OPEN SPACE RADIO: AN OPEN SOURCE IMPLEMENTATION OF STRS 1.01

Sahana B. Raghunandan (MPRG, Wireless@Virginia Tech, Blacksburg, VA, USA; sahanarb@vt.edu);
Dileep Kumaraswamy (MPRG, Wireless@Virginia Tech, Blacksburg, VA, USA; dileep@vt.edu);
Lillian Le (MPRG, Wireless@Virginia Tech, Blacksburg, VA, USA; lle86@vt.edu) ;
Carl B. Dietrich (MPRG, Wireless@Virginia Tech, Blacksburg, VA, USA; cdietric@vt.edu); and
Jeffrey H. Reed (MPRG, Wireless@Virginia Tech, Blacksburg, VA, USA; reedjh@vt.edu).

ABSTRACT

The Space Telecommunications Radio System (STRS) details high level specifications for the development, testing operation and maintenance of software defined radios (SDR) used by NASA for space communications. As it provides a broad framework for SDR development across different mission classes, radio designers can apply this open architecture to their individual requirements and build an end-to-end system. In this paper, we describe an open source implementation based on an excerpt of the STRS 1.01 standard and its applications. The development of Open Space Radio was initially motivated by the need for an STRS based proof-of-concept system for a space application being developed by AeroAstro, Inc. This proof-of-concept space radio system involves interfacing the General Processing Module (GPM) with a Signal Processing Module (SPM) based on a commercial FPGA board. It is hoped that the availability of Open Space Radio will facilitate SDR research for space applications.

1. INTRODUCTION

Software defined radios provide a flexible radio architecture with features such as multifunctionality, mobility, compactness, power efficiency, ease of manufacture and upgrade [1]. In addition to these features, one of the major advantages that SDR technology renders to space-based radios is the ability to communicate with different stations without the inclusion of multiple radios for each communication end point [2]. In order to fully harness all these capabilities, a robust software architecture is quintessential. The STRS aims at lending a unifying framework that embraces a systems approach to develop, test, operate and maintain reconfigurable space communication and navigation assets. Although STRS targets future space communication system needs, it provides high level abstraction for reuse of existing hardware and software components.

At the system level, classifying the radio functions into objects can aid in portability and maintenance and has proved as one of best practices in SDR implementations to date. *Open Source SCA Implementation - Embedded* (OSSIE), a SDR framework based on JTRS (Joint Tactical Radio System) Software Communication Architecture (SCA), developed at Virginia Tech is a good illustration of this concept [3]. The hierarchical model that is a key concept in an open SDR architecture provides partitioned software modules controlled by managing software [4]. Maintaining compatibility with existing SDR architectures, the STRS standard prescribes the relationship between software components instrumental in software execution and defines the Application Programming Interface (API) between the Operating Environment (OE) and the waveform application [5]. The STRS infrastructure, which is a part of the General Purpose Processor (GPP) OE, provides functionality to the interfaces defined by this API and supports waveform operations upon deployment [6]. A STRS waveform is a term used to refer to an executable software or firmware application that is abstracted from the radio platform [4]. The initial release of the STRS open architecture gives a complete insight into the various hardware and software modules and lends itself as an open standard with a dynamic structure that can be updated as required.

The development of Open Space Radio was inspired by the need for an STRS based proof-of-concept radio system for a space application being developed by AeroAstro,Inc. The aim of this effort is to leverage STRS as an open standard that can facilitate research and implementation of SDR components for space applications. The focus of this initial implementation has been on the development of STRS infrastructure and STRS API for waveform instantiation. A light weight web server has been integrated into the current system to control waveform initialization and termination.

2. SYSTEM DESCRIPTION

The STRS architecture allows for a modular radio design with a choice of hardware implementation that entails the functional attributes defined in the specifications. The proposed hardware architecture that is based on reconfigurable elements consists of: (a) General Purpose Processing Module (GPM); (b) Signal Processing Module (SPM); (c) Radio Frequency Module (RFM); (d) Security

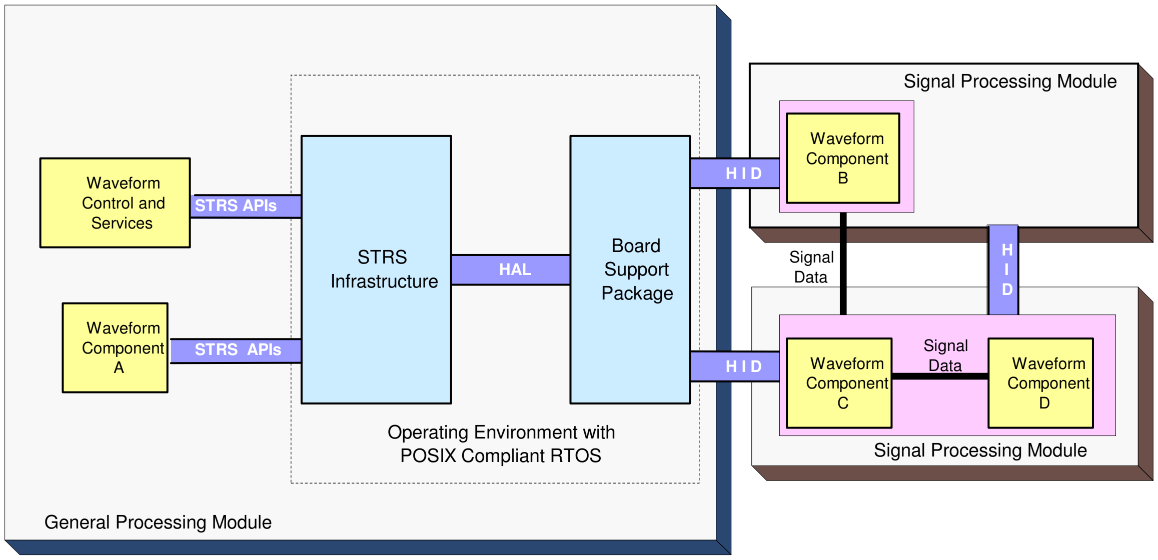

Figure 1 Waveform Component Instantiation [7]

Module (SEC); (e) Network Module (NM); and (f) Optical Module (OM). The instantiation of a waveform component on an example radio platform consisting of the GPM and the SPM is depicted in figure 1. The proof-of-concept radio system is being designed in a similar manner to support loading and execution of a waveform. A brief description of the same has been provided below.

2.1. General Processing Module

This module may consist of a GPP and memory elements such as a persistent memory storage element and work area memory element. It needs to incorporate a system control element that arbitrates the system bus and provides necessary interfaces to the SPM or RFM. In this project, a single board computer has been chosen to serve as the GPM.

The focus of the research team at Virginia Tech has been on the design of GPM functionality that encompasses configuration and control of the STRS architecture. This has mainly revolved around the development of the STRS infrastructure that includes device control and waveform management as illustrated in figure 1. On one end the STRS infrastructure provides services essential to load, verify, execute, modify parameters and unload the waveform, and on the other end it establishes communication with the specialized hardware through the Hardware Abstraction Layer (HAL) [2].

The Hardware Interface Definition (HID) as the name suggests indicates how the various hardware modules within the radio platform are physically connected. The Board Support Package (BSP) that furnishes abstraction of the specific hardware module is integrated into the OE to interface with the infrastructure and coordinate device configuration. This particular kernel level entity is currently being provided by the team at AeroAstro, Inc, whose focus has been on the design of SPM using Field Programmable Gate Arrays (FPGAs) on a dedicated Peripheral Component Interconnect (PCI) board.

2.2. Signal Processing Module

The SPM contains modules that manipulate the bit stream from and to the GPM. A FPGA, a Digital Signal Processor (DSP) or an Application Specific Integrated circuit (ASIC) or a combination of the above can be used to process the received signals from the A/D converter as well as generate the transmit signals to the D/A converter. The use of FPGAs enhances system reconfigurability and permits the addition of new signal processing functionalities to the SDR without redesigning hardware.

A multichannel transceiver with Virtex-4 FPGAs from Pentek, Inc [8] is being used to develop the SPM. The digital transceiver has on board memory resources, four A/D converters and one digital upconverter capable of handling baseband complex and real signals. The FPGA (Xilinx XC4VSX55) that interfaces with these modules contains DSP slices which are ideal for implementing digital communication functions such as digital modulation/demodulation, carrier recovery and synchronization, scrambling/descrambling and encryption/decryption. The PCI 2.2 bus specification that is a part of the HID resides on the second Virtex-4 FPGA (Xilinx XC4VFX60/LX100), which additionally includes two PowerPC cores. The currently undertaken tasks at AeroAstro,Inc, involve the customization of the FPGAs to incorporate signal processing components that meet the requirements of the space application.

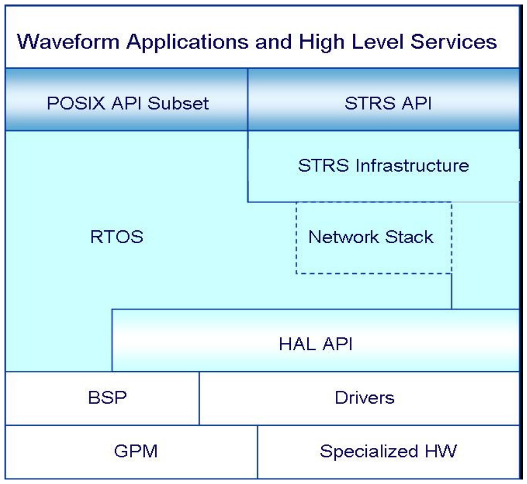

Figure 2 Software Execution Model [7]

3. SOFTWARE ARCHITECTURE

The software architecture proposed in STRS specifications defines interfaces and the relationship between the various software executable components in an STRS compliant radio [7]. The objectives of the software execution model depicted in figure 2 includes: (a) application waveform abstraction from OE using custom routines made available through standard interfaces such as Portable Operating System Interface (POSIX) ; (b) lower level abstraction between OE and hardware platform and (c) layering of software components to elucidate the relationship among different levels of abstraction [7].

The STRS API, which is the focus of the open space radio implementation here, provides an open software specification that is described as a *"well-defined set of interfaces used by the waveform applications to access specific radio functions or used by the infrastructure to control the waveform applications"* [6]. These interfaces are standardized for use with any infrastructure platform, which could include a combination of POSIX, RTOS or BSP, to allow portability. The fact that the STRS API decouples the intellectual property rights of the platform and module developers makes it viable for the development and interoperation of different mission-specific components of space radio [6].

4. IMPLEMENTATION

The STRS standard specifies the minimum API definition required to execute space radio applications and deliver control and data packets to the installed hardware components. Nonetheless, there are several use cases and APIs out of which a subset required for the proof-of-concept system has been implemented here. A Unified Modeling Language (UML) class diagram generated from the C++ open source code implementation is depicted in figure 4. It

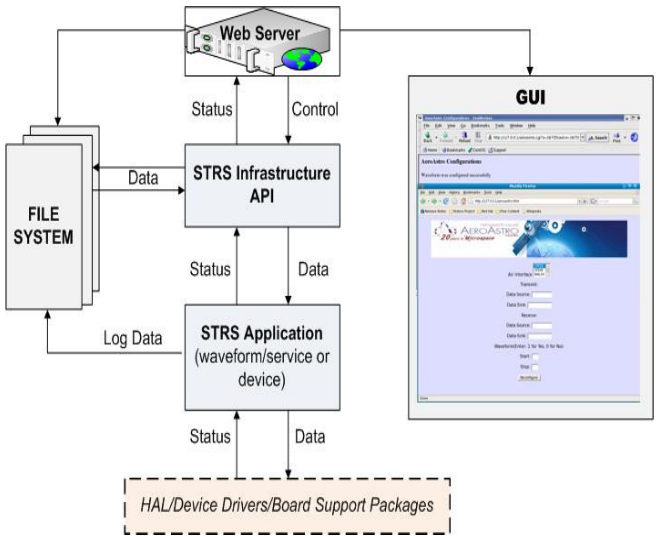

Figure 3 Functional Diagram of STRS API Implementation

illustrates the inheritance between the classes and the corresponding implementation objects.

Some of the key elements of the standard required for the proof-of-concept system have been implemented, viz., (a) STRS Application Control API; (b) STRS Infrastructure Application Control API; (c) STRS Infrastructure Application Setup API ; and (d) STRS Infrastructure Memory API. The STRS Application Control API is similar to a Resource Interface in the SCA specifications. Every STRS application which could be a waveform, service or device is required to implement the methods included in this API. The *STRS_ApplicationImpl* class is a good example of this implementation. The STRS Infrastructure Application Control API has methods corresponding exactly to STRS Application Control API, which are used to access methods in the latter API. This is done using a handleID which is a unique identifier which can also be used to obtain access to resources such as devices, files, or message queues. Control of one waveform from another and logging application status are handled by STRS Infrastructure Application Setup API. Tasks involving memory management and manipulation are done by STRS Infrastructure Memory API. The other class that is depicted in figure 4 is *STRS_Device*. A STRS Device has been defined as *"a proxy for the data and/or control path to the actual hardware"* [7]. It is a STRS Application that may use interfaces available in HAL to connect to the actual specialized hardware.

The sample implementation of a STRS waveform application has been executed to test the instantiation of objects and execution of the participating software packages. A successful demonstration of simultaneous instantiation of multiple waveform components using POSIX threads has been done. One of the other highlights of the system was the integration of a light weight web server, thttpd, [9] to support a graphical user interface (GUI) that can control the execution of a waveform. An overview of the current application set up is shown in figure 3. For the purpose of parsing waveform specific and platform specific configuration files represented in eXtensible Markup Language (XML), the use of the API provided by TinyXML is proposed [10].

5. SOFTWARE DEVELOPMENT METHODOLOGY

5.1. STRS Compatibility Approach

To ensure compatibility, the STRS software development is based on UML diagrams and function descriptions provided in Section 9 of STRS 1.01 [7]. Where possible, the pseudocode provided in STRS 1.01 document has been used as a starting point for the code.

5.2. Development Team using Tested Methodologies and Tools

The STRS implementation uses most of the tools and methodologies that have been used for the development of OSSIE [3]. As in the OSSIE project, extensive use is made of open-source software development tools.

5.3. Revision Control and Trac Wiki

Subversion, an open-source revision control system, is being used to store the STRS source code and track changes. This allows multiple developers to collaborate on the code development. Tasks are assigned to minimize overlap so that the developers' edits to the code are unlikely to conflict with one another. In the event of a conflict, the subversion

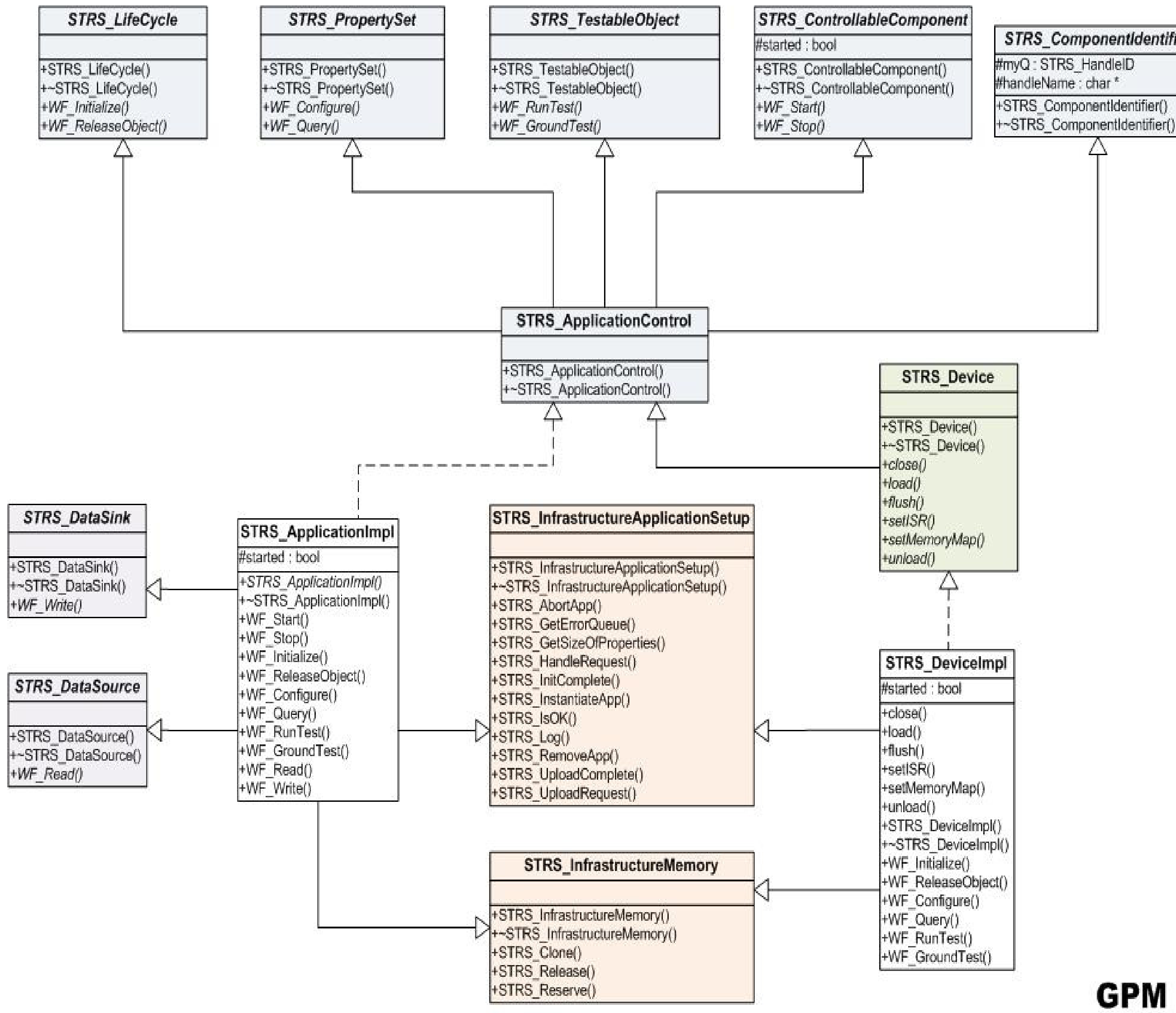

Figure 4 Class Inheritance Structure in Open Space Radio

system can be used to identify specific edits that need to be harmonized.

A Trac Wiki system is interfaced with the subversion repository which allows browsing of the source code and tracking of edits using a web browser.

6. CONCLUSION

The implementation described in the paper has been successfully compiled and results indicate that it aligns well with the requirements laid out for the demo system. The exclusive use of open source tools has made it possible to reinstate the fact that this architecture lends itself as an open standard that can be used to encapsulate proprietary software and hardware modules. The modularity of this open architecture is established through the implementation and ability to integrate additional features such as the GUI. With the availability of STRS compliant hardware resources, it is hoped that this effort will enable rapid development of mission specific waveform application and services.

7. FUTURE WORK

The next step would be to integrate the STRS Application and Infrastructure implementation residing on the GPM with the SPM through the HAL layer. Next, the integration testing of the proof-of-concept system would be done to ensure it fulfills all the requirements. As far as the API implementation is concerned, there is scope for extending the existing functionality to include a networking module and management units for messaging, security and overall system health monitoring. Compliance testing and architecture verification techniques can be investigated and incorporated into the current software execution model. Improvisation on the GUI and web server can be done to port the application to an embedded device. With these enhancements, a number of radio platforms spanning different mission classes can be updated, tested and operated.

REFERENCE

[1] J.H.Reed, *Software Radio: A Modern Approach to Radio Engineering,* Pearson Education, New Jersey, USA, 2002.

[2] "Space Telecommunications Radio System Open Architecture Standard," Revision 1.0, National Aeronautics and Space Administration, April 2006.

[3] http://ossie.mprg.org/

[4] "Space Telecommunications Radio System Open Architecture Description," National Aeronautics and Space Administration, April 2006.

[5] "Space Telecommunications Radio System Open Architecture Use Cases," National Aeronautics and Space Administration, April 2006.

[6] T.J. Kacpura, L.M. Handler, J.C. Briones and C.S. Hall, "Updates to the NASA Space Telecommunications Radio System (STRS) Architecture," January 2008.

[7] "Space Telecommunications Radio System Open Architecture Standard," Revision 1.01, National Aeronautics and Space Administration, June 2007.

[8] http://www.pentek.com/Products/Detail.cfm?Model=7642

[9] http://www.acme.com/software/thttpd/

[10] http://www.grinninglizard.com/tinyxml/

[11] R.Lafore, *Object Oriented Programming in C++*, 4th Edition, Sams Publishing, U.S.A, 2002.

[12] M.T.Jones, *GNU/Linux Application Programming*, Charles River Media, U.S.A, 2005.

ACKNOWLEDGEMENT

This work was supported by AeroAstro, Inc. Shereef Sayed from MPRG, Wireless@VirginiaTech, provided valuable inputs and feedback during the course of this implementation.