STRATEGIES AND INSIGHTS INTO SCA-COMPLIANT WAVEFORM APPLICATION DEVELOPMENT

Yun Zhang, Scott Dyer, Nick Bulat
The MITRE Corporation
Bedford, MA

ABSTRACT

Embedded communications engineers are faced with numerous challenges as radio technology drives towards Software-Defined Radio (SDR)-based heterogeneous compute platforms. The Joint Tactical Radio System (JTRS) program has attempted to ease development and porting costs by requiring the use of the Software Communication Architecture (SCA) for SDR systems. The SCA is there to help understanding by providing a common framework; however, communication engineers must work in an unfamiliar environment filled with system software concepts such as Object Oriented Programming (OOP), Portable Operating System Interface (POSIX), and middleware using Common Object Request Broker Architecture (CORBA). The addition of SCA to SDR development is impacting design methods, work flow, testing, and tooling. Understanding of these impacts, and how to best capitalize on the benefits of SCA, is imperative to the success of any SCA-compliant development. This paper will provide insights into SCA development, work flow, testing, and tooling. This paper will also present an approach to leverage SCA as a means of abstracting the radio management software from the radio functional software components and further parallelize system development.

INTRODUCTION

The mission of the JTRS program is to develop a family of affordable, high-capacity tactical radios to provide both line-of-sight and beyond-line-of-sight Command, Control, Communications, Computers and Intelligence (C4I) capabilities to the warfighters. The cornerstone of JTRS is the development and deployment of Software-Defined Radio (SDR) technology through standardized, open software architecture - SCA. SDR is flexible, by virtue of being programmable, to accommodate various physical layer formats and protocols. Separating the software application from the hardware on which it is hosted (the radio platform), provides significant advantages in application portability, reduced development time, software reuse, and cost-effective utilization of Commercial Off-the-Shelf (COTS) technology. Adhering to a common open system architecture – SCA provides further advantages of interoperability by allowing sharing waveform software between radios, even radios in different physical domains, advantages of life-cycle cost reduction, and new capabilities through software upgrades.

However, as a system specification, SCA introduces a new unfamiliar environment filled with system software concepts such as Object Oriented Programming (OOP), Portable Operating System Interface (POSIX), and middleware using Common Object Request Broker Architecture (CORBA) to embedded communication engineers. SCA requires developers to have a certain level knowledge of both the development and the deployment environment. The SCA design strategies, work flow, testing and tooling are different from traditional embedded system development. In SCA, the focus is on individual components; this provides a natural separation of concerns for waveform functional logic. Furthermore, there is a level of abstraction from the hardware mandated by the SCA. This level of abstraction allows developers to work on components in isolation; build, test, and deploy components with or without the business logic.

This paper attempts to provide some insights into SCA development based on our experience. There are certain challenges faced in design, development and testing of waveform software due to the component-based architecture of SCA and the natural separation this provides between the radio management software and the radio functional software.

EHF LITE CHALLENGES

Higher bandwidth waveforms require Field Programmable Gate Array (FPGA) and Digital Signal Processor (DSP) implementations to sustain the high data rate, and high speed signal processing requirements of such waveforms. The SCA was designed primarily for radios operating below 2 GHz where high-speed specialized processing was unnecessary and processing could be performed completely in General Purpose Processors (GPPs). Such processing implementations are not natively supported by the SCA Version 2.2 Specification [1]. The MITRE Programmable Radio Technology (PRT) laboratory was founded with a goal to investigate the challenges of implementing SCA-compliant waveforms on software con-

figurable modems to meet the needs of the SATCOM community. EHF Lite is representative of high bandwidth SATCOM waveforms and incorporates elements such as Gaussian Minimum Shift Keying (GMSK) modulation, Turbo encoding, interleaving, and hop structuring.

The SCA defines a set of rules that constrain the design of systems. To make our waveform application fully SCA-compliant, we explicitly followed these rules. For the Operating Environment (OE), we used Harris dmTK SCA Core Framework (CF) v2.2.2, Real-time CORBA TAO, and Real-time Operating System VxWorks on Spectrum Signal System SDR-3001. The Board Support Package (BSP) provided by Spectrum is running in conjunction with the SCA domain manager for the Core Framework and provides the target platform dependent support required to run SCA applications on SDR-3001.

The purpose of this prototyping effort was to understand SCA portability, interoperability and overhead in the context of development of bandwidth efficient, flexible and extensible waveforms. We pushed the "boundary" of SCA down into the internals of the modem and are inline with SCA 3.0 specification. We took the data source, encoder, interleaver and modulator, etc. as waveform components and made these components SCA Resources; see Figure 1. With the individual components making up the information processing pipeline of the modem, we are able to change parameters either at deployment, via the attributes for the components in the SCA deployment descriptors, or via a Java user interface that uses CORBA to communicate with the SCA components and modify any writable (changeable) parameters on the fly.

We are investigating the challenges of implementing a high speed SCA-compliant waveform including the effects of SCA on waveform timing requirements, portability and extensibility; implementing waveforms with DSPs and FPGAs, and developmenting APIs to maximize waveform portability while minimizing negative impact. The implementation of the prototypical waveform follows the general approach outlined in the JTRS SCA Developer's Guide. In addition, the waveform attempts to implement as many of the processing components as possible as SCA resources that can be swapped out and replaced with alternative versions, in order to measure the performance impacts that differing implementations have on the waveform throughput.

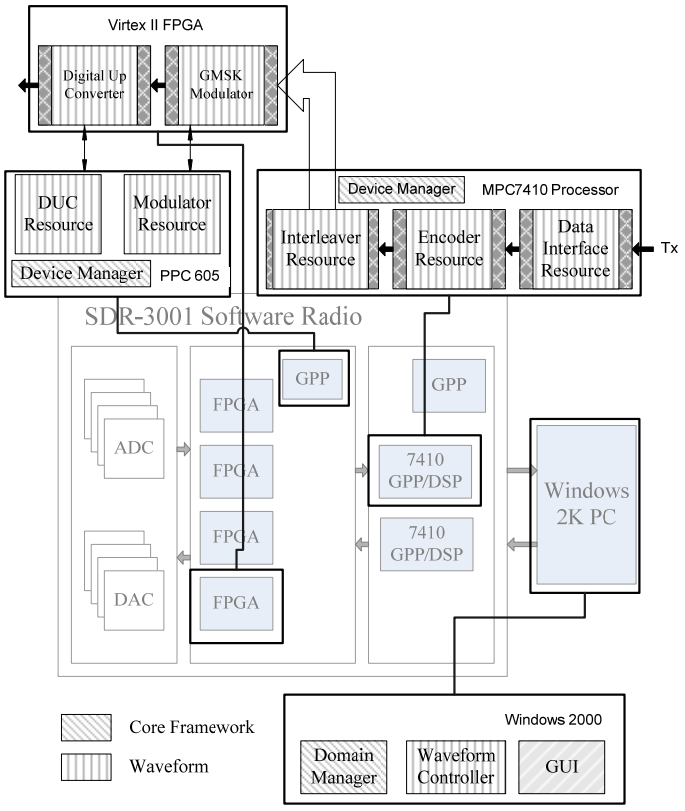

Figure 1 - Mapping Functions to Resources

EHF Lite was the first SCA development effort for the MITRE PRT Lab team. A great deal of effort was spent digesting and understanding the SCA specification. Knowing we faced this challenge, we divided the development effort up among three teams: communication engineering, waveform function development, and SCA development. The communication team developed the initial proof of concept, specification, and bit accurate models of the waveform. The waveform function development team worked closely with the communication team to take the specifications and implement waveform functions in C/C++ and VHDL with careful attention paid to portability. The SCA development team developed the empty SCA components and interface, and later incorporated the waveform functional code into the SCA components. The development work flow can easily be surmised as three teams implementing, testing, and synchronizing waveform development, see Figure 2. With respect to portability, we made every effort to limit RTOS system calls to POSIX interfaces, avoid the use of proprietary IP, and made use of Hardware Abstraction Layer Connectivity (HAL-C) for the FPGA.

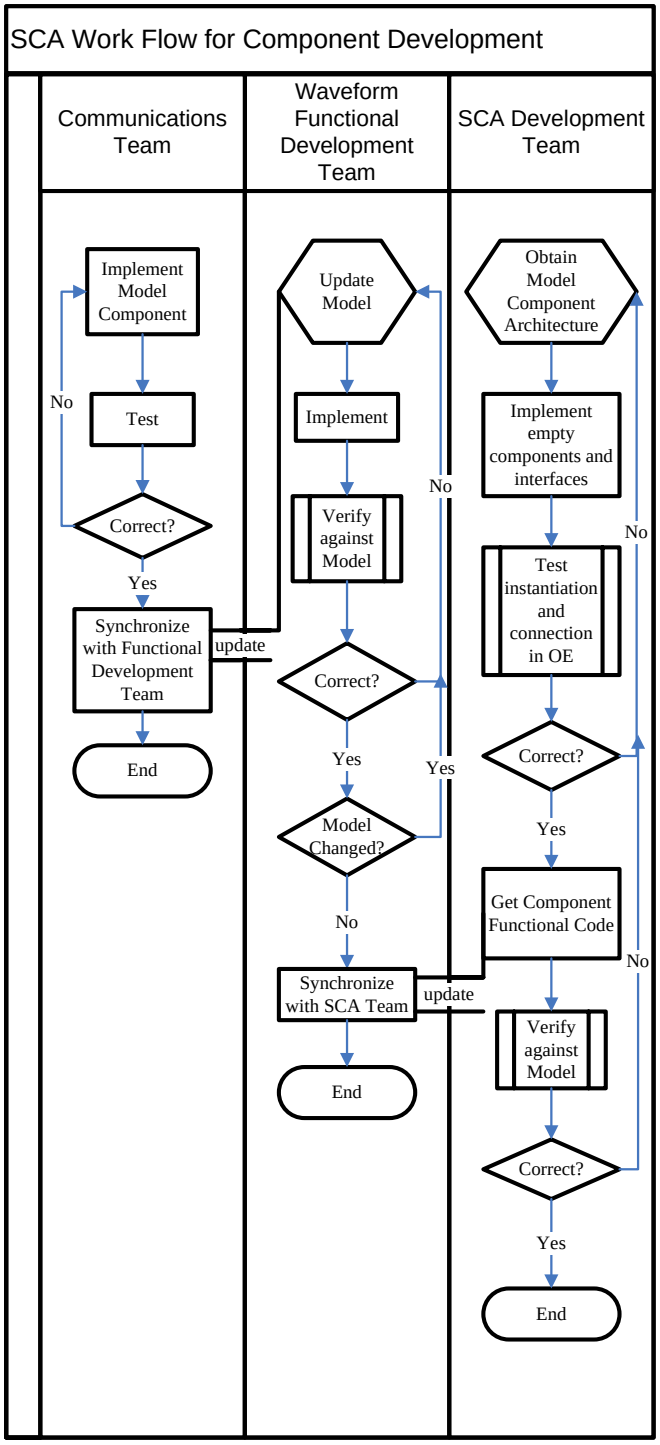

Figure 2 - Development Work Flow

For the embedded systems engineer without CORBA and OOP background, it could take several months to fully understand SCA. Early EHF Lite development activity focused around the communication team writing the waveform specification and creating a MATLAB model of the waveform. This provided a window of time for our waveform function developers to get familiar with the target, and the SCA developers to become familiar with the SCA.

In our efforts to understand SCA we worked with examples provided by Spectrum Signal Processing. The examples were relatively easy to follow; however, when we defined our own set of SCA components, a great deal of time was spent creating and debugging domain profiles. The majority of the time was spent manually cross checking consistency among the application XML files, source code, and vendor-provided Device Properties Descriptors. Inconsistency in these files could only be detected at run time. Run time errors were often vague and almost never referred to specific lines in XML files. Once consistency was achieved, a single change or addition to the XML would inevitably result in new errors. The waveform complexity increases the probability of errors in the XML files and increases the time to isolate any single error.

Along our EHF Lite waveform development we found unit test and integration challenging. The waveform function team tested each modem function prior to integration with the SCA environment. The SCA development team tested the connectivity surrounding the modem function. Even though we performed unit test on each waveform functional block we were not immune to the challenges faced in classic real-time embedded systems. For example, our interleaver and encoder initially resided on different processors. The encoder and interleaver are connected via SCA ports that are CORBA based. The overhead associated with CORBA could not satisfy the required data rates and forced the collocation of interleaver and encoder on the same processor. To the credit of SCA this required only a change in the XML.

When we implemented our SCA adaptors and modem function wrapper classes, we took a modular approach that would allow us to plug in new modem functions and dynamically (re)configure processing paths and components. We observed that there is a trade off between the finer SCA implementation granularity with higher portability and the coarse SCA implementation granularity with better performance. However, finer granularity components can also be applied to a wider range of applications, providing a level of portability between waveforms.

PRACTICAL SCA DESIGN STRATEGIES

Traditional embedded system developers have a number of choices regarding the software execution model; specifically writing their own scheduler, or using a number of

available real and non-Real-Time Operating Systems. Regardless of the "OS" choice the developer must divide the application into logical abstract parts, provide necessary synchronization, and then assign appropriate priorities and resources. This provides the developer a great deal of flexibility and control.

In the SCA domain, logical abstract parts are put into SCA component containers. This is very different from traditional embedded system development. SCA components are executed through the Object Request Broker as a single entity. Understanding the ORBs execution model and overhead are essential in developing reliable waveform applications. RT-CORBA is prevalent for a large number of the Operating Environments (OE) available today. However, SCA 2.2 has language in the specification that suggests waveform applications adhere to the minimum outlined in the specification [1]. SCA's minimum required services for CORBA are represented by minimumCORBA. Although most ORBs used in SCA-compliant targets are in fact RT-CORBA, the real-time features are rarely used in the developed waveforms due to this interpretation of the specification. There has been a great deal of discussion around RT-CORBA in the SCA community; it would be wise for developers to track this as the specification evolves.

SCA can be difficult to grasp for those new to the domain. New developers are looking for very specific guidance and direction, however, the SCA aspires to not limit or impede design or implementation. SCA is best thought of as a minimum set of services to provide instantiation, configuration, connectivity, and control of resources. SCA does not specify the roles and responsibilities of the resources outside of these bounds. Therefore, it is up to the development team to decide the level of granularity they wish to apply to the waveform applications.

The choice of SCA component granularity will affect system performance and efficiency as seen in our EHF Lite development. For each SCA component, there is overhead associated with it being a CORBA object. CORBA services will add overhead in program text space, stack, and heap. For fine grain waveform applications, this overhead can be significant.

Performance costs are also associated with calls between CORBA Objects. When the two components are on the same system, this overhead is minimum, resulting in an additional function call. Even with such minimal impact, an extra function call multiplied by the number of components in a real-time system can be significant. Thus, waveform software architects need to weigh all associated costs when considering the software granularity.

Whether it is traditional waveform software development, or waveform software development utilizing SCA, an appropriate reference model created in a high-level language, such as MATLAB, is essential to the success of software radio development. It is both possible and desirable to create a waveform model prior to platform selection or specification. A good reference model will facilitate a greater understanding of platform resource requirements.

Reference models are usually implemented in high-level languages without the constraints of meeting real-time requirements. It is not necessary to prototype all modes or features of a waveform. It is usually sufficient to implement the set of modes/features that represents the waveforms most resource intensive operations or highest risk with respect to implementation.

Taking a lesson from the OMG PIM/PSM approach [3], it is desirable to define components in the implementation that have identical functionality as corollary components in the reference model(e.g. create components similar to the libraries found in MATLAB®, or functionality defined by IT++ [4], or GNU Radio [5]). The benefit to this strong correlation between model and implementation can be realized in a number of ways. Identical data passed to a reference model component and its implementation counterpart can verify correctness of the implementation component. This is an excellent indication of completeness to the developer implementing components defined in the model. Validation of system components can help provide insight into correctness of the aggregate system. A discrepancy between implementation and the waveform model will expedite problem isolation and resolution. This allows side-by-side comparison of developed software with the reference model for debugging, testing, and system verification.

When it comes to implementation, the functionality represented by the reference model must be implemented in a language that will allow the creation of artifacts appropriate for execution on the target. As discussed in the last section of this paper, there are benefits to creating components that have a counter part with identical functionality in the reference model. However, the final implementation will have additional management software, and provide greater flexibility in configuration and control which may not necessarily be present in the model. The SCA leverages CORBA in order to provide a platform-independent, language-independent architecture for writ-

ing distributed, object-oriented applications in heterogeneous systems. This allows radio system developers to wrap the waveform signal processing logic in platform independent containers. This approach naturally suggests the parallel development that allows developers to work in their own domain.

SCA TOOLS

In the early days of SCA, XML files had to be edited by hand. This is a tedious task which involves verifying lengthy arbitrary strings such as UUIDs. In attempts to reduce this burden, developers have employed scripting languages such as Perl or utilized XSLT transformation [6]. A few early development tools made strides to reduce some of the burden of XML editing but still left some tedious tasks to the user.

Currently there are a few powerful SCA tools on the market –such as CRC Scari Suite, PrismTech Spectra, and Zeligsoft Component Enabler– that present a visual model of the system, show component connections in a schematic-like view, and drastically simplify XML generation. Competition in this market has pushed these tools beyond XML editing and generation; code and build artifact generation is now also featured. The tools also support target and deployment modeling.

Perhaps most beneficial to waveform porting activity is the ability of these tools to reverse engineer existing waveforms and display a graphical schematic view. Unlike most generic code understanding tools, these tools only show the high-level abstraction defined by the waveform application XML files, a much preferred first view of the application before diving into details of the source code.

SCA WAVEFORM COMPLIANCE

SCA waveform compliance verification is handled through the Joint Tactical Radio System (JTRS) Technology Laboratory (JTeL). The JTeL provides support and guidance for SCA verification and provides final assessment and compliance recommendations to the JTRS Joint Program Office (JPO). Currently, the JTeL test procedures and guidance are tailored for use with JTRS programs. The procedures are not well aligned with non-JTRS developments which may also require SCA compliance verification such as high bandwidth SATCOM waveforms and terminals. Thus, for SCA developments seeking compliance verification, it is advantageous to begin the JTeL coordination process early in the development cycle.

Current JTeL procedures call for coordination to begin early with an assigned JTeL test director involved in program milestones including SDR and PDR; these procedures also call for waveform developers to be notified of their target platform by waveform CDR. However, without early coordination, some of these test requirements may become problematic. For instance, there may be difficulty in identifying target platforms capable of running some high bandwidth waveforms. The focus of current JTeL efforts on JTRS programs increases the need for non-JTRS programs to begin coordination early in order to identify and resolve issues unique to individual programs.

The SCA 2.2 specification [1] contains a total of 487 requirements split between the waveform application and the Operating Environment (OE), with 192 requirements applying directly to the waveform as indicated in the Waveform SCA Test and Evaluation Plan (TEP). In order to facilitate the testing of SCA requirements, the JTeL has developed several software tools to aid in testing. These tools are not required for use in verification but are provided to aid in testing. Programs are allowed to utilize their own test suites; however, the verification approach must be coordinated with and approved by the JTeL.

The Waveform Test Tool (WTT) version 2005 SP1 [7] and Data Reduction Parser (DRP) comprise the test software supplied by JTeL to aid in waveform SCA compliance testing. The JTeL also produces the JTeL Test Application (JTAP) to assist in testing an Operating Environment for SCA compliance. The current versions of the tools only support testing to version 2.2 of the SCA specification.

The JTeL Waveform SCA TEP divides waveform compliance testing into three major categories: off-line, run-time, and Application Program Interface (API) inspection. Of these, the WTT is designed to support the run-time test category while the DRP is designed to serve as a backup for some WTT verification while providing custom parsing/search support for the off-line test category. The off-line and API inspection portions cover the remainder of the SCA waveform requirements and utilize manual code inspections aided by COTS tools as well as custom parsers and editors.

The JTeL DRP was designed to assist with off-line SCA waveform compliance testing. The DRP does not require any connection to a representative set unlike the WTT. The DRP can be used in combination with the WTT to track the status of tool-assisted test results; however, its main purpose is to assist with manual off-line testing. The

DRP functions mainly as a test symbol generator and custom data parser/search tool. The DRP parses application code and locates relevant code blocks to allow for manual algorithm inspections. In addition, many of the supported tests function as back-ups to run-time requirements verification supported by the WTT.

Out of the 192 requirements identified in the Waveform SCA TEP, the WTT/DRP test suite addresses only 91 of those requirements. The remainder of the requirements must be verified manually with the aid of additional data parsers and search tools. Of the 91 tool supported requirements, a majority also require additional manual verification.

The WTT and DRP should succeed in reducing the overall effort required to perform waveform SCA verification. However, the tools support a limited number of the overall waveform requirements and manual testing is still required for many of the tool supported tests. So while the JTeL tools serve to reduce the work load, there is still a large level of effort in manual code inspection and verification needed for compliance testing. This fact further supports the need for early program focus on SCA compliance and coordination with JTeL. Commercial tools such as CRC Scari Suite, PrismTech Spectra and Zeligsoft CE provide developers with additional verification tools that have not been previously available. These tools are capable of verifying numerous SCA requirements through XML parsing and inspection and could replace or augment the JTeL-provided tools given specific program needs. However, any use of non-JTeL tools would require coordination with the JTeL. In addition, a majority of SCA requirements would still require manual inspection and testing.

WAVEFORM PORTING

One of the JTRS program objectives is the reduction of logistics cost through reuse of common software. Our SCA implementation is intended to verify reusability and portability. We performed a porting exercise on our EHF-lite waveform. The source platform includes a Pentium 4 PC running Windows 2000 and a Spectrum Signal SDR-3001 using VxWorks RTOS, and Harris dmTK SCA Core Framework v2.2.2, Real-time CORBA TAO and Spectrum Signal Board Support Package running in conjunction with the SCA domain manager. The porting target platform is an embedded Pentium M PC running Fedora Core 3, CRC Scari++ Core Framework and RT-CORBA TAO with a PCMCIA Red River FPGA card.

As with any embedded development, there is a fair amount of time required in preparation and training prior to porting to include (1) Evaluation of the source and destination platforms resources; (2) Analysis of hardware and software dependencies, risk and mitigation; (3) Evaluation of possible functional allocation changes based upon the application requirements and hardware capacities; (4) Estimation of build and execution environment changes. The SCA adds to these, OE differences such as the ORBs, Core Frameworks, and device support for SCA.

Through our exercise, we learned that the porting effort differs based on several factors. One is the complexity of the porting waveforms and the processing elements. For example, our modulation function resides on FPGAs for the SDR-3001 system with BSP provided by Spectrum Signal. On the porting platform, we had to create our own similar board support package for our embedded FPGA to support SCA. The portion of waveforms on the GPP is easier to port than on specialized hardware such as DSP, FPGA and ASIC due to the limited CORBA support and the lack of standard hardware abstraction for these devices. Another factor is the level of the SCA compliance. Ideally, radio platforms, SCA Core Framework implementations, ORBs, RTOSs and waveforms should be SCA-compliant, but right now there are only a couple of hardware platforms, CFs and ORBs certified as SCA-compliant. The design of a waveform is one major factor that affects SCA portability. It is possible to create a single super SCA adaptor that is SCA-compliant. However, it could be a great deal of effort to port such a waveform to a new platform if repartitioning of functionality across multiple processing elements is required. With sufficient documentation, powerful SCA development tools, and knowledgeable SCA developers applying appropriate granularity, porting costs can be reduced.

To increase SCA-compliant waveform portability, developers should use POSIX interfaces and avoid OS and platform dependent libraries. Sometimes, such APIs have to be used, but should be confined in waveform functional software. For example, EHF Lite makes use of Spectrum Signal's quickComm library to utilize rapid I/O for the higher data rate. However, it is mapped into a CORBA pushPacket interface and isolated from the SCA components.

The SCA specification is a system specification, not an implementation specification. SCA Core Framework implementers might have different interpretations of the specification resulting in different implementation approaches. SCA developers should watch out for proprie-

tary APIs provided by Core Framework implementations. These APIs could be utilities for threading, synchronization and buffering, etc. The SCA Core Framework implementers also have their own set of APIs for the SCA port connection and entry points for starting execution. All of these areas need to be modified if the porting target uses a different Core Framework. Separating the SCA components from their waveform business logic will ease later porting efforts. Fortunately, SCA development tools with code generation capabilities can be of assistance in these areas.

NEXT STEPS FOR SCA

The SCA provides an operating environment that strongly encourages portable software development, where portable is defined as the reuse of software for a given application on two or more targets with minimal modification. This is possible since the SCA defines an Operating Environment that is identical on each target. However, the reuse of software components from one application by another application requires that each of the applications have the same API for that component. In order to allow a broader reuse of software components between applications, it is necessary to define a set of standard components and interfaces implemented in portable software. An excellent example of this approach is the GNU Radio project [5]. Another noteworthy set of portable signal processing open-source software applicable to communications is the IT++ library [4].

The down side to standardizing on APIs is that it narrows the applicability of the SCA. The current SCA specification is usable for a wide range of applications where distributed deployment, connection, configuration, and control are desired. Included in the SCA is a set of APIs that center around communication systems. In order to preserve this broad range of applicability, the waveform development APIs must remain apart from the SCA. Furthermore, the focus of the SDR community needs to shift from the SCA to the waveform development API. Attention should only deviate from the waveform development API to the SCA when the operating environment fails to provide generic functionality required by communication systems engineers. An example of this would be for real-time services. Real-time services are not limited to the needs of waveform developers; any number of applications could benefit from real-time services being included in the SCA. On the contrary, very few applications outside of the communication system domain would benefit from a Viterbi decoder.

To date there is limited material available in the SCA community that allows would-be SCA developers an affordable way to ramp up on the SCA in a short amount of time. There are a number of outstanding initiatives available such as the Open Source SCA Implementation: Embedded (OSSIE) [8] by the Mobile Portable Radio Research Group (MPRG) at Virginia Tech, and SCARI-OPEN Reference Implementation from Communication Research Centre [9]; and recently the SDR Forum contracted Mercury Computers Systems, Inc. to develop a reference waveform for SCA implementation. SCA-ready target choices are limited; however, core framework vendors are often willing to help get an OE running on any number of targets.

Within the SCA there are a number of observable design patterns such as Inheritance, Proxy, Delegation, Factory etc. A formal study of design patterns and how they best apply to Software Defined Radios and the SCA should be investigated.

REFERENCES

[1] JTRS JPEO, SCA version 2.2, http://jtrs.spawar.navy.mil/sca/

[2] JTRS JPEO, SCA Specialized Hardware Supplement version 3.0, http://jtrs.spawar.navy.mil/sca/

[3] Object Management Group, PIM and PSM for Software Radio Components, 1st FTF Convenience Document, dtc/2005-04-02, April 2005

[4] IT++, Welcome to IT++! , http://itpp.sourceforge.net/latest/index.html

[5] GNU Radio, GNU Radio - The GNU Software Radio, http://www.gnu.org/software/gnuradio/

[6] Wikipedia, XSL Transformations, http://en.wikipedia.org/wiki/XSLT

[7] JTRS Technology Laboratory (JTeL), Products, https://jtel.spawar.navy.mil/products.asp

[8]The Mobile and Portable Radio Research Group (MPRG) at Virginia Tech, OSSIE Open Source SCA, http://ossie.mprg.org/

[9] Communications Research Centre (CRC), SCARI – OPEN, http://www.crc.ca/en/html/crc/home/research/satcom/rars/sdr/products/scari_open/scari_open