide the system services. These interfaces support both core and none-core applications. They include: File, FileSystem and FileManager.

The CF uses a domain profile to describe the components in the system. The domain profile is a set of XML files that describe the identity, capabilities, properties, inter-dependencies, and location of the hardware devices and software components that make up the system [1]. The software component characteristics are contained in the software package descriptor (SPD), software component descriptor (SCD) and software assembly descriptor (SAD).

The hardware device characteristics are stored in the device package descriptor (DPD) and device configuration descriptor (DCD). The Properties Descriptor contains information about the properties of a hardware device or software component. And the DomainManager configuration descriptor (DMD) contains the configuration information for the DomainManager.

Although the SCA uses the CORBA middleware for its software bus, the application layer can reach the OS by other means. CORBA adapters can be used to wrap the legacy software components. Figure 4 shows the relationship between the OS, the application and the OE.

Figure 5: Relationship between SCA components
3.2.2.2 Middleware

Middleware is a layer of software between the applications and the underlying network. This layer provides services like identification, authentication, naming, trading, security and directories. The middleware also provides hardware and location transparency to software entities. It functions as a translation layer. With middleware, software applications running on different platforms can communicate transparently. CORBA has been chosen as the middleware layer of the Software Communications Architecture.

Distributed processing is a fundamental aspect of the JTRS system architecture. Distributed processing is based on component model programming. Applications are designed as assemblies of components, and each component can run on a different part of the network [46]. The components are usually implemented as objects. Distributed processing differs from messaging middleware in that it causes processes to be executed in real time rather than by sending data.

CORBA is an architecture and infrastructure that computer applications use to work together over networks. A CORBA-based program from any vendor, on almost any computer, operating system, programming language, or network, can interoperate with a CORBA-based program from the same or another vendor on almost any other computer, operating system, programming language or network.

CORBA is used to provide a cross-platform middleware service that simplifies standardized client/server operations in this distributed environment by hiding the actual communication mechanisms under an Object Request Broker software bus [47].

Each object in a CORBA application is defined as an interface using IDL. Any client that wants to invoke an operation on the object must use this IDL interface to specify the operation it wants to perform, and to marshal the arguments that it sends. When the invocation reaches the target object, the same interface definition is used there to unmarshal the arguments so that the object can perform the requested operation with them.

The transparency and interoperability provided by CORBA is enabled by IDL. IDL separates the interface from the implementation. The object’s running code and its data are hidden from the rest of the system with a boundary that the clients cannot pass. Clients can only reach the objects through their advertised interfaces. An IDL compiler compiles the given IDL into client stubs and object skeletons. The stubs and skeletons act as proxies for clients and servers, and they run on top of Object Request Brokers (ORB). Figure 6 shows this configuration. The strict definitions of the interfaces provides the possibility for the
stubs and skeletons match together, even if they are written in different languages, and running on different platforms and on different ORBs.

Clients reach objects by using object references. In CORBA, every object has a unique object reference, and this reference can be obtained by a client by a number of ways. Once this reference is obtained, clients invoke operations on these objects as if they are local objects. Actually, these operations are invoked on the client’s stub, which then invokes the ORB on which it is running. This ORB locates the ORB that has the real object implementation. The invocation continues through the target ORB, and the skeleton on the implementation side, to get to the object where it is executed.

When the ORB that runs the client discovers that the actual object implementation is on a remote ORB, it routes the invocation out over the network to the remote object’s ORB. As the ORBs might be implemented by different vendors, and CORBA promises vendor-independent interoperability, the architecture specifies a common protocol called the Internet Inter-ORB Protocol (IIOP).

This protocol specifies a representation to specify the target object, operation, all parameters (input and output) of every type that may be used, and how all of this is represented over the wire. CORBA provides the mechanism through which different software defined radio vendors can develop compatible software and hardware interfaces. Any component on a radio can be replaced or upgraded, and the download process can be made transparent to the user.
**Waveform**

In the SCA context, a radio application is known as a *waveform*, which is defined as the set of transformations applied to information that is transmitted over the air and the corresponding set of transformations to convert received signals back to their information content. The core framework defines a common mechanism to manage and control waveforms and their components. Therefore, components can come from different sources and still use the same mechanisms to be deployed, connected, and managed.

Each new waveform on the radio system is deployed as a new application, which inherits the SCA CF Application. A waveform application creates a CF Resource for each functional unit, connects these resources to each other using CF Ports, and deploys these resources on CF Devices. New functionality can be added easily to an existing system by creating a new Resource, and connecting it to the necessary resources. When a new hardware device is installed on a system, a new logical CF Device is created as a software proxy for this hardware device, and the Device is registered to a CF DeviceManager.

Port provides a specialized connectivity to a component. It is used to set up and tear down connections between application components in the CF domain. A port is a logical element that enables components to exchange data. Ports are classified into Uses ports (clients) and Provides ports (servers). A provides port of a component is used to retrieve an object reference for a server object contained in the component. A uses port of a component is used to retrieve an object reference for a proxy object connected with a server object contained in another component. The Port interface also provides components with connect and disconnect functionalities, which are necessary to assemble waveforms. Figure 7 illustrates the connection of components via ports in SCA model.
Significant efforts are being carried out to facilitate SCA-based SDR software development on integrated development environments, reusable software modules, and implementations of software architectures. For example, the Open-Source SCA Implementation::Embedded (OSSIE) [17], developed at Virginia Tech for research and education, is an open SCA implementation that can reduce the entry cost of SCA development and training. But less attention has been brought to finding an effective and efficient testing strategy to make the system more robust and reliable.

It is also important to mention that the JTRS Test and Evaluation Laboratory (JTEL) has developed tools to help evaluate SCA compliance. For SCA version 2.2, JTEL developed the JTRS Test Application (JTAP) and Waveform Test Tool (WTT). The former tests core frameworks to make sure they implement all the OE interfaces correctly, and the latter tests that applications can be deployed and managed with a compliant framework. But the reliability of the component interaction analysis has not been seen yet and it is a very important factor to the success of SDR systems [27].

Figure 7: Connection of components via Ports
3.2.4 Wireless Network Protocol Analysis & Testing

One of the key advantages of SDR is that it can act as a network router in wireless communications systems. Routing plays an important role in the networking of the SDRs. Routing protocols should be fast, accurate, efficient and scalable [15]. As traditional routing protocols cannot apply to the software radio network, the protocols for the network should be designed and extended carefully to combine routing design with the characteristics of SDR network.

SDR is the main device for a user terminal to access different types of network using individual IP addresses. With SDR, wireless communication will become an integrated system comprised of sub-networks formed with various IP-capable waveforms.

Network services of SDR provide a comprehensive suite of networking functions, capabilities and services at the Open Systems Interconnection (OSI) Network Layer, including network addressing, packet routing, IP multicast, IP Quality of Service (QoS), IP infrastructure services and network management.

Figure 8 illustrates these functions as components in SCA context.

![Network services in SDR architecture](image)

Figure 9: Network services in SDR architecture
Su and Guo’s paper [15] proposed an implementation scheme of routing application based on SCA. The system architecture on which the routing application depends was presented. The paper emphasizes the design of routing components and application program interface (API) and using sequence diagrams to interpret the interaction among routing selection components, adaptive channel selection components and link layer components.

Earlier efforts to address the gap between network routing protocol development and realistic testing of mobile network behavior include ad hoc network test beds known as bench top, indoor, fixed outdoor, and mobile outdoor. Bench top test beds use link layer filtering, RF attenuators, or other emulation techniques to shrink the wireless range so that meaningful experiments can be performed within a single room [40]. These allow protocol development and testing to operate in an easy environment that is more realistic than simulation, but does not capture all significant behavior such as route discovery latency, overhead cost, and route caching correctness. Indoor test beds within a building provide more complex and realistic environments, especially when the intended application is indoors [38, 39]. These do not fully capture the mobility and propagation of the outdoor environment. Full scale outdoor test beds are often restricted to fixed sites [41, 43]. These efforts provide insights into a full-scale outdoor environment, but ignore mobility.

4. Proposed Research Approach

My research will consider two main viewpoints of testability from the SCA architectural viewpoint: controllability and observability. Freedman’s definition for controllability is: “controllability refers to the ease of producing a specified output from a specified input” [13]. The purpose is the overall control over the system and implementation from testability’s point of view. Jimenez, Taj and Weaver [12] summarize controllability as how easy it is to control a program on its inputs, output, operations and behavior. According to Freedman, “observability refers to the ease of determining if specified inputs affect the outputs” [13]. A more compact definition for observability is given by Jimenez: “observability in software is how easy it is to observe a program in terms of its operational behavior, input parameters, and outputs” [12]. The possibility of being able to view and understand the internals of a system enables the validation of the behavior by comparing it to the expected behavior. In my test approach, controllability and observability are increased by systematically reducing the complexity of software intensive embedded systems in architectural based modeling and analysis.
My focus in this research is to provide a more systematic approach to conduct integration testing to evaluate the dependencies between SDR waveform components. This is to ensure that components are following SCA specification consistently and communicating accurately. Therefore, this interaction information must occur between different software component development teams to generate interaction test cases.

The component information related to the application is derived from the SCA architecture framework. This method is suitable for increasing the testability of SCA-based embedded communications systems, which involves service composition, integration, interoperability, architecting, runtime re-composition and service composition control. I call this method Port-based Integration Testing for SCA-based embedded systems (PITSES).

PITSES is an architectural component-based integration testing approach with architectural test coverage criteria. This method is proposed to solve the following problems:
(1) Incomplete component interactions testing due to the complexity of the system
(2) No measurement for test coverage at software architectural level

The proposed project will develop criteria to test software defined radio empowered wireless communications systems. I expect these criteria to help testers generate meaningful and effective test cases. The outline for the proposed research is as follows:

- Define SCA architecture properties to be tested
- Define testing requirements
- Define architecture graphical representations
- Define test criteria
- Generate test cases
- Define validation method
- Develop prototype tool
4.1 General Properties to Be Tested at the Software Communication Architecture Level

SCA is a component-based architecture and can be viewed as components, connectors and configurations, where components perform the primary functionality of the system, including both software and hardware; connectors define the interactions between components; and configurations define the topology of the components and connectors.

1. Components

Application Components: Software that manipulates input data and determines the output of the system. The application software implements the base application interfaces.

Systems Components: Software that provides the capabilities for waveforms to execute and access the systems hardware resources. There are two types of system components – SCA devices and SCA services.

SCA Devices: Software components that provide access to the system hardware resources. The SCA devices implement the base device interfaces.

SCA Services: Non-hardware (software-only) resources provided by the system for use by applications.

The application components communicate with each other or with the system components through SCA-defined Port interfaces. Communications between the application and the framework services interfaces are accomplished through CORBA middleware. The APIs are standardized for a given system or domain so that all communications between the application and the system are uniform across multiple systems.

2. Connectors

Components may have multiple, disparate connections to multiple other components or to the same component. An application’s dependencies are specified as connections. The components of an SCA compatible application connect to each other through ports.

3. Interfaces

Component compositions are defined by interfaces to express logic relationships. Interfaces are used to build the basic blocks for a component, to assemble multiple components into a radio system and to reach the operating system. Interfaces connect application services, core framework services, CORBA services and operating systems.
These interfaces can be categorized into two groups: provider service interfaces and user service interfaces. In this approach, all components are considered plug-compatible in the sense that a service using an interface can be connected to a service providing interface.

4. Configurations

Configurations define the topology of components and connectors. The components are assembled according to a software assembly descriptor file. Figure 10 shows our focus in software communications architecture, which is used to propose test requirements for software architecture-based testing.

Figure 10: SCA components interaction
4.2 Software Communications Architecture Key Requirements

Architecture-based test criteria can be defined on the architecture properties described above. This would support algorithmically defining test data to cover the architecture and developing architectural test plans. Following are key architectural requirements to be tested at the software architecture level.

1. Component Requirements

In SCA, CF Resource shall be used to build a component. The Resource interface shall provide the operations to control and configure a component by inheriting the PortSupplier, TestableObject, PropertySet and LifeCycle interfaces, and by providing start and stop operations. New functionality shall be added to an existing system by creating a new Resource and connecting it to the necessary resources. When a new hardware device is installed on a system, a new logical CF Device shall be created as a software proxy for the hardware device, and the Device shall be registered to a CF DeviceManager.

Each application component shall bind its object reference to the naming context IOR using the name binding parameter. Each executable component of an application shall set its identifier attribute using the component identifier.

2. Connector Requirements

A waveform application shall create a CF Resource for each functional unit, connect these resources to each other using CF Ports, and deploy these resources on CF Devices. The Port interface shall provide the connect and disconnect operations needed to assemble and disassemble components. Application specific ports shall inherit the Port interface. These component dependent, application specific ports define the direction and control of the data flow. Components that provide ports shall inherit the PortSupplier interface, which defines the getPort operation. This operation is used to obtain a specific consumer or producer port.

3. Interface Requirements

Interfaces provided by a component shall be described in a software component descriptor file as service provider. Interfaces required by a component shall be described in a software component descriptor file as service user.
SCA service definitions consist of APIs, behavior, state, priority and additional information that provides the contract between the service provider and the service user. IDL is used to define the interfaces for service definitions to foster reuse and interoperability. IDL provides a method to inherit from multiple interfaces to form a new service definition.

4. Configuration Requirements

The hardware devices and software components that make up an SCA system domain shall be described by a set of files that are collectively referred to as a *domain profile*. These files describe the identity, capabilities, properties, inter-dependencies, and location of the hardware devices and software components that make up the system.

4.3 Graphical Representations of Architectures

Graphical representations have been used to visualize the definitions of test criteria. This proposal introduces two types of graphical representations to help define the dependencies relationships between interacting components.

The Component Port Interaction Graph (CPIG) represents the connectivity relationships between components in software communication architecture specifications. It helps to see the big picture of the overall system configuration and connectivity. The Component Port Behavior Graph (CPBG) represents the relationships between application, SCA services, SCA devices and ports in a component.

These two types of graphs will be used to define test criteria and further generate test cases.

4.3.1 Component Port Interaction Graph

The Component Port Interface Graph (CPIG) is used to describe the interrelation of components. It consists of three types of nodes.

1. **Component**: Components are represented as rectangular boxes.

2. **Connector**: There are two types of connectors: *provide service ports* and *use service ports*. Plain boxes represent *provide service* ports, while shaded boxes represent *use service* ports.

3. **Interface**: A solid arrow connects a *use service* port to a *provide service* port. A dash-line arrow connects interfaces inside a component and it represents a connection rule inside a component. For example, Figure 11 shows CPIG for multiple components in an application.
**CPIG path**

A CPIG path is a set of component, port and interface nodes, such that the first component user service port node connects to the second component *provide service* port node and the second component user service port node connects to the next component *provide service* port node.

Component \( C = (P, U) \), where

\[ P = \{ P_1, P_2, \ldots, P_n \} \] is the set of *provide service* ports

\[ U = \{ U_1, U_2, \ldots, U_n \} \] is the set of *use service* ports

The *provide* and *use services* of a component \( C \) are denoted by \( C.P \) and \( C.U \) and \( C.P \cap C.U = \emptyset \). If there is an edge from \( C_1.P_1 \) to \( C_2.U_21 \) in CPIG it means the user service \( U_21 \) of \( C_2 \) has been satisfied by the providing \( P_1 \) of \( C_1 \), so \( C_2.U_21 = C_1.P_1 \).

There is an interface mapping from *use services* to *provide services* of a component \( f: C_2.U_21 \rightarrow C_1.P_1 \). It means that the *use services* of component \( C_2 \) are provided by *provide services* of component \( C_1 \). For instance, a path in Figure 12 is \( C_1.U_1 \rightarrow C_2.P_21 \rightarrow C_2.U_31 \rightarrow C_3.P_3 \).
4.3.2 Component Port Behavior Graph

Components are represented in a CPBG as subnets, each of which shows the connection behavior of that particular component.

A CPIG is a directed graph where \( C = (V, E) \)

\[ V = ( \mathcal{S}_S \cup \mathcal{A} \cup \mathcal{S} \cup \mathcal{S}_t ) \] is a set of nodes within a component

\( \mathcal{S}_S \) is the start state node in a task

\( \mathcal{A} \) is the set of component interface nodes in application layer

\( \mathcal{S} \) is the set of service interface nodes in OE layer

\( \mathcal{S}_t \) is the target node of a task

\( E \) represents the set of directed edges. For example, an edge from a node \( v_i \) to \( v_j \) is represented as \( E_{i,j} = (v_i, v_j) \). A path or sub-path is a finite sequence of nodes \((v_0, v_1, \ldots, v_k)\), such that \( E_{i,i+1} = (v_i, v_{i+1}) \) for \( i = 0, 1, \ldots, k \).

A Component Port Behavior Graph represents all the data and control flow relevant to a test component. This is shown in Figure 13.
The component is represented to depict states, transitions, events, actions and message sequence of waveform operation behavior. This information contains all the possible component services execution and the dependencies between the services. The importance of having the CPBG model is that it captures all the possible component interactions information.

Component internal behavior and components interactive behavior are represented in the graph. As an example, the CPBG in Figure 14 represents the connection behavior between an Ethernet packet consumer and an Ethernet packet producer. The connection follows SCA architecture framework and the rules defined in interface requirement specification. The purpose is to send Ethernet packets from producers to consumers. Both components start their own tasks by initializing required port interfaces. The component Ethernet packet producer connects with the Ethernet packet consumer. It can then either disconnect or push Ethernet packets to the Ethernet consumer. The Ethernet consumer will open its connection at the Ethernet producer's request and wait for any request. Once it receives a `pushPacket` request from the Ethernet consumer, it will process the Ethernet packet and send out a result to the Ethernet producer. The Ethernet consumer will wait for more of Ethernet producer’s requests as long as the connection is kept open. When the Ethernet producer receives results from the Ethernet consumer, it may send out more requests to the Ethernet consumer or it may disconnect the connection. The above task repeats if the Ethernet consumer has more information to send to the Ethernet producer. If the Ethernet producer disconnects the connection, then the Ethernet consumer will close its connection immediately.
It can be seen from this example that the behavior inside a component interface and across two components can be represented in terms of CPBG graph representation. The CPBG will further be used to help generate test cases under defined test criteria.

**Figure 14: A CPBG Example**

**CPBG behavior path**

A *CPBG behavior path* is a set of state, application, service and interface nodes within a component subnet, such that

- There is a directed edge from the state start node to either a service or an application node.
- There is a directed edge from every application node to its corresponding service node.
- There is a directed edge from every service node to its corresponding application node.
- There is a directed edge from every service node to its corresponding interface node.
There is a directed edge from every user interface node to its corresponding provider interface node.

In general, every task produces a path in the component port behavior graph from its start state node, through the application, service and interface nodes, to the target state node. As an example, the CPBG behavior paths from Figure 12 are:

1. $\text{Producer.Ss} \rightarrow \text{Producer.A0} \rightarrow \text{Producer.S0} \rightarrow \text{Producer.A1} \rightarrow \text{Producer.S1} \rightarrow \text{Producer.A2} \rightarrow \text{Producer.S2} \rightarrow \text{Producer.A3} \rightarrow \text{Producer.S3} \rightarrow \text{Producer.S4} \rightarrow \text{Producer.St.}$
2. $\text{Producer.Ss} \rightarrow \text{Producer.A0} \rightarrow \text{Producer.S0} \rightarrow \text{Producer.S4} \rightarrow \text{Producer.St.}$
3. $\text{Producer.Ss} \rightarrow \text{Producer.A0} \rightarrow \text{Producer.S0} \rightarrow \text{Producer.A1} \rightarrow \text{Producer.S1} \rightarrow \text{Producer.A2} \rightarrow \text{Producer.S2} \rightarrow \text{Producer.A3} \rightarrow \text{Producer.S3} \rightarrow \text{Producer.S4} \rightarrow \text{Producer.St.}$
4. $\text{Producer.Ss} \rightarrow \text{Producer.A0} \rightarrow \text{Producer.S0} \rightarrow \text{Producer.A1} \rightarrow \text{Producer.S1} \rightarrow \text{Producer.A2} \rightarrow \text{Producer.S2} \rightarrow \text{Producer.A3} \rightarrow \text{Producer.S3} \rightarrow \text{Producer.S4} \rightarrow \text{Producer.St.}$
5. $\text{Producer.Ss} \rightarrow \text{Producer.A0} \rightarrow \text{Producer.S0} \rightarrow \text{Producer.A1} \rightarrow \text{Producer.S1} \rightarrow \text{Producer.A2} \rightarrow \text{Producer.S2} \rightarrow \text{Producer.A3} \rightarrow \text{Producer.S3} \rightarrow \text{Producer.A0} \rightarrow \text{Producer.S0} \rightarrow \text{Producer.A1} \rightarrow \text{Producer.S1} \rightarrow \text{Producer.A2} \rightarrow \text{Producer.S2} \rightarrow \text{Producer.A3} \rightarrow \text{Producer.S3} \rightarrow \text{Producer.A2} \rightarrow \text{Producer.S2} \rightarrow \text{Producer.A3} \rightarrow \text{Producer.S3} \rightarrow \text{Producer.S4} \rightarrow \text{Producer.St.}$

**CPBG Interaction path**

A CPBG interaction path is a CPBG path that crosses multiple component subnets. Between two subnets A and B, it starts with a state node in component subnet A, where this node's output transition leads to the port connection of component subnet B. The path ends at the place in B that is the output place of the transition that has a connection with A.

For example, in Figure 13, the CPBG interaction paths are shown as follows.

1. $\text{Producer.A0} \rightarrow \text{Producer.S0} \rightarrow \text{Producer.Up} \rightarrow \text{Consumer.Pc} \rightarrow \text{Consumer.S0} \rightarrow \text{Consumer.A0.}$
2. $\text{Producer.A0} \rightarrow \text{Producer.S0} \rightarrow \text{Producer.Up} \rightarrow \text{Consumer.Pc} \rightarrow \text{Consumer.S0} \rightarrow \text{Consumer.S4} \rightarrow \text{Consumer.St.}$
3. $\text{Consumer.A0} \rightarrow \text{Consumer.S1} \rightarrow \text{Consumer.Uc} \rightarrow \text{Consumer.Pc} \rightarrow \text{Producer.S1} \rightarrow \text{Producer.A2.}$
4. $\text{Consumer.A0} \rightarrow \text{Consumer.S1} \rightarrow \text{Consumer.Uc} \rightarrow \text{Consumer.Pc} \rightarrow \text{Producer.r.S1} \rightarrow \text{Producer.A0.}$
5. $\text{Producer.A2} \rightarrow \text{Producer.S2} \rightarrow \text{Producer.Up} \rightarrow \text{Consumer.Pc} \rightarrow \text{Consumer.S3} \rightarrow \text{Consumer.A2.}$
6. $\text{Consumer.A2} \rightarrow \text{Consumer.S2} \rightarrow \text{Consumer.Up} \rightarrow \text{Producer.Pc} \rightarrow \text{Producer.S3} \rightarrow \text{Producer.A2.}$

### 4.4 Test Criteria

Formal definitions of test criteria will be based on the graphs CPIG and CPBG. Formal definitions of the test criteria will appear in the dissertation. Graph coverage criteria are used to impose test requirements on a test set. Some possible test criteria are listed here:

- Each prime path in the CPIG should be exercised at least once.
- Each prime path in the CPBG should be exercised at least once.
- Infeasible Testing: Discover infeasible paths in CPBG and generated test cases for them.

Table 1 shows the graphs that are needed to satisfy testing requirements defined in section 4.2.

<table>
<thead>
<tr>
<th>Test Requirements</th>
<th>Graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>CF Resource requirements</td>
</tr>
<tr>
<td></td>
<td>CF Device requirements</td>
</tr>
<tr>
<td></td>
<td>CORBA Services requirements</td>
</tr>
<tr>
<td>Connector</td>
<td>CF Port requirements</td>
</tr>
<tr>
<td>Configuration</td>
<td>SCA descriptor requirements</td>
</tr>
<tr>
<td>Interface</td>
<td>SCA API requirements</td>
</tr>
</tbody>
</table>

Table 1: Testing Requirements and Graphs

### 4.5 Test Case Generation

Algorithms will be developed to generate test cases from the two graphs. The test case generation algorithms will be based on the formal definitions of the test criteria and the CPIG and the CPBG graphs.

A test path represents the execution of a test case on a graph. As an example, one test criterion is applied to the interactions between two SCA components. If we choose the test criteria of “test all the CB-paths in the CPBG,” then we will have multiple test scenarios.
Input values that satisfy each test path will be generated. This brings the process down from the design abstraction level to the implementation abstraction level. The input values are augmented with other values needed to run the tests (including values to reach the point in the software being tested, to display output, and to terminate the program). The test cases are then automated into test scripts (when feasible and practical), run on the software to produce results, and the results are evaluated. It is important that results from automation and execution be used to feed back into test design, resulting in modified or additional tests. An example of generating input values is given as follows.

Example: Testing an SDR waveform integration

SDR Solder Radio Waveform (SRW) runs on the Integrity platform and SRW network applications running on a Linux machine with an Integrity environment. The key components include a Link Layer Connection (LLC) provider, an LLC user, an SCA Ethernet device, an Ethernet Mux and an LOI VMN Ethernet driver. Network traffic will go through the SRW wireless network and transport it over Ethernet. The purpose of this test is to exercise IP functionality. Figure 15 illustrates the IP packet link between different components:

Figure 15: Waveform Components IP Packet Path
Different network traffic inputs will run the software under different packet paths. When testing the interactions between SDR SRW components, test scripts will be implemented in a Linux application to generate network traffic, including IP, ICMP, TCP and UDP traffic. With all the types of traffic generated, this test program can emulate true network activity. A traffic generator tool will be used to automatically generate test inputs to satisfy each test path. For example, a test path to fulfill “route discovery” will be enabled by inputting “route request” packets. A test case is specified as “Input route request message from SDR SRW Ethernet driver (IP address 196.168.1.1) to LLC provider (IP address 196.168.1.6)”. A route request message will automatically be generated by the tool with the IP address and the message type as inputs. The message generated will have all the fields required in a standard IP header format, including type of service (TOS), total length (TOL), identification (ID), flags (FLG), fragment offset (FRO), time to live (TTL), protocol (PRO), header checksum (IP_SUM), source address (SRC), destination address (DEST), options (OPT), padding and data.

Example:

```
7E FF 03 00 21 45 00 00 40 00 01 00 00 3C 11 E0 31 CE D9 8F 1F C7 B6 78 CB 04
63 00 35 00 2C AB DA 00 01 01 00 00 01 00 00 00 00 00 00 00 00 04 70 6F 70 06 6E 74 63 6F 6D 03 63 6F 6D 00 01 01 01 0B 81 7E
```

The above test data input are then automated into a test script, run on the software to produce results, and the results are evaluated.

### 4.6 Validation Method and Prototype Tool

Validation of this research will be conducted through application to an existing SDR communications system. The goal of the validation is to determine whether the proposed testing method can detect faults effectively. To facilitate this experiment, a prototype tool will be developed as part of the research to evaluate the proposed test criteria. The following sections provide an outline of the experimental design.

#### 4.6.1 Experiment Design

**4.6.1.1 Hardware**

The hardware to be used for this experiment involves GNU Radio and Universal Software Radio Peripheral (USRP). GNU Radio is a signal processing package. USRP is the most common hardware
platform to run GNU Radio. USRP allows general purpose computers to function as high bandwidth software radios.

4.6.1.2 Software

Subject software includes OSSIE based waveform software written in C++, an operating system, CORBA middleware and routing infrastructure for the ad-hoc routing protocols. As mentioned in section 2.2, the military, a driving force behind software-defined radio, has mandated that any radio developed for the Joint Tactical Radio System (JTRS) program of the Department of Defense (DOD) must be implemented in the SCA environment. The objective for OSSIE is to provide a platform that is simple, easy to expand, and open-source for the development of waveforms following the guidelines laid down by the SCA specifications under the JTRS program as well as the Object Management Group (OMG).

OSSIE implements most of the interfaces described in the CF. Framework Control Interfaces (e.g., DomainManager and ApplicationFactory) are application-independent and can be used directly. OSSIE also provides examples of base application and device interface implementations that can be used directly or as a reference.

4.6.1.3 Test Adequacy Criteria

Three existing, typical, methods will be compared:

(1) **Manual testing** based on experience and requirements specification. It is the standard way testing is currently done.

(2) **Simulation**. All types of scenarios can be run in simulation environment without a full deployment.

(3) The proposed testing method at the architecture level.

4.6.1.4 Test Data

Test data will be generated for each test criterion applied to the subject programs. Each set will satisfy a coverage criterion. The generation of each test data set will be specific to each test criterion applied.

Each subject program implements a waveform application. Each waveform serves as a proof-of-concept, and proves that any waveform can be tested in the proposed method. The waveforms to
be considered include Future Multiband Multimode Modular Tactical Radio (FM3TR), Solder Radio Waveform (SRW) and Wideband Network Waveform (WNW).

4.6.1.5 Fault Set

The research will be validated by determining the effectiveness of the software communication architecture-based test criteria at detecting faults that are associated with the connections of the SDR components. It is necessary to inject architecture related types of faults. This work will initiate a study of faults that occur at this level. A survey of fault types at architecture level will be developed.

4.6.1.5 Measurements

The fault detecting effectiveness of a given test adequacy criteria $C$ for a given architecture $A$ with respect to a specific fault set $F$ is defined as the ratio of the number of faults detected to the number of faults seeded. This measurement will be made for each pair of subject architecture program and test adequacy criterion.

4.6.2 Experimental Procedure

The conduct of the experiment will consist of several steps. In the following, let $AP$ be the set of architecture based programs, $C$ the set of test adequacy criteria described above, and $T$ be the set of test data sets generated for each combination of program and test adequacy criterion.

For each $ap \in AP$ and $c \in C$:

Step 1. Generate adequate test data set $T$.

Step 2. Define fault set $F$ for architecture-based program $AP$.

Step 3. For each $f \in F(ap)$, define the fault seeded architecture program $AP(f)$ by seeding $AP$ with faults, yielding fault-seeded architecture programs $ap(f)$ where each $ap(f) \in AP(f)$.

Step 4. Generate a test program to apply each test data $t \in T(ap,c)$ to $ap(f)$.

Step 5. For each $t \in T(ap,c)$, if it detects faults, increase $Num(ap,c)$ – the number of faults detected by test data set $T(ap,c)$.
Step 6. Determine the fault detection rate $R(\text{ap}, c)$, for test adequacy criterion $c$ with respect to architecture based program $p$, as: $R(\text{ap}, c) = \frac{\text{Num}(\text{ap}, c)}{|F(\text{ap}, c)|}$.

Step 7. Determine the fault detection effectiveness $E(\text{ap}, c)$, for test adequacy criterion $c$ with respect to architecture based program $p$, as: $E(\text{ap}, c) = \frac{R(\text{ap}, c)}{|T(\text{ap}, c)|}$.

The cost of applying the criteria will be estimated by calculating the number of tests performed, logging time spent on designing test cases and evaluating the effort to automate tests.

4.6.3 Prototype Tool

A prototype tool will be developed to generate test cases, test scripts, test interfaces, test program and test GUI by applying proposed test criteria. As shown in Figure 16, the prototype tool serves several purposes: experimentation support, data dissemination, data analysis and test automation.

Figure 16: The Prototype Tool
5. Unique Contributions

The proposed research will have the following contributions:

- Formal criteria for testing SCA-based embedded software systems
- A technique to apply during early software development phases
- A method to generate a complete component interactions test for complicated system
- A metric to measure test coverage at the architectural level
- A graph-based architecture modeling technique
- An effective way to do integration testing of component-based embedded system

6. Research Schedule

Remaining tasks:

- Formally define the CPIG graph, especially the mapping relations
- Formally define the CPBG graph. The CPBG graph may be extended to include more features when needed
- Formally define test criteria
- Develop techniques to refine test requirements into automated tests
- Apply testing technique to SCA based subject program
- Analyze test results
- Write up dissertation

Table 2 shows tentative schedule:

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
<th>End Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Formal Test Criteria Definition</td>
<td>July 1, 2011</td>
</tr>
<tr>
<td>2</td>
<td>Test Criteria Applied to SDR based communications system</td>
<td>September 15, 2011</td>
</tr>
<tr>
<td>3</td>
<td>Conduct of Experiment</td>
<td>November 15, 2011</td>
</tr>
<tr>
<td>4</td>
<td>Evaluation</td>
<td>January 6, 2012</td>
</tr>
<tr>
<td>5</td>
<td>Write-up Dissertation</td>
<td>February 15, 2012</td>
</tr>
</tbody>
</table>

Table 2: Tentative Schedule
7. Planed Papers

Papers from this researched will be prepared and submitted for ICST, Milcom, and TSE.

8. References


