Standardisation in Software Defined Radio

Amit Kumar

Abstract: ***Software Radio as a radio and communication technology has evolved with the evolution of digital electronics. It has undergone many changes in terms of technology and uses. With the advent and deployment of software defined radio (SDR) the radio is no longer the physical manufacturing of a single waveform but a computer host onto which different waveforms can be loaded. This paper provides an insight into the efforts to standardise the configuration and operation of software defined radio. The software communication architecture has been developed to assist in the development of SDR communication systems and captures the benefits of most recent technology advances to greatly enhance the interoperability of communication systems and reduce development and deployment costs.***

Keywords: Software Radio, Software Defined Radio, SCA, STRS, CORBA, ASI, POSIX, RTOS,

I. INTRODUCTION

SDR is a type of radio communication system where communication is carried out by the use of software on embedded system or personal computer instead of implementing hardware such as filters, amplifiers, mixers, detectors, modulators/ demodulators. SDR are capable of transmitting and receiving a wide spectrum of frequency. When the data from a source is converted into digital format, the remaining activities involved in radio are carried out with the help of software driven automated functions. SDR further optimizes the tactical information system as embedded software used in SDR helps in the dynamic selection of the communication channel. The number of digital service users is increasing resulting into the improved adoption rate of software defined radio. Public safety, military and commercial use are the three major end-use applications of SDR systems. The demand for SDRs is on the rise owing to efficiency, cost effectiveness and incorporation of latest technology supporting multiband, multiservice, multistandard and multichannel waveforms on a single platform.[1][2]

1.1. Issues for Adoption

Modernization programs being carried out by several countries such as South Korea, India, Germany, Japan, US etc and the interoperability provided by SDR are major driving forces for SDR. The issues faced in the integration of the various sub systems and the variety in each of the sub systems pose a challenge to the implementation of SDR. This is where standardisation plays a significant role in facilitating handshaking. Further, the development of software platforms, technologies and tools, which allow flexible specification, design and implementation of radio systems and its compatibility with the legacy systems, poses another significant challenge.

Software defined radio has potential opportunity in resolving the problem of frequency congestion in future. Within a SDR, the radio contains several processing elements (GPPs, DSPs, and FPGAs) that can be programmed by the waveform to deliver the required functionality. However, if each waveform must be tailored to the unique capabilities of each individual platform, (e.g., the type of GPP, DSP, and FPGA), significant portions of the waveform may have to be rewritten if they need to be ported to different hardware platforms. As a result, the move toward SDRs has prompted the development of open standards, to make it easier to develop waveforms that can run on multiple platforms with minimal change.

II. SDR STANDARDISATION

SDR standards must address three different parts of a waveform life cycle: (a) Describe the waveform - A waveform description should be entirely hardware independent but contain all the necessary information to implement the waveform on appropriate hardware. (b) Implement the waveform - Waveform implementation should be done in a relatively hardware-independent environment to facilitate portability to different hardware platforms. (c) Control the waveform- Once the waveform is operating, its features may be modified at runtime. Standardisation of SDR has been progressing for many years. The JTRS/SCA standard developed by the US Army [3] and STRS standard developed by NASA [4] define robust and powerful infrastructures for flexible radios.

III. SOFTWARE COMMUNICATIONS ARCHITECTURE AND JTRS

The largest single SDR development effort is spearheaded by the joint tactical radio system (JTRS) joint program executive office (JPEO) [5]. The JTRS program started in 1997 with the stated goal as development of a standard to facilitate reuse of waveform code between different radio platforms. The main standard developed by JTRS is the software communications architecture (SCA). SCA defines how waveform components are defined, created, connected together, and the environment in which they operate.[6] The standard was developed with a very software-centric view—an SCA-compliant radio was envisioned as a true SDR running on a GPP. The JPEO has further updated the SCA specification to SCA Next.

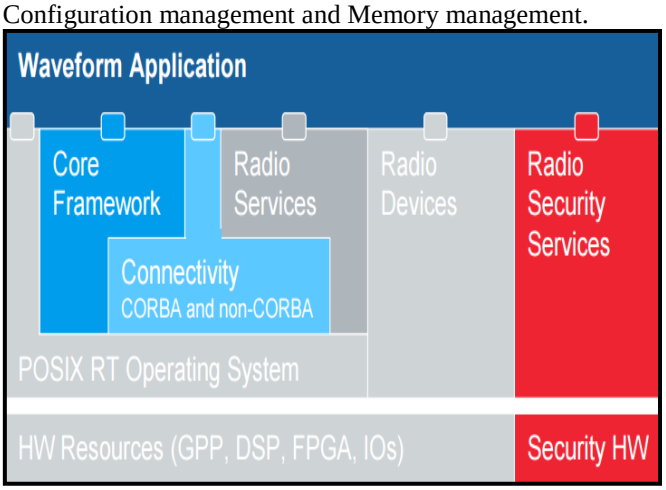
3.1. SCA Background

The SCA defines the following components. An operating system (RTOS) for the hardware that the radio runs on. The OS must be substantially POSIX compliant. A middleware layer takes care of connecting different parts of the radio together and handles data transfer between them. In the first version of SCA (2.2), the middleware layer had to be CORBA.

Revised Version Manuscript Received on May 18, 2016.

Mr. Amit Kumar, Department of Electronics and Communication, CBS Group of Institutions Affiliated to Maharshi Dayanand University, Rohtak (Haryana). India.

CORBA is an industrial-grade software architecture construct that was developed to facilitate communications between software written in many different languages and executing across multiple computers. The SCA Next standard removes the requirement that the middleware be CORBA and leaves it up to the radio developer. A set of interface definition language (IDL) classes provide an abstract interface to different objects that make up a radio (e.g., IO ports, components, devices, etc.). These classes, together with software code that enables their use, are known as the core framework (CF). An XML ontology (Domain Profile) describes all the components that make up a radio and how these components are to be interconnected and executed to implement the desired waveform. APIs are used for many frequently used interfaces (e.g., audio, Ethernet, etc.). SCA is also designed to address security requirements of military radios and has built-in support for red/ black (secure/open) separation. SCA has numerous key features that can be organized into four major categories: Software architecture, Hardware architecture, Security architecture and Application program interfaces (APIs). The software architecture is based on an embedded distributed computing environment mainly defined by an operating environment (OE), applications, and logical devices. The OE itself is comprised of a real-time operating system (RTOS), a real-time object request broker (ORB), core framework (CF) defined by the SCA, and services. The RTOS utilizes a POSIX 2 -based processing infrastructure in order to enable full JTRS functionality. The interfaces defined by the SCA for the ORB are based on a subset of CORBA 3 (common object request broker architecture) in order to seamlessly integrate and enable the interoperability of various software and hardware components from the same or various vendors. The hardware platform is divided into three main sections: the black section where the data is encrypted between the various interfaces, the crypto section that encrypts and de-encrypts the data going from the red section to the black section and vice versa, and the red section where the data flows in de-encrypted format between various devices. The security architecture is comprised of various security elements of the SCA that are relevant to both commercial and military applications. The various security requirements can be classify ed into key categories of- Encryption and decryption services, Information integrity, Authentication and no repudiation, Access control, Auditing and alarms, Key and certification management, Security policy enforcement and management, Configuration management and Memory management.



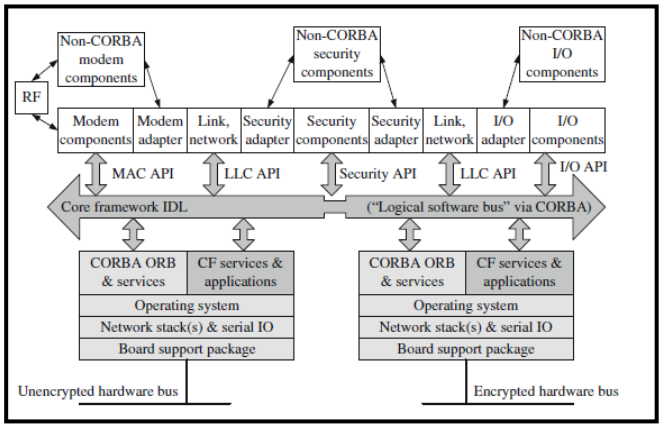


Fig1-Conceptual hierarchy of SCA components

3.2. CORBA

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 aims to provide hardware and location transparency to software entities. It functions as a conversion and translation layer. It is a consolidator and integrator. With the help of middleware, software applications running on different platforms can communicate transparently. CORBA is used as the underlying middleware. CORBA has been chosen as the middleware layer of the Software Communications Architecture, because of the wide commercial availability of CORBA products and its industry acceptance. Distributed processing is a fundamental aspect of the JTRS system architecture. 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 . CORBA is the Object Management Group's open architecture that provides the infrastructure for computer applications to work together over a network.

A large and computationally intensive software program runs on many processors. Some of the processors are directly connected to each other (perhaps even on the same chip), while others are in a different chassis. Routines executing on different processors need to communicate with each other. CORBA is one implementation of a distributed object system [7]. The key to CORBA is an object request broker (ORB) that is responsible for data transfer between and execution of software objects residing on possibly different processors. An object is described using interface description language (IDL)—specifying the inputs, outputs, and methods. The IDL description is compiled by a tool provided by the developer of the ORB to create two wrappers: one for the client side, called a stub, and one for the server side, called a skeleton. The client and server may use different languages and the IDL compiler can generate wrappers for each language (e.g., C, C++, Python, JAVA, etc.). The ORB itself may be written in any language. The ORB executing on each processor may come from a different vendor.

The 'object skeleton' contains auto-generated code to support data transfer, setting parameters, and calling of methods. CORBA supports powerful features that allow software objects to find each other at runtime by querying the domain name service. The ORB abstracts the communications medium. While most CORBA implementations rely on the standard TCP/IP network stack for low-level communications, this is not a requirement. ORB works just as well over PCI-express, or a custom hardware interface. A sophisticated ORB can detect when the two objects are executing on the same processor and use shared memory for communications. IDL descriptions are automatically generated by tools such as OSSIE that allow the user to think in terms of IO ports and data types that are meaningful to radio designers. CORBA is intended to be implemented on GPPs. However, many SDRs include non-GPP devices such as FPGAs and DSPs. These devices are not suitable for executing a full-featured ORB and therefore cannot fully participate in the ORB infrastructure. SCA addressed non-GPP hardware by defining an adaptor component. This component executes on the GPP and translates CORBA messages to modem hardware abstraction layer (MHAL) messages. The MHAL messages are simple packets with the packet format left entirely up to the developer. The SCA Next standard specifies two new CORBA profiles: (a) Lightweight Profile for SCA applications hosted on resource-constrained platforms such as DSPs (b) Ultra-Lightweight Profile targeted at applications hosted on FPGAs. These profiles support a small subset of CORBA functions and data types.

3.3. Controlling the Waveform in SCA

SCA provides two mechanisms to control an executing waveform: 1. Low-level interface to each component via the configuration properties accessed by the Configure method. 2. High-level (Domain Manager) interface to load, start, stop, and unload a waveform. Each waveform is described by a different SAD file. The low-level interface can be used to control a few runtime parameters such as the symbol rate or modulation. However, it is not suitable for large-scale changes that require new components to be loaded and swapped for currently executing components. Fast waveform changes were not considered during the development of the SCA standard. Unloading one waveform and loading another is relatively time-consuming, especially on a resource-constrained device. Even retrieving the code from non-volatile memory can take a long time. The latency incurred during this process may be unacceptably high. One obvious solution is to develop a super waveform that contains the code for all the waveforms of interest. As far as the SCA OE is concerned, only one waveform, called 'Super-Waveform', is loaded. The 'waveform selector' component is configured using the low-level configuration interface. Since all the components are simultaneously resident, they all consume memory. Note that the code for each of the components is only loaded once and can be shared among all the waveforms (i.e., if the same component is used for phase tracking in all the waveforms, it is only loaded once). However, each component does allocate its own runtime data.

3.4. SCA APIs

SCA compliance is not sufficient for easy waveform portability. Consider an application (waveform) that relies on a specific and unique audio interface that is only available on one hardware platform. Even if the application is SCA compliant, it would be quite difficult to port to another platform. To address this problem, the OE defines a set of standard interfaces to frequently used devices. A layer of standardized APIs is placed between the application and hardware. The APIs, just like the rest of SCA components, are described using IDL. The hardware vendor must provide a Radio Device which translates between the JTRS API and the low-level, hardware specific API. A set of fundamental, abstract APIs define interfaces common to a wide range of more specific APIs. The specific APIs are derived from the primitive APIs.[8]

3.5. Comparison of SCA Variants

SCA 1.0 was the first official version released in 2000. SCA 2.2 was specified for yhe first radio product released in 2001. SCA 2.2.1 was an interim version released in 2004. SCA 3.0 was an innovative interim version which was cancelled shortly afterwards. SCA 2.2.2 is most stable and modern version in use since 2006. SCA 4.0 is an improved version not yet in use and SCA 4.1 is the most improved version with backward compatibility to 2.2.2. SCA 2.2.2 is currently the best suited solution for powerful and secure SDRs. All major programs like the US JTRS, the German SVFuA and the European ESSOR are based on the SCA 2.2.2. The technical benefits include portable waveform applications, modularity enabling reuse of waveform components, scalability and ease of integration of new features. Issue which is nor being optimally addressed by the SCA 2.2.2 is the lack of support for small battery powered systems. SCA 4.0 was released in 2012 and was the first major release in six years. Major enhancements in comparison of 2.2.2 includes support for incorporation of additional software and hardware platforms via profiles, permits static component connection, supports nested waveforms and interconnections and incorporates technology advances. But porting an existing SCA 2.2.2 to an SCA 4.0 compliant SDR platform would have required substantial rework. SCA 4.1 provides crucial edge over SCA 2.2.2. SCA 4.1 provides substantial advances of SCA 4.0, solves the backward compatibility issue and includes further improvements like the lightweight profiles.

IV. STRS

SDR is very attractive for use in satellites and deep space missions. The radios on space platforms have very long life cycles. Changing the communications waveform can extend the life of a mission, increase the amount of returned data, or even save the mission from failure. NASA decided to develop a lightweight standard modeled loosely on the JTRS standard, and called it 'Space Telecommunications Radio System' or STRS.[9]

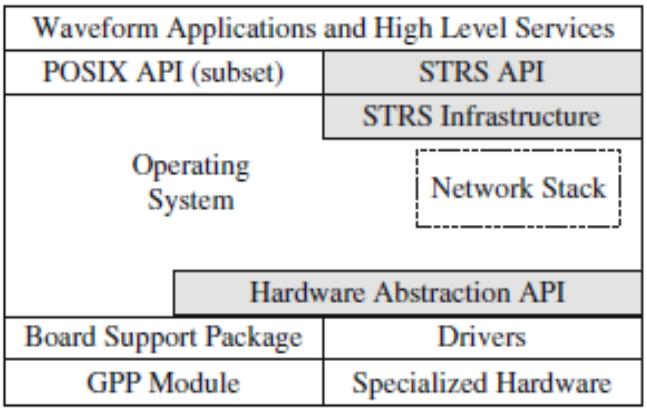


Fig 2-Conceptual hierarchy of STRS

In comparison it clearly demonstrates that STRS is significantly simpler than SCA. Perhaps the largest difference is that no middleware layer (i.e., CORBA) is used for STRS. The bulk of the STRS standard deals with APIs. Unlike SCA, the STRS standard also deals with hardware implementation of the radio. JTRS developers were mostly concerned with application and waveform portability. STRS developers also want to be able to swap hardware boards between radios or add new ones. STRS explicitly addresses the 'signal processing module (SPM)' made up of ASICs, FPGAs, and DSPs that do not execute software. In fact, the GPP is meant for control only and is not expected to do any signal processing at all. The set of APIs defined by STRS is much smaller than for JTRS. Most of the required APIs deal with the infrastructure and would be considered a part of the Application Factory in JTRS.

V. RESULTS

A fundamental challenge of SDR is to provide an ideal platform to application separation, such that waveform applications can be moved from one SDR platform to be rebuilt on another on without having to change or rewrite the application. SCA contributes to such application portability by providing a standard for deployment and management of SCA based applications. It also standardizes the interconnection and intercommunication both between the components of the application, and between components and system devices. Significant pieces that are not standardized by the SCA itself are the APIs to the services and devices of the system platform. In order for portability to extend across domains, the APIs to the services and devices will need to be standardized across domains as well. Comparisons of various versions of SCA reveal interesting future prospects for SCA 4.1 with backward compatibility.

VI. CONCLUSION

This paper has presented the aspects of standardisation in software defined radio. The software communication architecture has contributed in reducing the complexity of the technology by bringing the waveforms on a common platform and in a structure conducive to interaction. Further efforts in developing the architecture would significantly contribute towards reducing the multiple platforms of communication.

REFERENCES

1. J. Mitola III, "Cognitive radio: An integrated agent architecture for software defined radio." PhD thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, May 2000.
2. www.wirelessinnovation.org
3. JPEO JTRS (2011) SCA: application program interfaces (APIs) (http://sca.jpeojtrs.mil/api.asp)
4. Reinhart RC et al (2007) Open architecture standard for NASA's software-defined space telecommunications radio systems (Proc IEEE 95:1986–1993)
5. JPEO JTRS http://sca.jpeojtrs.mil/home.asp
6. Bard J (2007) Software defined radio: the software communications architecture. Wiley, New York
7. Ciaran McHale, CORBA explained simply http://www.ciaranmchale.com/corba-explainedsimply
8. Eugene Grayver," Implementing Software Defined Radio"
9. Reinhart RC et al (2010) Space telecommunications radio system (STRS) architecture standard. NASA glenn research center, Clevelend, TM 2010-216809