UPDATES TO THE NASA SPACE TELECOMMUNICATIONS RADIO SYSTEM (STRS) ARCHITECTURE

Thomas J. Kacpura (ASRC Aerospace, Cleveland, OH, USA, Thomas.J.Kacpura@nasa.gov); Louis M. Handler (NASA Glenn Research Center, Cleveland, OH, USA, Louis.M.Handler@nasa.gov); Janette C. Briones (NASA Glenn Research Center, Cleveland, OH, USA, Janette.C.Briones@nasa.gov); Charles S. Hall (Analex Corporation, Cleveland, OH, USA, Charles.S.Hall@nasa.gov)

ABSTRACT

This paper describes an update of the Space Telecommunications Radio System (STRS) open architecture for NASA space based radios. The STRS architecture has been defined as a framework for the design, development, operation and upgrade of space based software defined radios, where processing resources are constrained. The architecture has been updated based upon reviews by NASA missions, radio providers, and component vendors. The STRS Standard prescribes the architectural relationship between the software elements used in software execution and defines the Application Programmer Interface (API) between the operating environment and the waveform application. Modeling tools have been adopted to present the architecture. The paper will present a description of the updated API, configuration files, and constraints. Minimum compliance is discussed for early implementations. The paper then closes with a summary of the changes made and discussion of the relevant alignment with the Object Management Group (OMG) SWRadio specification, and enhancements to the specialized signal processing abstraction.

1. INTRODUCTION

Since the original release of the STRS architecture,[1,2] NASA has received comments. A consistent theme has been to increase the detail of the architecture. NASA has recently released STRS Architecture Standard Version 1.01,[3] which is an update and has improved the details of the software architecture. A key focus of the updates has been refining the STRS infrastructure and the specific STRS API.

The STRS Infrastructure is part of the General Purpose Processor (GPP) Operating Environment (OE) and provides the functionality for the interfaces defined by the STRS API specification. Once the waveform is deployed, the infrastructure supports the waveform operations through the STRS API and its internal subsystems. The infrastructure is composed of multiple subsystems that interoperate to provide the functionality to operate the radio. The components shown in Figure 1-1 represent the high level subsystems and services needed to control waveforms and applications within the radio platform. These services are provided by the platform infrastructure and support applications as they execute within the radio platform.

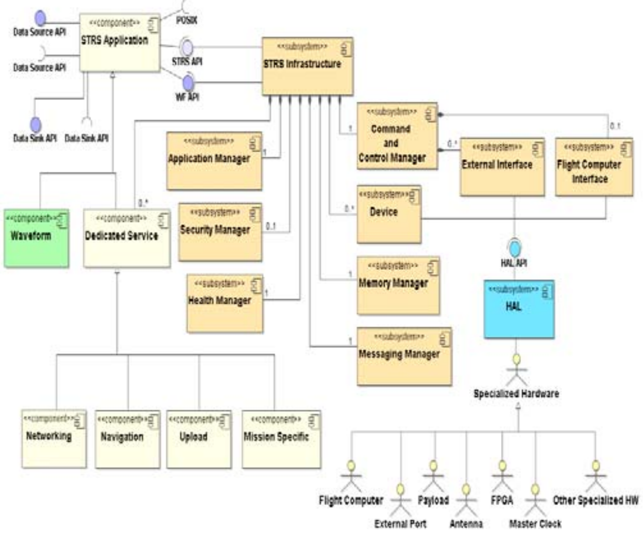

Figure 1-1 STRS Infrastructure

The infrastructure implements the STRS API. The STRS API is the 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. The STRS API provides the interfaces that allow applications to be instantiated and use platform services. This API also enables communication between waveform and application components. The STRS API includes support of external interface commands for normal radio operations. It hides the routine names actually used by the STRS infrastructure from the waveforms to facilitate portability. Although the STRS infrastructure may use any combination of Portable Operating System Interface (POSIX), real time operating system (RTOS), board support package (BSP) functions, or other infrastructure methods to support radio functions, which may vary on different platforms, the STRS API will be identical to allow portability.

2. STRS API

The STRS API provides an open software specification for the application engineer to develop STRS waveform application programs. The goal is to have a standard API available to cover all application program requirements so that the waveform programs can be reused on other hardware systems with minimal porting effort and cost of the waveform software (and firmware) development. Two trade-offs in the development of the API specification are a) the larger the API specification then the greater the software overhead, which affects size, weight, and power (SWaP) and b) standardization of the API which limits the ability to use custom routines for optimization. The STRS API definition minimizes dependencies on specific capabilities of the GPPs.

The API layer specification decouples the intellectual property rights of platform, waveform, and module developers. The API layer allows development and interoperability of different radio aspects while protecting the investment of the developers.

2.1. STRS Application Control API

A key aspect of a software-architecture is the definition of the API that is used to facilitate software configuration and control of the target platform. The philosophy, on which the STRS architecture is based, avoids the conflict between open architecture and proprietary implementations by specifying a minimum API used to execute waveform applications and deliver data and control messages to installed hardware components.

Figure 2-1 is a class diagram in Unified Modeling Language (UML) that illustrates the inheritance between the classes and the corresponding implementation objects in C++. In a C or C++ implementation, it depicts the hierarchy of include files. The figure also shows a grouping of API. A waveform or service is a STRS Application implementation object that must implement the STRS Application Control API. The STRS Application Control API is comprised of the STRS ComponentIdentifier, STRS ControllableComponent, STRS LifeCycle, STRS PropertySet, and STRS TestableObject API groups.

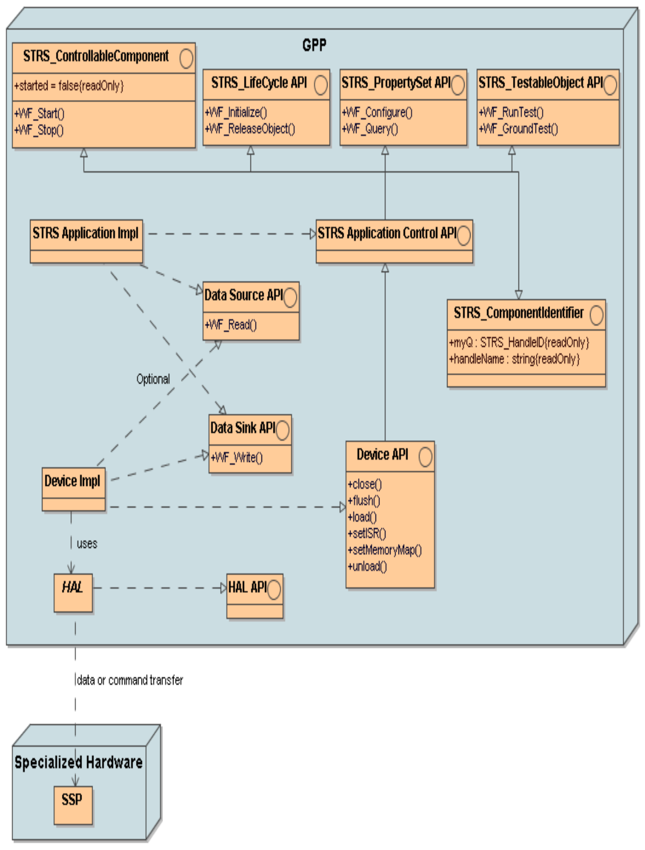

Figure 2-1 STRS Waveform/Device Structure

STRS requires the methods shown in the table below to be implemented by each waveform or service. The STRS Application Control API shown below exhibits similar functionality to a Resource Interface in the OMG SWRADIO or SCA specifications except that the notion of ports has been replaced with the optional source or sink. The API may be implemented using the same OMG SWRadio Platform-Independent Model (PIM).

STRS Application Control API	
WF_Configure	Set values for one or more properties in the waveform.
WF_GroundTest	Perform unit and system testing usually done on ground before deployment. The testing may include calibration. The method is similar to WF_RunTest except that it contains more extensive testing that can be eliminated for actual flight.
WF_Initialize	Initialize the waveform to a known initial state. Used to restart from the beginning rather than from where it left off.

STRS Application Control API	
WF_Query	Obtain values for one or more properties in the waveform.
WF_Read	Method used to obtain data from the waveform. Optional.
WF_ReleaseObject	Free any resources the waveform has acquired. An example would be to close open files or devices.
WF_RunTest	Test the waveform. The tests provide aid in isolating faults within the waveform.
WF_Start	Begin normal waveform processing.
WF_Stop	End normal waveform processing.
WF_Write	Method used to send data to the waveform. Optional.

2.2. STRS Infrastructure Application Control API

The Infrastructure Application Control methods correspond to the STRS Application Control API exactly and are used to access those methods. These methods are implemented by the STRS infrastructure but may be used by any STRS Application or any part of the infrastructure that is desired to be implemented in a portable way. A handle ID is an identifier that is used to control access to applications and resources such as another waveform, device, file, or message queue.

STRS Infrastructure Application Control API	
STRS_Configure	Set values for one or more properties in the waveform (or device).
STRS_GroundTest	Perform unit and system testing, including calibration, usually done on ground pre-deployment.
STRS_Initialize	Initialize the waveform. Used to restart from the beginning rather than from where it left off.
STRS_Query	Obtain values for one or more properties in the waveform (or device).
STRS_Read	Method used to obtain data from a source or supplier.
STRS_ReleaseObject	Free any resources the waveform has acquired. An example would be to close open files or devices.
STRS_RunTest	Perform built in test.
STRS_Start	Begin normal waveform processing.
STRS_Stop	End normal waveform processing.
STRS_Write	Method used to send data to a sink.

2.3. STRS Infrastructure Application Setup API

The Infrastructure Application Control Setup methods are used in general or to control one waveform from another. A handle ID is an identifier that is used to control access to applications and resources such as another waveform, device, file, or message queue.

STRS Infrastructure Application Control Setup API	
STRS_AbortApp	Abort a waveform or service
STRS_GetErrorQueue	Transform an error code into an error queue.
STRS_GetSizeOfProperties	Compute number of bytes in a STRS Properties struct containing a given maximum number of STRS Property name/value structs. The number returned is used to allocate space for the STRS Properties struct.
STRS_HandleRequest	The table of object names is searched for the given name and the handle ID is returned that is used to control access to another waveform, device, file, or message queue.
STRS_InitComplete	Return initialization completion status when the task is initiated independent of the completion.
STRS_InstantiateApp	Instantiate a waveform or service (or device).
STRS_IsOK	Return true, if return value of previous call is not an error code.
STRS_Log	Send log message for distribution as appropriate. Time stamp is added automatically.
STRS_RemoveApp	Remove specified waveform or service from persistent storage.
STRS_UploadComplete	Return upload completion status.
STRS_UploadRequest	Begin or continue upload.

2.4. STRS Infrastructure Device Control API

STRS Devices are controlled using the STRS Infrastructure Device Control API shown in this section. A STRS Device is a proxy for the data and/or control path to the actual hardware. A STRS Device may use any available platform-specific Hardware Abstraction Layer (HAL) to communicate with and control the specialized hardware. A STRS Device may also be used to hide the details of networking from the waveform. The purpose of abstracting the hardware interfaces in a standard manner is to make the waveforms more portable. A STRS Device is a STRS application that responds to the STRS Infrastructure Application Control API calls as well as to the following additional calls.

STRS Infrastructure Device Control API	
STRS_DeviceClose	Close the device.
STRS_DeviceFlush	Send any buffered data immediately to the underlying hardware and clear the buffers.
STRS_DeviceLoad	Load a binary image to the device.
STRS_DeviceOpen	Open the device.
STRS_DeviceReset	Reinitialize the device. Reset is normally used after the device has been started and stopped, before starting the device again.
STRS_DeviceStart	Start the device.
STRS_DeviceStop	Stop the device.
STRS_DeviceUnload	Unload the device.
STRS_SetISR	Set the Interrupt Service Routine for the device.

2.5. STRS Infrastructure Memory API

These Infrastructure Memory methods are used to isolate the memory manipulation on small and large platforms so that the memory is used in a portable way. On a small platform, the total available memory may be severely limited. On a large platform, the total available memory may be limited only by the size of a disk swap area. The same methods are used in both situations for portability.

STRS Infrastructure Memory API	
STRS_Clone	Acquire a section of memory to use, copy data into it, and return the new memory location.
STRS_Release	Release a section of memory previously acquired with STRS_Clone or STRS_Reserve.
STRS_Reserve	Acquire a section of memory to use and return the new memory location.

2.6. STRS Infrastructure Messaging API

The messaging methods allow STRS applications to use a single target handle ID to send messages between applications or to multiple parts of the radio. The ability for waveforms to communicate with other STRS applications is crucial for the operation of radio services as well as separating the receive and transmit functionality between two waveforms. The messaging API is implemented using a form of the Observer or Publish-Subscribe design pattern.

STRS Infrastructure Messaging API	
STRS_QueueCreate	Create a queue.
STRS_QueueDelete	Delete a queue.
STRS_Register	Register an association between a publisher and subscriber.
STRS_UnRegister	Remove an association between a publisher and subscriber.

2.7. STRS Infrastructure Time Control API

These Infrastructure Time Control methods are used to access the hardware and software timers.

STRS Infrastructure Time Control API	
STRS_GetNanoseconds	Get the number of nanoseconds from the STRS_TimeWarp object.
STRS_GetSeconds	Get the number of seconds from the STRS_TimeWarp object.
STRS_GetTime	Get the current base time and the corresponding time of a specified type.
STRS_GetTimeWarp	Get the STRS_TimeWarp object containing the number of seconds and nanoseconds in the time interval.
STRS_SetTime	Set the current time in the specified clock/timer by adjusting the time offset.

STRS_Synch	Synchronize clocks. The action depends on whether the clocks to be synchronized are internal or external.

3. CONFIGURATION FILES

STRS configuration files shall contain platform and waveform specific information for the installation and customization of waveforms. Platform configuration files provide the STRS infrastructure with information on what hardware devices and modules are installed in the system. The configuration files are used by the STRS Infrastructure to determine what files, devices, waveforms, and services are used by the STRS radio. The name of the starting configuration file is specified on the command line when initializing the STRS Infrastructure. If none is specified, a mission specific default would be employed. A waveform (STRS application) configuration file contains specific information that 1) allows STRS to instantiate the application; 2) provides default configuration values; and 3) provides connection references to devices, queues, and services needed by the application.

The format of the configuration files shall be defined in Extensible Markup Language (XML) using an XML Schema. The XML Schema Definition Language is an XML language for describing and constraining the content of XML documents. The XML can be preprocessed to optimize space on the STRS Radio memory while keeping the equivalent content.

One approach to accomplish the preprocessing, used in the STRS Reference Implementation, is to use an XSL transformation. Here the XSLT language, which itself uses XPath, was used to specify how to transform the given XML input into the desired output. One suggestion for a more compact representation is S-Expressions, which could be used if a more compact representation is desired.

3.1. Platform Configuration Files

The contents of a platform configuration file include a list of hardware modules having memory able to contain data and executable software. There is a unique module name for each hardware module accessible from the current GPP. The platform configuration file includes a list of memory areas of various types (e.g. ROM, RAM), sizes, units, and access. The platform configuration file includes a memory map list which provides the base name, base address, memory size, and memory read and write access. It also contains a module type which is the name of the hardware type. The module type may be the GPP, RF, FPGA, DSP, ASIC, etc.

3.2. STRS Infrastructure Configuration Files

The STRS Infrastructure configuration data is one example of the data that defines the infrastructure. The infrastructure configuration file includes a list of files to read, write, or append, from multiple locations using a handle ID. The file data includes a handle name, file name, file type and file access. The infrastructure configuration file includes a list of devices to read or write from multiple locations using a handle ID. The device data has a handle name, device name, device type, device access, and attribute list. The infrastructure configuration file includes a list of attributes that are tested against specific values to indicate the health of the system. The infrastructure configuration file includes a queue list containing the correspondences between publishers and subscribers.

3.3. STRS Waveform Configuration Files

A waveform (STRS application) configuration file contains specific information that 1) allows STRS to instantiate the application; 2) provides default configuration values; 3) provides connection references to devices, queues, and services needed by the application.

An example of a Waveform Configuration File in XML is shown in Figure 3-1:

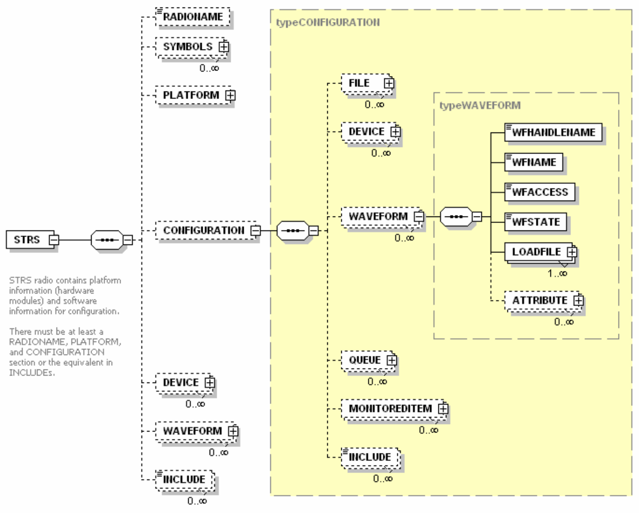

Figure 3-1 Example STRS Waveform Configuration File in XML

The contents of a STRS Waveform configuration file include a handle name, that is a unique shortened form of the waveform name used in messages and a waveform name, (usually a shortened form of the waveform that will be the C++ class name). Access to the waveform may be specified as read, write, both, or none. Read indicates that the waveform implements WF_Read(). Write indicates that the waveform implements WF_Write(). The initial state is the state at which the waveform is left after processing the configuration file. The state may be instantiated or running. A file list contains a list of files to be loaded for execution and includes the file name and the target module name. An attribute list contains the list of properties having a name and value pair set as the default during initialization.

4. STRS MINIMUM COMPLIANCE

A minimum compliance has been defined for systems installed on constrained space platforms and that supports upwards compatibility on larger platforms. It is expected that this minimum compliance will be satisfactory on early STRS platforms, enabling the experience and lessons learned to feedback into further architecture definition. The minimum compliance builds upon the previously defined APIs and configuration files and adds the following additional elements discussed below.

Minimum compliance requires publishing the Hardware Interface Definition (HID) and HAL, employing configuration files defined in XML (described by a XML schema), the use of selected POSIX subsets, and using the minimum list of the STRS API. The HID has been compared to an Interface Control Definition, with the requirement to publish interfaces and the operating requirements of the hardware system after delivery. The HAL in the GPP is software that configures, controls, and communicates with specialized hardware by abstracting the physical hardware interfaces. The HAL API shall be published so that specialized hardware made by one company may be integrated with the STRS Infrastructure made by a different company. Platform and Waveform Configuration Files require the use of XML to describe the contents; however an approach for the expected transformation to a more compact form to meet space memory requirements is suggested but not mandated as part of the architecture.

The STRS API is split into the STRS Application Control API and the STRS Infrastructure API. A waveform is a STRS Application and waveform developers must implement the STRS Application Control API listed above and defined in the STRS Architecture Standard. The STRS Infrastructure is part of the OE and provides the functionality for the interfaces defined by the STRS API specification. The STRS infrastructure must implement the STRS API listed to support applications as they execute within the radio platform. Additional functionality must be implemented in the STRS Infrastructure for radio robustness and mission dependent requirements. In addition, radio developers must provide the HID and HAL documentation.

The STRS architecture requires that compliant radios must use a POSIX conformant RTOS, or provide a POSIX abstraction layer (minimum POSIX real time profile PSE51) to provide the POSIX API missing from RTOS. For constrained resource platforms, with limited software evolutionary capability, where the waveform signal processing is implemented in specialized hardware, the suppler may request permission from NASA to only implement a subset of POSIX PSE51 as required by the portion of the waveforms residing on the GPP. The waveforms created for this platform must be upward compatible to a larger platform containing POSIX PSE51. If none of the waveforms for a constrained resource platform use any of the interfaces in a unit of functionality, then the supplier may request permission from NASA to eliminate that entire unit of functionality.

The difference between a POSIX conformant RTOS and a non-conformant RTOS is illustrated in Figure 4-1. On the left side, the POSIX AEP is provided entirely by the RTOS. The POSIX API is included for the RTOS. On the right side, if the RTOS is not POSIX AEP conformant then a POSIX Abstraction Layer must be provided to implement the required missing functionality.

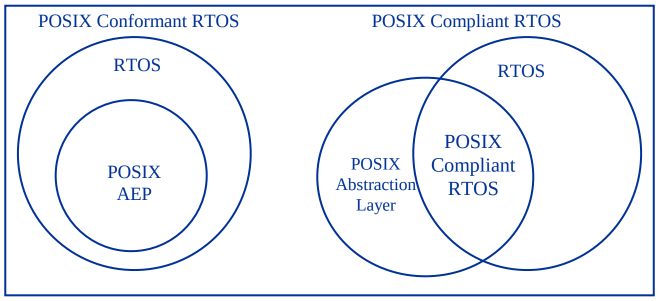

Figure 4-1 POSIX Compliant versus Conformant

5. CONCLUSION/ACKNOWLEDGEMENTS

The STRS architecture has been updated, focusing on key elements of the software architecture. Reference implementations and early STRS compliant radios are being developed, and minimum compliance criteria are described. Future planned updates include adding more detail to specialized signal processing abstraction, the hardware architecture, and providing a waveform developer's guide.

The authors would like to acknowledge the support of the NASA STRS project team and the SDR Forum Space Working Group (SWG). A key recommendation from the SWG has been implemented, where possible, to align with the OMG SWRadio specification, improving API definition.

6. REFERENCES

[1] "Space Telecommunications Radio System Open Architecture Description," December 2005.
[2] Thomas J. Kacpura, and Richard C. Reinhart, "STRS Architecture Standard", Revision 1.0, April 2006.
[3] Thomas J. Kacpura, and Richard C. Reinhart, "STRS Architecture Standard", Revision 1.01, June 2007.