SDR AND THE SCA: A WAVEFORM IMPLEMENTATION CASE STUDY

Rick Woodring; Jeremy Kneuper
(Nova Systems Solutions; Cincinnati, OH, USA; {rwoodring,jkneuper}@nsseng.com)

ABSTRACT

The Software definable radio (SDR) and the Software Communications Architecture (SCA) are poised to reshape the communications industry by replacing proprietary systems with a standardized, open architecture capable of integrating legacy, current, and future technology. Under the SCA, developers will be able to design and implement complete waveforms without requiring access to expensive, limited production radio hardware. System integrators will also be able to combine small components from many independent vendors to create fully functional waveform applications.

Nova Systems Solutions (NSS), a division of Nova Engineering, has collaborated with the thought leaders in the SDR industry. As a result, NSS is implementing an SCA-compliant, multilayer, shaped offset quadrature phase shift keyed (SOQPSK) waveform to serve as an example to waveform developers. The waveform demonstrates an SCA compliant design and the operational requirements placed on a waveform by the standard. Based on the experience and insight gained during the design and initial implementation of this waveform, this paper discusses the steps used in a successful design process as well as suggestions to assist with implementation.

1. INTRODUCTION

The SCA specification establishes a hardware-independent development framework with baseline requirements for the Joint Tactical Radio System (JTRS) software definable radios. These requirements are comprised of interface specifications, application programming interfaces (APIs), behavioral specifications, and rules. The goals of these specifications are to ensure the portability and configurability of the software and hardware, and to ensure interoperability of products developed using SCA [1].

The SOQPSK waveform is being developed under the Wideband Intra-battlegroup Communications (WIC) Small Business Innovation Research (SBIR) program sponsored by the Space and Naval Warfare Systems Command (SPAWAR). The SCA compliant SOQPSK waveform was originally targeted for operation within the JTRS Joint Technology Laboratory (JTeL) Waveform Test Environment. To make the waveform more accessible to developers, a custom user interface is also being developed.

To be SCA compliant, the radio must have a Core Framework (CF) which is an operating environment on which waveforms and other radio applications operate. The CF provides standard services used by waveform applications and an abstract interface to the underlying software and hardware of a radio. These services reduce the complexity of the implementation by bundling specific tasks such as file system access, logging, and application installation. Without the CF, the user would have to create each function and interface, which can prove to be extremely tedious and time consuming. As shown in Figure 1, the CF provides an abstraction between any SCA client software and the underlying hardware and software.

The SCA provides the user with the ability to combine and simplify common tasks such as construction, initialization, virtual connection, execution, release, and destruction of components and waveforms. The SCA allows the user to develop the radio software only once, and then port that software to other compliant hardware radio platforms. This portability eliminates re-development of the same software by multiple vendors, reducing acquisition cost and allowing the software to integrate with other SCA compliant applications. The portability afforded by a selection of SCA compliant components allows the user to "pick and choose" hardware and software from multiple vendors to be implemented on the radio. This interchangeability of SCA compliant components allows a "best of breed" evolution to occur.

The SOQPSK waveform is a multi-layer waveform consisting of network, data link control (DLC), and physical layers in both upstream and downstream channels. The implementation also includes a data source, data sink and channel simulator to simulate operational conditions.

2. KEY TECHNOLOGIES

Implementing the SOQPSK waveform involved the use of several key technologies, particularly Unified Modeling Language (UML), Interface Definition Language (IDL), Common Object Request Broker Architecture (CORBA), and eXtensible Markup Language (XML).

Proceeding of the SDR 03 Technical Conference and Product Exposition. Copyright © 2003 SDR Forum. All Rights Reserved

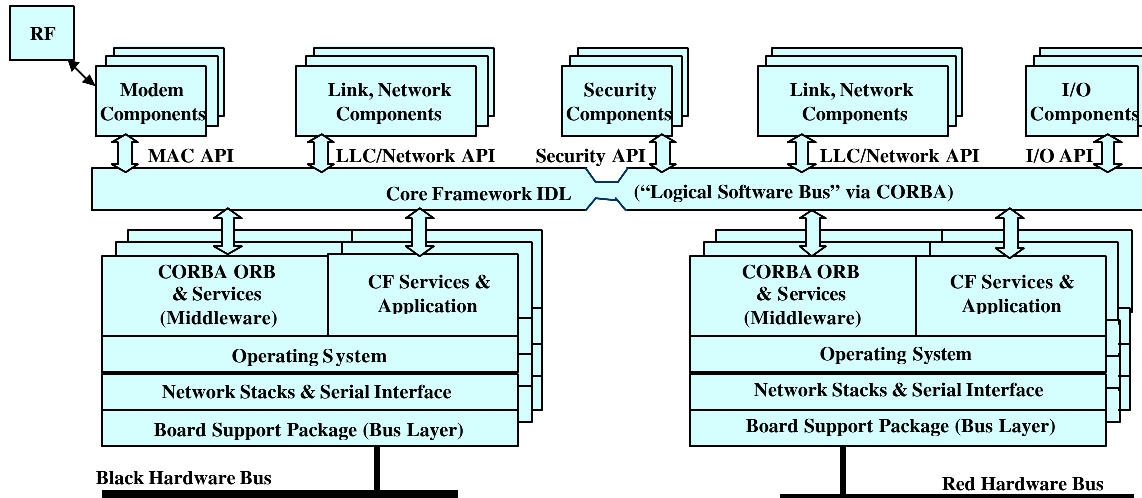

Figure 1: Software Structure of an SCA Radio. SCA separates waveform into components based on the Open Standards Interconnect (OSI) model. All communication from waveform components to the ORB and CF occur through CORBA requests [1].

UML is the industry standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. Standardized by the Object Management Group (OMG), UML simplifies the complex process of software design by making a graphical "blueprint" for construction. This blueprint represents the SCA interfaces, scenarios, use cases, and collaboration diagrams [1]. Figure 2 shows an example of a UML class diagram.

From each SCA interface defined in UML, an IDL representation can be derived. Like UML, the IDL is programming language independent, but IDL can be mapped into an implementation language such as C++, Java, Ada, and many others [1]. An IDL compiler supplied with the CORBA implementation performs the mapping.

In addition to IDL compilation, CORBA acts as the middleware between software components in a distributed environment. CORBA essentially establishes and controls interfacing between components and layers within an SCA application. CORBA is the "glue" that binds the SOQPSK waveform together.

The SCA uses XML to define a profile for the domain in which waveform applications can be managed. When creating an XML document, rather than drawing from a finite set of predefined elements, the developer creates unique elements and assigns them arbitrary names—hence the term "extensible." The user can therefore use XML to describe virtually any type of document, from a musical score to a software defined radio. However, for SCA applications, this extensibility is limited to the SCA-defined Document Type Definitions (DTD). The DTD files specify a set of rules for the structure and content of an XML document [2]. The DTD files list the elements, attributes, notations, and entities contained in a document, as well as their relationship to one another.

3. DESIGN PROCESS

NSS follows a structured methodology from conception through production to ensure the reliable execution of the waveform design. This process typically brings together designers, test engineers, and the customer at milestone intervals to review progress and coordinate execution. The documents produced in this design process contain the information required by JTeL to perform SCA compliance testing. During compliance and portability testing, JTeL representative will use the documents as directed in the JTRS waveform test and evaluation process.

The SOQPSK waveform design began with the creation of the System Requirement Specification (SRS). The SRS defines all testable requirements of the complete waveform, such as infrastructure and node operations. For this waveform, the SRS was written internally and approved by the customer. To complete the requirements definition process, waveform developers and JTeL representatives conduct a system requirements review.

After approval of the SRS, the Waveform Design Specification (WDS) was created. The WDS describes behavior of the complete waveform and specifies an implementation solution for each system requirement. The WDS also decomposes and describes the waveform application in terms of its constituent components. In the case of the SOQPSK waveform, those components are separated into data processing and waveform exercising components. The data processing components consist of an assembly controller, network down, network up, DLC down, DLC up, and physical modem. The exercising components include a data source, data sink, and channel simulator. The waveform exercising elements are discussed more in Section 5. To aid in the process of developing solutions for each

 Copyright © 2003 SDR Forum. All Rights Reserved

requirement, software structures and types are defined, and state machine diagrams are generated to describe activities. At a preliminary design review, developers and JTeL representatives evaluate the WDS on design, risk, test strategies, requirements adherence, and cost.

The software requirements document (SRD) includes detailed use case scenarios for all software requirements within the waveform. Based on the actions required of each component in the use cases, the developer decomposes the system requirements from the SRS into unit requirements for each waveform components. These component level requirements provide a roadmap for implementation of each component and ensure that the collection of waveform components fulfill all system requirements. A JTeL review of the SRD is not required.

The final step in the waveform design process is the Software Detailed Design (SDD) document. The SDD contains UML designs of each of the system components, along with detailed descriptions of each class. More importantly, the SDD describes in detail how all component requirements from the SRD will be fulfilled in the final software implementation. The SDD is reviewed by developers and JTeL representatives at a critical design review, where the design is examined for requirement satisfaction, risk assessment, and supportability.

The last document included in the design of the SOQPSK waveform is the Formal Qualification Test (FQT). The FQT is based on the SRS and can be developed in parallel with the WDS and SDD. The FQT defines a verification test for every requirement established in the SRS. Prior to testing, a JTeL representative must review the FQT procedures to ensure the accuracy of the tests. The JTeL representative must also witness the successful completion of the FQT to verify that the waveform conforms to the system requirements.

4. WAVEFORM CONSTRUCTION

The SOQPSK waveform designed and implemented in this study provides a reference example for developers designing multi-layer waveforms. To minimize the amount of waveform specific code included, this design sacrifices functionality for clarity of implementation. By limiting the quantity of waveform processing code included, developers can more easily separate the code required to make the waveform SCA compliant. The ability to distinguish between these different types of code will make this implementation a valuable resource to developers implementing SCA designs for the first time. Figure 3 shows the way in which the waveform components are connected.

The assembly controller provides a single interface through which the CF and client applications can access the waveform. The assembly controller is responsible for implementation of the *runTest*, *start*, *stop*, *configure*, and

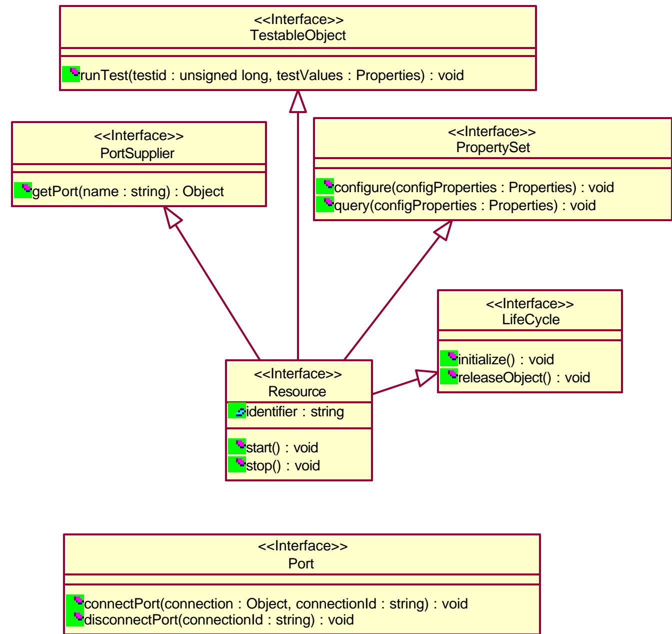

Figure 2: UML class diagram of SCA Resources and Port interfaces. SCA waveform developers are only required to implement the CF::Resource and CF::Port interfaces

query operations of the CF::Resource interface for the waveform. When a request is received, the assembly controller may delegate the request to another component of the waveform.

The data I/O layer of the SOQPSK waveform contains two components that exchange data with the network layer of the waveform. The data source component generates packets of data and then pushes the packets to the downstream network interface for transmission. The data sink accepts packets of data that have been received and processed by the upstream portion of the waveform from the network layer. Both components of the data I/O layer include the option to log packet contents to files.

The network layer contains upstream and downstream components that perform network layer operations on all packets processed. The downstream component attaches source and destination IP addresses to the front of all packets and also maintains a history containing the destination of the last ten packets sent. The upstream component reverses the operations of the downstream component by removing the IP addresses from the packet and recording the source IP to a history. In the example implementation, the network layer does not provide routing, multicasting, or relaying capabilities; however, additional capabilities could be added by replacing the example components with more complete components that implement the same data transfer interface.

The DLC layer also contains an upstream and a downstream component. Both components identify packets based on a unique packet sequence number and checking

Proceeding of the SDR 03 Technical Conference and Product Exposition. Copyright © 2003 SDR Forum. All Rights Reserved

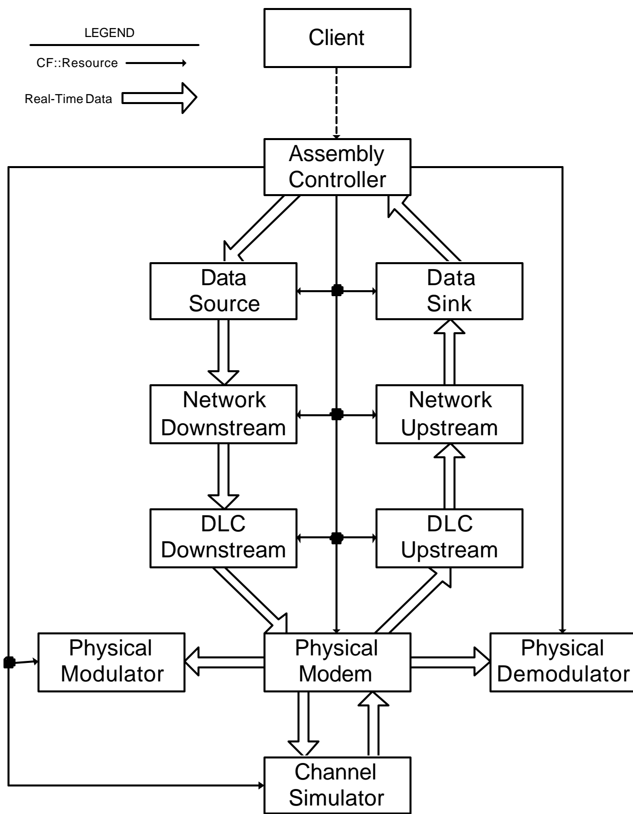

Figure 3: Waveform component structure. The assembly controller configures waveform components via the CF::Resource interface. Data is transmitted through real-time interfaces defined by each component.

for corruption with a 16 bit cyclic redundancy check (CRC). The downstream component attaches a CRC value and a sequence number to the front of each packet. The upstream component removes the values attached during transmission, validates the CRC of the packet, and records the sequence number. The DLC layer does not implement error correction, flow control, or packet retransmission. As with the network layer, additional functionality can be added by updating the implementation of this layer.

The physical layer contains three components: a bi-directional modem component that interfaces with adjacent layers and two coprocessor components that perform modulation and demodulation. When a packet is received on the downstream interface, the data is scrambled and SOQPSK modulation is performed. The resulting in-phase and quadrature (IQ) sample pairs are then pushed to the waveform output. When the physical layer is not transmitting data, it searches the IQ pairs entering the upstream interface for the start of a new packet. When a new packet is detected, the physical layer demodulates and descrambles the data. The recovered packet is then pushed to the upstream interface of the DLC layer.

The channel simulator consists of one element that receives IQ data from the waveform output and routes the data to the input of the waveform. The channel simulator is also capable of adding a combination of propagation delay, frequency offset, and additive white Gaussian noise (AWGN). These components form the full SOQPSK waveform implemented for this case study.

5. WAVEFORM IMPLEMENTATION

Prior to implementing the multi-component SOQPSK waveform, a generic waveform application containing only an assembly controller was created. The generic waveform was used to test the operation of the CF and assist in writing installation and debugging scripts. After successfully demonstrating the installation, operation, and uninstallation of a single component waveform in the CF, two additional components, a data source and data sink, were added to the waveform. To minimize the potential for configuration errors, the new components were added to the waveform without connecting any component ports. The waveform was then used to demonstrate the installation and instantiation of a multiple component waveform. After verifying that the CF loaded and started the waveform as expected, the component ports were connected to the rest of the waveform. The these ports, the new components were then exercised and tested. Additional components were added to the waveform in the same manner, one layer at a time.

As mentioned previously, the reference implementation of the waveform includes components that would normally not be included as part of the waveform. By default, the data source, data sink, and channel simulator components are included in the example waveform to simplify the process required of the end user to start and exercise the waveform. Alternatively, these components could be removed from the waveform and made into applications of their own.

Because of the component based nature of SCA waveforms, the extra components are easily moved from the waveform to a new application with only minor changes to the XML descriptors. To remove a component from the waveform, the component's implementation and instantiation lines must be removed from the software assembly descriptor (SAD) XML file for the waveform. Any port connection involving the component being removed must also be updated in the SAD. After removing the three extra components, the waveform can be used as expected to connect a data I/O interface to a transmission interface.

6. LESSONS LEARNED

While implementing the SOQPSK waveform, several unforeseen problems were encountered that could have been avoided with additional planning or precautionary measures during the initial waveform design. This section discusses potential waveform design and implementation pitfalls of circular references in IDL, code redundancy, user

 Copyright © 2003 SDR Forum. All Rights Reserved

interface development, and incremental testing. Suggestions to avoid these problems in future development are also proposed.

One troublesome situation encountered during waveform implementation were circular references in the IDL. Initially, Rational Rose generated all the waveform IDL based on the UML model created during the system design phase of the development. Upon initial visual inspection, the IDL appeared to be correct, but it failed to compile. A closer inspection revealed that the way the IDL was separated into files caused *file 1* includes *file 2*, *file 2* includes *file 3*, and *file 3* includes *file 1* creating a circular reference. Because the language provides no measures to prevent compilation of an IDL file multiple times, these scenarios created an infinite loop for the compiler.

In order to correct the circular references, the separation of interfaces in the IDL was broken down from one IDL file per layer to several files. For each layer, the IDL was separated into real time and non-real time interfaces for upstream and downstream components, and a single file that holds constants and type definitions for the whole layer. In order to reduce the number of IDL files included, complex structures based on type definitions from several layers were replaced with standard types. The most common example of converting structures to an array of standard types is the payload parameter of the pushPacket operation. For instance, the payload parameter was originally a structure containing two IP addresses and an octet sequence with a maximum length of 1500. To avoid the need for other layers to understand the contents of a DLC payload, the parameter was modified to be only one octet sequence with a maximum length of 1508, which corresponds to four octets for each IP address and 1500 for the data.

Another important waveform development practice is to place the implementation of frequently used interfaces such as CF::Resource and CF::Port (shown in Figure 2) in a common base class. For example, rewriting the implementation of a common interface every time it is used can create unnecessarily long code that is difficult to manage. By placing the implementation in a base class that is inherited by the component classes, the default behavior of all resources can be the same. Restricting an implementation to one class also makes updates easier because the change is required in only one place rather than in every component.

Code reuse can also decrease the implementation time of a waveform in the XML descriptors. Every CF::Resource component within a system must have a software package descriptor (SPD), software component descriptor (SCD), and properties file (PRF). These files have a high percentage of redundancy from one component to another. Consequently, writing XML files from scratch for each component of a waveform involves an unnecessary amount of document reconstruction. By creating a generic component such as the one initially used to test the operation of the CF, a generic CF::Resource component and a corresponding set of XML files can also be validated. The generic component and XML files can then serve as the base for creation and testing of XML for all future components.

Another implementation detail that is not immediately apparent from the SCA specification is that the user interface should not be part of the waveform. When creating a custom software interface for a waveform, it may be tempting to include a user interface as part of the waveform. This, however, is not necessary. The user interface should exist independently from the waveform and only interface with the waveform via standard SCA methods. An important implication of including the user interface as a waveform component is that the interface must be a CF::Resource and can only be used once the waveform has been started. A second implication is that the interface must have an ORB accepting requests, rather that just acting as a client.

An alternative to having the user interface loaded and connected to the waveform automatically is to have the user interface be a client to the CF::DomainManager as well as to the application. As a client to the CF::DomainManager, the user interface is capable of requesting the list of instantiated applications and searching for the appropriate one. This solution allows independent development of the waveform and user interface and ensures that other user interfaces can also communicate with a waveform.

7. CONCLUSION

Through a structured design process compatible with the JTRS waveform testing and evaluation process, Nova Systems Solutions has designed a multi-layer SOQPSK waveform. This waveform provides an example implementation of both waveform exercise and bit processing components operating within an SCA core framework. The bit processing components developed in this case study do not contain complete layer 2/3 functionality, but serve as placeholders where more complex implementations may be added later. Through development of this waveform, NSS has encountered and addressed implementation problems. The waveform will also serve as a reference to future waveform developers.

8. REFERENCES

[1] Software Communications Architecture Specification, MSRC-5000SCA, V2.2, November 2001.

[2] Joint Tactical Radio Systems SCA Developer's Guide, Contract No. DAAB15-00-3-0001, V1.1, June 2002

Proceeding of the SDR 03 Technical Conference and Product Exposition. Copyright © 2003 SDR Forum. All Rights Reserved