CONSIDERATIONS FOR THE NEXT REVISION OF NASA'S SPACE TELECOMMUNICATIONS RADIO SYSTEM ARCHITECTURE

Sandra K. Johnson (NASA Glenn Research Center, Cleveland, OH)
Sandra.k.johnson@nasa.gov
Louis M. Handler (NASA Glenn Research Center, Cleveland, OH)
Louis.m.handler@nasa.gov
Janette C. Briones (NASA Glenn Research Center, Cleveland, OH)
Janette.c.briones@nasa.gov

ABSTRACT

Development of NASA's Software Defined Radio architecture, the Space Telecommunication Radio System (STRS), was initiated in 2004 with a goal of reducing the cost, risk and schedule when implementing Software Defined Radios (SDR) for National Aeronautics and Space Administration (NASA) space missions. Since STRS was first flown in 2012 on three Software Defined Radios on the Space Communication and Navigation (SCaN) Testbed, only minor changes have been made to the architecture. Multiple entities have since implemented the architecture and provided significant feedback for consideration for the next revision of the standard.

The focus for the first set of updates to the architecture is items that enhance application portability. Items that require modifications to existing applications before migrating to the updated architecture will only be considered if there is compelling reasons to make the change. The significant suggestions that were further evaluated for consideration include expanding and clarifying the timing Application Programming Interfaces (APIs), improving handle name and identification (ID) definitions and use, and multiple items related to implementation of STRS Devices. In addition to ideas suggested while implementing STRS, SDR technology has evolved significantly and this impact to the architecture needs to be considered. These include incorporating cognitive concepts - learning from past decisions and making new decisions that the radio can act upon. SDRs are also being developed that do not contain a General Purpose Module – which is currently required for the platform to be STRS compliant.

The purpose of this paper is to discuss the comments received, provide a summary of the evaluation considerations, and examine planned dispositions.

1. INTRODUCTION AND BACKGROUND

To reduce the cost, risk, and schedule for developing the emerging software defined radios for space missions; NASA began a task in 2004 to develop a standardized architecture to abstract waveforms from the SDR platforms. The development of a new waveform for a space SDR is more expensive then a ground based development, partially due to the significant documentation and test processes required to insure reliability in a space environment.

The Software Communication Architecture (SCA) [1] and Object Management Group (OMG)'s SWRADIO [2] were investigated as the potential architecture for NASA's needs for space SDRs [3], [4]. For a constrained space environment, the required resources (size, weight, and power) of the SDR must be a primary consideration for the architecture. Processors and other electronic devices used in space require radiation hardening. These components lag at least a generation or two behind the processing capabilities of their terrestrial-based equivalents. Due to slower processors and limited memory footprint, these reduced capabilities constrain the operating environments of space radios compared to radios using the latest commercial components without these constraints.

Space waveform applications requiring digital signal processing have historically been executed in specialized Application Specific Integrated Circuits (ASICs). ASICs have the lowest power requirements and greatly satisfy radiation requirements for space. However, they are not reprogrammable. Reconfigurable signal processing continues to evolve and gain acceptance within NASA with the availability and use of space-qualified Digital Signal Processors (DSPs) and Field Programmable Gate Arrays (FPGAs).

There are other challenges that are factors implementing SCA on a space radios. The SCA compliant core framework must fit on the space-qualified platform in terms of resources, footprint, and features. If the SCA is used to provide an environment where radio capabilities can be reprogrammed, the necessary core framework will take up resources that would normally be dedicated directly to

signal processing. The SCA in space also has to address concerns with the added software complexity and the extended time for testing required insuring reliability in a space environment. The use of DSPs and FPGAs are treated as special cases in SCA 2.2. SCA 4.1 improves the situation regarding waveform components implemented on FPGAs and DSPs by allowing lightweight profiles and other recommendations to optimize their extremely limited processing capabilities.

Due to these differences in the needs for space radios, the STRS architecture was developed to focus on the abstraction layer between the operating environment (OE) and applications to enhance portability, reliability, scalability, extensibility, flexibility, adaptability as well as to minimize size, weight and power (SWaP) necessary to be launched into space and executed in a space-based environment. Unlike the SCA, the minimal set of requirements developed for STRS did not include Common Object Request Broker Architecture (CORBA) nor an EXtensible Markup Language (XML) parser.

In 2012, STRS was implemented on three SDR platforms and installed on an external pallet on the International Space Station (ISS) as part of the SCaN Testbed [5]. All platforms and waveforms on the SCaN Testbed implemented STRS Version 1.02.1, the baseline version of the STRS architecture [6]. These SDRs have been reconfigured multiple times with new waveforms. The SDR platforms were developed by three separate entities: Harris Corporation, General Dynamics, and Jet Propulsion Laboratory. Multiple research projects within NASA have also developed STRS compliant applications and platforms and provided suggestions for improvement

In 2014, the STRS architecture standard was approved by the NASA Office of Chief Engineer as an official NASA standard [7]. Changes from STRS Version 1.02.1 to the NASA standard version (NASA-STD-4009) were primarily editorial, formatting, and clarification. A supporting Handbook, containing implementation suggestions, rationale for architecture decisions and addressing many implementation questions, has also been released [8].

Questions and comments from developers were captured to influence the next updates to STRS. The STRS Handbook is also in the process of being updated. A Project and Acquisition Guidance document is in development to address items that are not architecture related but instead must be considered when implementing an STRS platform and/or application.

2. DETAILED DESCRIPTION OF CONSIDERED UPDATES

NASA received approximately 100 comments on the STRS architecture that ranged from simple editorial changes to suggestions requiring significant changes to the architecture. The STRS project goal is to have a new release of STRS in mid 2017. To meet this schedule, the architecture updates must be completed for review by the end of 2016, requiring prioritization and focus on only a few key areas.

Potential updates were considered primarily on their potential to enhance application portability. Additional updates to clarify existing requirements were also considered. Because multiple platforms and applications exist, suggestions that require modifications to existing applications or operating environments but did not have compelling rationale for enhancing application portability were not selected. The following sections contain a discussion of the key areas under consideration for updates or significant clarification.

2.1. Devices

The definition and use of STRS Devices evoked many formal comments and informal queries from STRS platform developers. Most of this confusion comes as a result of comparing an STRS Device to an application. An STRS Device is an extension of an STRS application having a set of portable APIs that may use the Hardware Abstraction Layer (HAL) to read, write, and control hardware devices. An STRS application must be able to use the STRS Devices in a standard way. The OE interfaces to the physical devices are defined by the platform provider, are not standard, must be documented in the Hardware Abstraction Layer (HAL), and are used by the STRS Devices.

Although the SCA has many types of specific device interfaces, STRS has only one. The SCA requires specific APIs and services that are required by most Joint Tactical Radio sets. NASA has a different set of use cases where most of the signal transformations happen in FPGAs, not software. In the SCA, the Modem Hardware Abstraction Layer (MHAL) provides interfaces to Computational Element (i.e. GPP, FPGA, or DSP) by abstracting the channel modem interfaces from the application software via an MHAL API [9]. The STRS architecture accomplishes the same thing using STRS Devices as the standard interface to the application.

An STRS Device is defined in the STRS Architecture Standard as "a proxy for the data and/or control path to the actual hardware." An STRS Device is a "bridge" used to decouple an abstraction from its implementation so that the two can vary independently. An STRS Device is called using the methods in the STRS Infrastructure Device Control API, STRS Infrastructure-provided Application Control API, Infrastructure Data Source API, and Infrastructure Data Sink API to control the STRS Devices. The STRS Device implementation is suggested in Figure 1. The STRS Device may be implemented using any available platform-specific HAL to communicate with and control the specialized hardware.

Standardizing STRS Devices with required STRS Device-Provided Device Control API methods was suggested to aid in the portability of STRS devices between STRS operating environments. This could be implemented by adding a corresponding set of "STRS Device-provided Device Control API" calls analogous to the already defined "STRS Application-provided Application Control API" calls (APP_*) to provide an abstraction between STRS Devices and the operating environment

This approach would reduce the effort to port devices between operating environments and may reduce confusion when implementing devices, but this change will not be implemented for several reasons. The STRS architecture was not intended to simplify porting of STRS devices and implementing this suggestion would cause existing platform providers to change quite a bit of code. It also adds unnecessary complexity and could add extra burden to the process of accessing the device, which could affect performance.

An STRS Device may or may not be part of the OE depending on the project/mission requirements. The intention was that the platform provider creates a sample application that uses an STRS Device to exercise both the hardware and software for testing and that serves as a model for what can be done by the application developer. An STRS Device could be specified in the OE, for example, if the device is not really programmable but can be adjusted or turned on/off by the HAL.

Configuration and additional functionality for an STRS Device also required additional clarification. An STRS Device is a virtual device that has capability beyond that of an STRS application. There are 2 types of additional capability of an STRS Device beyond that of an STRS application: 1) STRS Device capabilities that correspond to STRS Infrastructure-provided Device Control methods, and 2) STRS Device capabilities that do not correspond to the STRS Infrastructure-provide Device Control methods, such as setMemoryMap(map). These are suggested in the STRS Architecture Standard as shown in Figure 1. The setMemoryMap() is suggested to enable the OE to put special configuration data into a table within the STRS Device to enable use of special memory locations to transfer

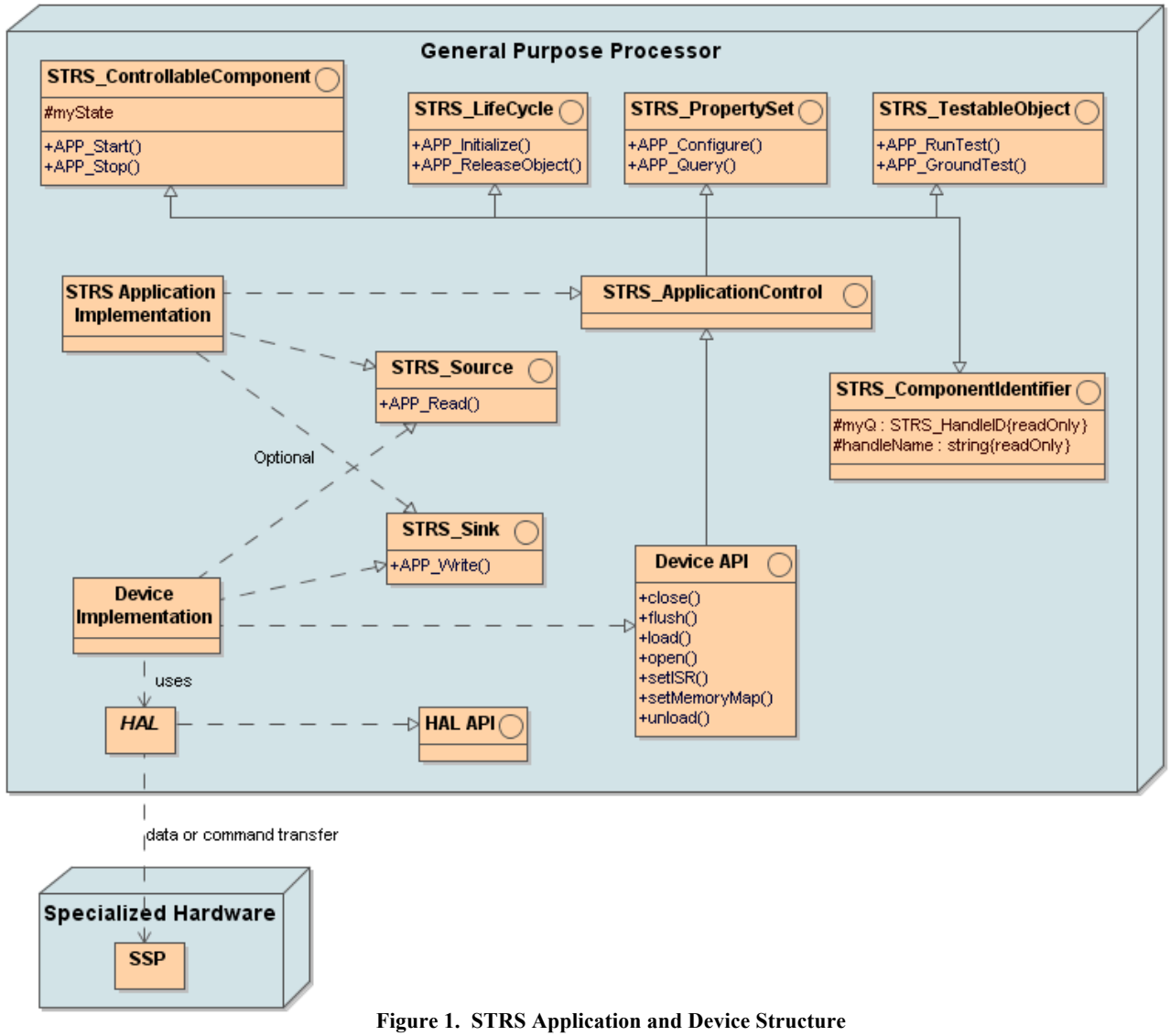

Figure 1. STRS Application and Device Structure

data between the STRS Device and the specialized hardware, e.g. FPGA. The location of some named data might not be specified as just a name/location, which could be configured by a normal name/value pair, but rather as a name, base location, location offset, bit offset, and bit length.

Although portability of STRS devices is not a goal of STRS, STRS platform developers commented that STRS device portability would be enhanced if there were more rigorous standards for how an application developer would use the STRS API to process the various types of data that must be sent or retrieved from the corresponding addresses in specialized hardware; e.g. FPGAs, DSPs, etc. Many SDRs use memory mapped locations in which storing/retrieving an item in memory automatically pushes/pulls the item to/from the specialized hardware. Other SDRs use special APIs to send/retrieve the item to/from the specific address in the specialized hardware. The STRS Device was created to be a bridge to abstract all of these differences, insulating the waveform application developer from knowing how the data got to its final destination, thus aiding the portability of waveform applications without restricting the platform developer's flexibility.

STRS Devices are allowed to know the memory map and are not required to be portable. A configuration file for the memory locations is suggested in the STRS standard (Appendix A.2) and Handbook as a way of allowing the FPGA to be reprogrammed with a change in the location for a particular purpose without searching and changing the locations hard-coded in the STRS Device. This is not a requirement. An alternative is to define named constant values that could all be changed in one place such as the STRS Device header file. Since the STRS Device is part of the OE, this breaks the strict division of what an application developer should need to know or do.

Other potential updates suggested for the use of STRS Devices included creating a new STRS_DeviceRead and STRS_DeviceWrite which specifies the location information as well as the storage area for the data. The location, offsets, and size could be defined as arguments. Although this might simplify the porting of STRS Devices, it is not required to enable application portability and the complexity and change for this capability is not critical so will not be added to the next STRS version.

2.2. Handle Names and IDs

Comments and discussions related to the uniqueness and time of existence of Handle Names and Handle ID have resulted in changes in the STRS architecture to the clarification in a description and parameter field for the STRS_InstantiateApp API and an additional section in the STRS Handbook.

The description field for STRS_InstantiateApp() stated "Instantiate an application, service, or device and perform any operations imposed by the configuration file. The configuration file specifies such items as initialization values and state. The infrastructure is responsible for calling the appropriate methods (e.g., STRS_Configure and/or APP_Configure) to configure the initial or default values. Other STRS methods may be called to perform additional functions, such as loading images or performing change of state as described in the application state diagram." The parameter field of the STRS_InstantiateApp() API states that the parameter "toWF – (in char*)" is the "storage area name or fully qualified file name of the deployed configuration file of the application (or device) that should be instantiated. The handleName corresponding to the application, service, or device specified in the configuration file is to be unique…"

Concerns with the definition and valid timeframe for the Handle Name and Handle ID have led to the following update to the Description and Parameter fields of the STRS_InstantiateApp. The Description field will be clarified to add "The configuration file specifies such items as the unique handle name, location of the executable(s), initialization values, and state. The handle name corresponding to the application, service, or device specified in the configuration file is to be unique." The parameter field "*toWF*" will require the storage area name or fully qualified file name of the deployed configuration of the application (or device) that should be instantiated.

A new section, preliminarily named "Uniqueness of Handle Names and IDs", will be added to the STRS Handbook. This section will address the time during which the Handle Name is valid, how the Handle ID is determined, when the Handle ID must be unique and the relationship between the Handle Name and Handle ID. Example use cases will also be included, with example code.

2.3. Timing

The use of the STRS timing APIs invoked many comments about standardizing a timing service for implementation on GPPs. This would enable portability of waveforms across platforms because a translation for the time stamps, etc. that use the timing service would no longer be required. Unless STRS eventually is expanded to include platform services with defined APIs, this change to the architecture standard will not be added to STRS. Instead, discussion about the use of the STRS time APIs will be updated in the STRS Handbook.

Since the OE does not need to be portable, the STRS application developer must document the application's use of the time API. The clocks/timers are used for determining when an event occurs and how long it takes, and/or coordinating internal and external events, including

timestamps for messages, navigation, a cognitive engine's event processing, or conveying networking application's bundle/frame information.

As computer speeds increase, more real-time functions for communication may be performed in the GPP. Some functions currently in FPGA may be transitioned to the GPP when the GPPs are fast enough and capable enough to handle the additional signal processing functionality. These GPP functions will need access to high-speed clocks/timers. It is recommended that one clock/timer match the required timestamp for STRS_Log so that an application, service, the OE, or even STRS_Log itself could obtain that time in a consistent way. It was suggested that the time for the timestamp be retrieved via STRS_GetTime using the handle ID corresponding to handle name "OEClockAppName" and kind given by property "OEClockKind". The use of this handle ID for STRS_SetTime could be restricted as necessary.

A mission could create an STRS infrastructure requirement concerning the format of the telemetry and error messages, including the timestamp, to be used by the mission. The mission could either specify a format or the documentation of a previously created format. In either case, the documentation must include information about the interpretation of such messages.

There were also comments about adding methods to convert times between mission time and UTC time, or other time conversion needs. Time conversion tools exist and formats have been standardized by the CCSDS [10]. New integer types are being considered for the STRS architecture for STRS_Seconds and STRS_Nanoseconds to allow the integer values to vary in length. Guidance will also be added in the description of the STRS time functions to calculate the number of bits required for suggest increasing for STRS_Seconds and STRS_Nanoseconds to prevent rollover for missions with a long operational lifetime.

2.4. Operating Environment Information

The ability for an application to obtain current information about the STRS infrastructure, such as the version, active applications, resources used and free, and faults, was suggested. To obtain information about the OE, all that is needed is a handle name and handle ID that could be used in STRS_Query. Although the handle ID might be used in STRS_RunTest to obtain more complicated information from the OE, that should not be part of the standard. To promote application portability, the OE handle name should be specified in a named constant with a requirement such as:

The STRS infrastructure shall contain the handle name STRS_OE_HANDLE_NAME whose corresponding handle ID may be used to query the OE for information about the OE.

For example, the handle ID to be retrieve information from the OE could be obtained by the following line of code: *STRS_HandleID oeID = STRS_HandleRequest(fromWF, STRS_OE_HANDLE_NAME).*

2.5. SDRs without GPMs

Many SDRs under development do not have integrated General Purpose Processors (GPPs). Instead, they rely on an external system, generally the flight computer, to provide the functions needed in a GPP to control, receive telemetry, and configure the Signal-Processing Module. Requirement (STRS-109) requires that an STRS-compliant SDR contains a General Purpose Module containing a GPP (Figure 2).

A few options exist to enable this approach with STRS with varying degrees of compliance:

1) Distribute the functionality of the GPM to the flight computer by implementing an STRS OE on the flight computer. This approach is compliant with the current STRS architecture.
2) Control the SPM without STRS APIs. This approach is not fully compliant with the STRS architecture, but offers benefit for third party use of the platform and limited reuse of the application.
3) Implement STRS configurations through data packet via waveform. The commands could be data via the waveform to the FPGA and the FPGA could parse the data to affect its operation without the flight computer ever being involved. Although this option is also not STRS compliant, it also offers partial benefit for platform reuse and application portability.

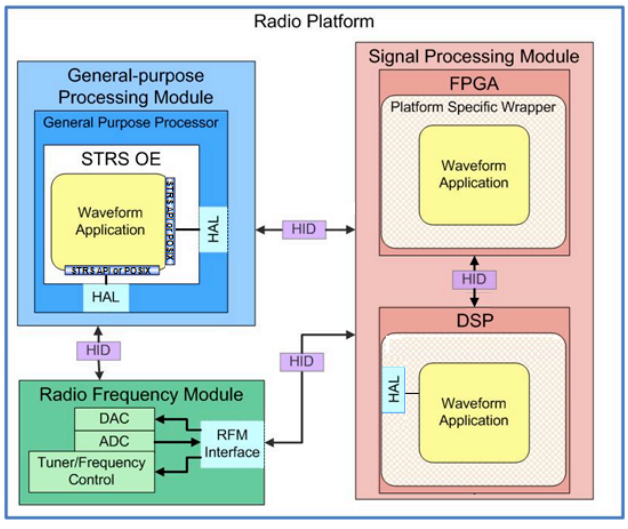

Figure 2. Current High-Level Software and Configurable Hardware Design Waveform Application Interfaces

4) Use a System on a Chip to implement the GPP functions. For this approach, the API methods would be implemented on the processing element on the chip and would therefore be STRS compliant.

Distributed functionality

STRS requirement #109 states "An STRS platform shall have a GPM that contains and executes the STRS OE and the control portions of the STRS applications and services software." Although the supporting figure (Figure 2) shows all components within a single radio platform, the radio functionality can be distributed. One method may be to implement the operating environment as part of the flight computer software as shown in Figure 3.

The issue with this approach might be the requirements placed on the flight computer software developer to add STRS APIs and behavior, along with the necessary additional test and documentation. It does reduce the number of processing elements and therefore would reduce the overall size, weight, and power of the combined flight computer and radio system.

Control the SPM using non-STRS methods

All specialized hardware has custom methods for configuration and control, such as specialized register settings. Without STRS APIs on a GPP, an SDR can continue to use this method. This obviously reduces the ability to port this application to a new, STRS compliant platform because these unique configuration items must be translated into STRS APIs. Other STRS requirements that enable third party development on the SDR, such as Hardware Interface Descriptions and test applications, will still offer advantages for future use of the platform.

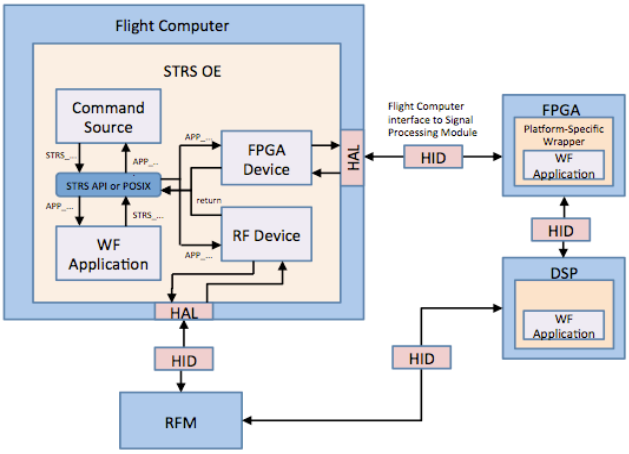

Figure 3. Distributed GPM Functionality

Control the SPM via the waveform

A waveform could have the ability to be a "command interpreter" for simple commands. By selectively parsing bits that are defined as commands from the incoming data stream received through the RF path and writing them to the preassigned register, the SPM could respond independently (without a GPP) to configuration and control functions. This has the same advantages and disadvantages for portability as method #2.

Implement STRS GPP functions on a SOC

This approach is STRS compliant, but for a highly constrained processing engine, the additional burden to implement STRS APIs may be more than the system can tolerate.

2.6. Cognitive Radio

Radios today are evolving from awareness toward cognition. An SDR provides the most capability for integrating autonomous decision making ability and allows the incremental evolution toward a cognitive radio. This cognitive radio technology will impact NASA space communications in areas such as spectrum utilization, interoperability, network operations, and radio resource management over a wide range of operating conditions.

NASA's cognitive radio will build upon the SDR infrastructure being developed by STRS. The STRS architecture defines methods that can inform the cognitive engine about the radio environment so that the cognitive engine can learn autonomously from experience and take appropriate actions to adapt the radio operating characteristics and optimize performance [11].

The STRS architecture contains methods to query to obtain information about the SDR (e.g. modulation scheme, power, signal to noise level, error rates, etc.), its environment, and its waveform applications as well as methods to control the operation of the SDR. A cognitive engine may use these features to optimize performance autonomously under adverse conditions such as mitigating the effects of unplanned interference and maximize the data throughput, reconfiguring due to propagation effects, or any effect that impacts communications. The cognitive data for a NASA SDR must include information about mission requirements and radio capability; because the SDR could be controlling satellite navigation or antenna pointing, for example.

Most processes described required for cognitive radios can be accomplished using the current STRS architecture with an "adapter" between the application layers and the cognitive engine as shown in Figure 4.

Following the concept, design, and proof of concept

implementation of a cognitive radio, it was determined that no changes are required to the STRS standard to implement CR. The CR should be a service invoked from an adapter STRS service. STRS services are software programs that provide functionality available for use by other applications. As a service, the CR can do a reboot, if necessary, or other functionality that is forbidden to an STRS application but may be necessary when the radio is more autonomous.

2.7. Acquisition Guidance

Many of the comments and questions are not addressed using a standard architecture, but are instead guidance that should be provided to a project when implementing an SDR platform and/or application. This guidance is being captured in the "STRS Project and Acquisition Guidance" document, to be released in 2016.

The focus of the first version of the guidance document is assuring that the data rights are considered when purchasing a platform or application. The platform software is not intended to be reusable across platforms, so the main focus in assuring the needed data rights are included in the purchase of an STRS compliant platform is requiring that the source code is available for STRS compliance tests. The distribution rights for the platform "wrapper" and the associated documentation must also be considered.

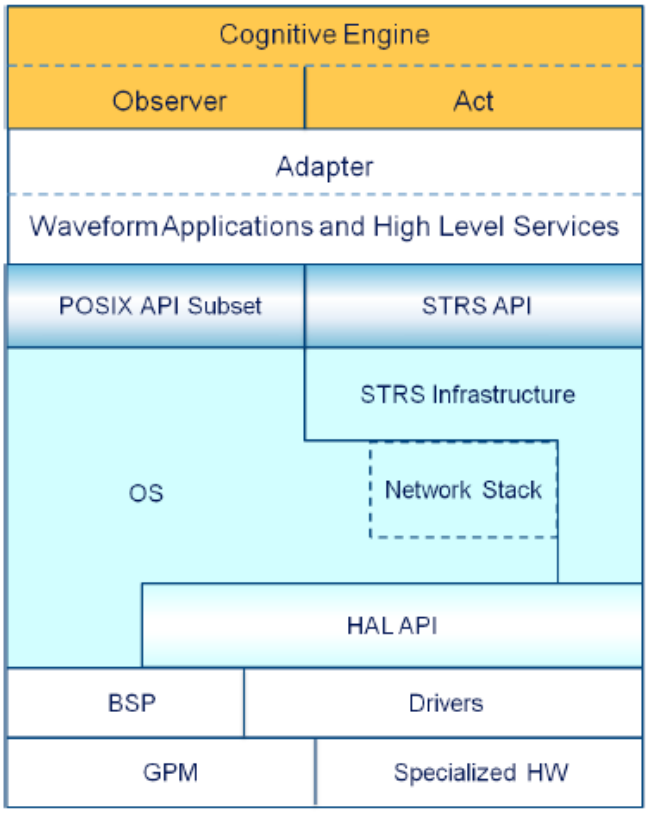

Figure 4. Cognitive Engine Layer Diagram

Data rights for applications must be considered in the initial procurement of the application in order to assure that it can be reused for future projects. It is also necessary that it can be redistributed by the STRS project. Redistribution by the STRS project provides an avenue to capture all of the relevant software and supporting documentation in a single place by a project with the responsibility to securely archive the application and provide it when requested. The STRS project has the responsibility to assure that the initial data rights are upheld when redistributing the application. NASA will not always fund the full application development and intellectual property will often be embedded with new developments. Therefore, it is critical to negotiate and understand the data rights prior to completing the contract. The acquisition guidance document contains suggestions for contract language to consider for multiple procurement vehicles, including SBIRs, Space Act Agreements, Cooperative Agreements, and Contracts.

Other challenges somewhat unique to SDR development that must be considered during the development of the Statement of Work include required documentation, STRS training and compliance, and consideration for spare resources for upgrades. New applications are likely to require additional external commands and telemetry. Considering this capability early in the design will reduce effort in the future. The SDR platform Verification and Validation (V&V) differs from that for a fixed platform. Test waveforms are suggested that access all interfaces and provide the necessary functions to test the range of the platform's capabilities without a full operational waveform development.

2.8 Standardized Platform Services

Platform services may be implemented as an STRS application or STRS service. There have been a few services that have been suggested for all radios, such as CCSDS service, security, commanding, telemetry, etc. However, none will be standardized in the next version of STRS but are likely candidates for future versions.

3. BACKWARDS COMPATIBILITY

The STRS version number will be captured for all applications submitted to the STRS application repository. Any changes to the application to integrate it with previous or future versions of an OE will be the responsibility of the integrator of the project reusing the application. Documentation will be developed detailing all changes to each version of STRS. Updates to STRS are expected to occur every two to three years, limiting the number of versions available. This would limit the amount of time

porting newer applications to preexisting STRS radios or porting existing applications to newer radios.

4. SUMMARY

In summary, the following significant suggestions for the STRS architecture were considered and the proposed resolution is as follows:

1) STRS Devices: No changes will be made to the current architecture, but additional clarification will be added to the supporting handbook.
2) Handle Names and IDs: Updates will be made to the parameter and description fields in the appropriate function calls to include the handle name and uniqueness aspects. A new section will be added to the supporting handbook to discuss timeliness.
3) Timing APIs: A new integer type will be added to allow for varying lengths for the applicable time values.
4) Operating Environment Information: A new requirement will be added to the architecture to require a standard naming convention for the application to use to obtain operating environment details.
5) SDRs without GPMs: Four designs for SDRs without conventional General Purpose Processors as a separate module in an SDR were proposed and the pros and cons for each suggestion were discussed.
6) Acquisition Guidance: A summary of the unique aspects to be considered when developing an SDR versus a traditional fixed radio was provided. These will be captured in detail in a future document.
7) Standardized Platform Services: Platform Services that are common to most NASA SDRs may be added to the STRS architecture in future versions.

5. FUTURE WORK

The updates mentioned in this paper will continue to be evaluated in their significance for application portability. The disposition recommended by the STRS team will be discussed with the commenter, if applicable, and reviewed by a broader team of personnel with STRS knowledge. If applicable, the suggested software changes will be coded to assure accuracy. NASA's reference implementation will be changed accordingly so as to have an executable model to test.

6. REFERENCES

[1] http://www.public.navy.mil/jtnc/sca/Documents/SCAv2_2_2/SCA_version_2_2_2.pdf

[2] "*Enhancing the Platform Independent Model (PIM) and Platform Specific Model (PSM) for Software Radio Components Specification Version 1.0—a.k.a. SWRadio specification*", OMG document number dtc/06-04-17, et al., Object Management Group (OMG), 2006

[3] T. Quinn and T. Kacpura, "*Strategic adaptation of SCA for STRS*", 2006 Software Defined Radio Technical Conf. Product Exposition, Nov. 2006.

[4] J. C. Briones, L. M. Handler, C. S. Hall, R. C. Reinhart, and T. J. Kacpura, "Case study: Using the OMG SWRadio profile and SDR Forum input for NASA's Space Telecommunications Radio System," in *SDR'08 Technical Conf. Product Exposition*, Oct. 2008.

[5] Richard C. Reinhart, Thomas J. Kacpura, Sandra K. Johnson, James P. Lux, "*NASA's Space Communications and Navigation Test Bed aboard ISS to Investigate Software Defined Radio, On-board Networking and Navigation Technologies*", IEEE Aerospace and Systems Magazine, Volume 28, Number 4, April 2013.

[6] Space Telecommunications Radio Systems (STRS) Architecture Standard, Version 1.02.1, NASA/TM-2010-216809.

[7] "*Space* Telecommunications *Radio System (STRS) Architecture Standard, NASA-STD-4009*", NASA Technical Standard, June 2014.

[8] "*Space Telecommunications Radio Systems (STRS) Architecture Standard Rationale, NASA-HDBK-4009*", NASA Technical Handbook, June 2014.

[9] http://www.public.navy.mil/jtnc/sca/Documents/SCA_APIs/API_3.0_20131002_Mhal_withErrata.pdf

[10] "*Time Code Formats*, *CCSDS 301.0-B-4"* Consultative Committee for Space Data Systems (CCSDS), November 2010.

[11] L.M. Handler, J.C. Briones, "*Space Telecommunications Radio Systems (STRS) Cognitive Radio*", Wireless Innovation Forum, 2013.