SOFTWARE DEFINED RADIO SERVICES AND DEVICE APIS

Eric Christensen, Ph.D. (General Dynamics C4 Systems, Scottsdale, AZ 85257
eric.christensen@gdds.com) David Dohse, (General Dynamics C4 Systems,
Scottsdale, AZ dave.dohse@gdds.com)

ABSTRACT

The Software Communications Architecture (SCA) Version 2.2 and the Application Programming Interface (API) supplement have been in existence since November 2001. The objective of the SCA is to foster an open architecture in which waveforms/applications are portable across a wide range of SDR implementations. There have been many incarnations of the Core Framework specified by SCA V2.2 however, to this point in time there have not been any APIs for Radio Services or Radio Devices which are unencumbered by intellectual property rights published for the community. The lack of publicly available APIs for Radio Services and Devices is inhibiting the progress of Software Defined Radio technology both from a hardware platform and SDR application vendor perspective. This lack of publicly available unencumbered APIs leads to proprietary single point implementations of waveforms and Radio Services and Devices. There are several factors that may be inhibiting the development of an open architecture SDR: 1) No commonly accepted definition of a set Radio Services and Devices which are part of a SDR platform; 2) No Naming conventions; 3) No commonly accepted content and format for an API. Current API standardization efforts within the SDR Forum and the OMG [1] have shown promise yet do not establish interfaces down to the level of method invocation signatures necessary for portability.

This paper advocates the premise that the SDR should provide a set of commonly used Radio Services and Devices to waveforms and other SDR applications. In essence, this means the SDR existence is independent of any particular waveform or application, but provides a general set of Radio Services and Devices that are usable by the many waveforms and applications. For example, a CVSD vocoder could be provided as a Radio Service or Device, which could be then used by several different waveforms to include SINCGARS, HaveQuick I/II, and SATCOM 181, and 183. This paper provides a definition of Radio Services and Devices, a classification of SDRs by capability and then proposes a set of Radio Services and Devices that should be present in a class of SDRs. A naming convention for Radio Services and Devices is proposed. The paper then proposes a specification for the content and format of Radio Service and Device APIs using a Serial IO device as an example.

1. INTRODUCTION

The Software Communications Architecture (SCA) Version 2.2 (updated to Version 2.2.1, April 30, 2004 and updated to Version 3.0 August 27, 2004) and the associated Application Programming Interface (API) Supplement [2] have been published since November 2001. In the past 2½ years there has been much attention directed towards the intricacies of the Operating Environment (OE) and Domain Profile but minimal if any focus on the API supplement or defining and specifying the services or devices that are sufficient and necessary to have a software defined radio (SDR) and waveform portability. The focus on the OE and Domain Profile have been primarily driven from an abstract perspective without consideration of the ultimate target environment of creating a SDR. This is evident from the multi-megabyte footprints of the OEs. There are multiple vendors claiming to have SCA V2.2 compliant OEs but to this date even though a multi-hundreds of million dollar development contract for SCA compliant Hardware and Waveforms (JTRS Cluster 1) was awarded in June 2002, there has not been any publication of unencumbered APIs for the Radio Services and Devices. The lack of published unencumbered APIs is inhibiting the progress of SDR technology and development. Without publicly available unencumbered APIs, innovative waveform and application developers cannot participate. This situation is akin to development of a proprietary closed architecture computer vs. an open architecture computer. In this case the development of SDRs and waveforms by JTRS Cluster 1 contractors is beginning to mirror the development of a proprietary closed architecture rather than the envisioned open architecture paradigm in which the APIs are published unencumbered resulting in a multitude of companies writing software applications and building JTRS compatibles.

2. RADIO SET VS. WAVEFORM VIEW

A major contributor to the lack of unencumbered published APIs for Radio Services and Devices is the fact there is not an accepted definition of what a Radio Service or Device is or what Radio Services and Devices are required for a SDR. Another factor clouding the API landscape is the perspective from which APIs should be defined. There are

two opposing views. One view is that APIs are defined in terms of waveforms such that each Waveform has its own set of APIs. Another view is that APIs are defined from the perspective of the Radio Set and then waveforms are developed to the Radio Set APIs. For the purpose of this paper these views are the Waveform API and the JTR Set API respectively. The Radio Set API view is based upon the premise that the Radio Set exists independent of any particular waveform and as such provides a common set of Radio Services and Devices that may be used by any waveform. The Waveform API approach implies that there is no commonality between waveforms and that the services or devices required for each waveform are unique.

The Waveform API approach is attempted in the SCA API Supplement but is abandoned with the statement "The range and variety of services at the various interfaces, most notably the MAC and Physical, make a common API for all waveform applications large and burdensome for resource constrained implementations." The SCA API supplement also attempted to partition waveforms using the OSI stack, which does not map directly to the Radio Set services and devices. Ultimately the SCA API supplement essentially abandoned the effort to define APIs by defining abstractions called "Building Blocks" (BB). These BB are intended to be templates, which are to be used by developers as a basis for forming APIs. A collection of the instantiated BB for a particular layer of the waveform defines the API for that waveform layer. The BB do not provide a sufficient level of abstraction to support portability since they are more physically than logically oriented.

When defined from the Radio Set perspective APIs are defined for the Radio Services and Devices which are then used by the waveforms. To put the Radio Set definition of APIs and Services into perspective and context of the SCA it is useful to use an architectural diagram of the Software Defined Radio Domain as shown in Figure 1[3]. Figure 1 shows there are 4 architectural components in the SDR Domain. The Computational Architecture component is the equivalent of the SCA Operating Environment. The Management Architecture contains the Domain Manager (Configuration Management) of the SCA and adds other necessary management services such as system control, fault management, performance management, virtual channel management, security management, and network management. The Services component of the architecture contains the radio domain services and devices. The Waveform/Application component of the architecture interacts through the Services architecture with the other architectural components. Another way of expressing the SDR Domain architecture is to view the Computational, Management, and Service components as a hardware abstraction layer upon which waveforms/applications are executed. From either a waveform view or a Radio Set view of APIs, each of the architecture components in the SDR Domain are required.

If the Waveform view is taken then each of the required services is defined in terms of a particular waveform. If another waveform requires that very same service (functionality) then that waveform will define the service in its terms. Thus for example SINCGARS and HaveQuick both require a CVSD vocoder. Each will implement the same algorithm (possibly from the same source) however each may choose to have different names for attributes and operations. In fact if the IDL structures are different the implementations will be guaranteed to have different names for the operations.

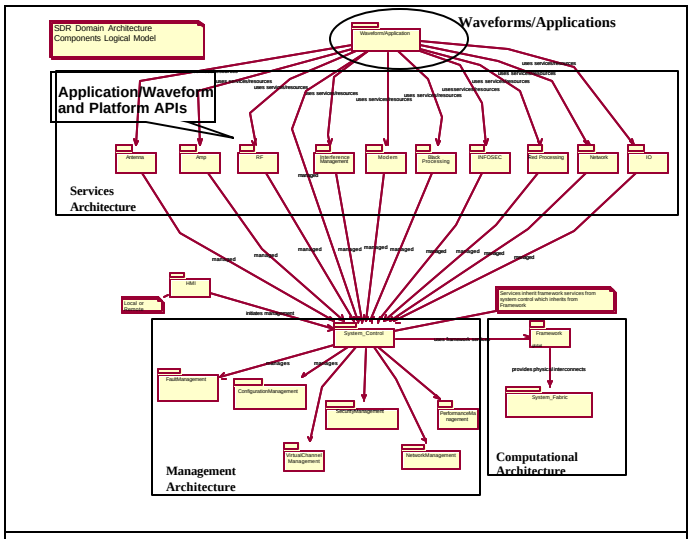

Figure 1 SDR Domain Architecture Model

If the Radio Set view is taken, a baseline for a service is defined based upon the requirements of a set of available waveforms. For example the JTRS ORD specifies the waveforms required for a particular domain. Using those waveforms an analysis can be performed to determine which vocoding algorithm is used by which waveforms. Further analysis can also be performed as to whether bridging may be required between the waveforms. From this analysis a determination could be made as to whether to implement a general Vocoding Service or to implement specific services such as CVSD, LPC-10, MELP, IMBE, and etc. Even if specific Vocoding services are implemented, it is advisable to implement them as commonly as possible to facilitate the implementation of transcoding which will be required, for example, to bridge a waveform that uses CVSD to a waveform that uses LPC-10.

The set of APIs for the Radio Services and Devices may be defined more narrowly and then extended to support additional waveforms. A common argument that derails serious attempts to define APIs from the JTR Set perspective

is "What are Radio Services and Devices and which of the Services and Devices are required?" For the purposes of this paper, a Radio Service is implemented in software, waveform independent, hardware independent, and inherits from the SCA CF Resource class. A device is hardware dependent, waveform independent, and provides a software interface inherited from the SCA CF Device Class. Examples of typical Radio Services are System Control, Vocoding, Waveform Monitoring, HMI interface; User Interface Control… Examples of typical radio devices are Serial I/O; Analog I/O; Ethernet I/O; DAC; Digital Signal Processor (DSP); General Purpose Processor (GPP), Field Programmable Gate Array (FPGA) Transmitter, Receiver; Transceiver, and Security Module.

3. CLASSIFICATION OF SDRS

Every SDR does not require the same set of services or devices. For example a single channel SDR rarely has need for a cosite interference mitigation service. Thus, the number of channels is one consideration in determining the required services. Other considerations may be size, weight, power, cost (SWAP-C), and types of waveforms to be hosted. The ultimate goal in defining and providing services is to minimize the effort to port a waveform and to maximize software reuse. Figure 2 illustrates the multi-dimensionality of SDR requirements.

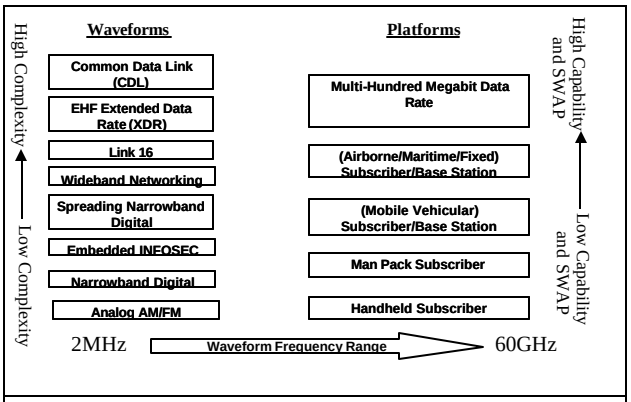

Figure 2 Many Considerations Impact What Requirements must be Anticipated in a SDR

In Figure 2, platform classes are defined as Handheld, Man Pack, Mobile/Vehicular, Airborne/Maritime/Fixed and Multi-Hundred Megabit Data. These classes are primarily defined in terms of number of channels, size, weight, and power consumption. The notation of subscriber and basestation also implies in some cases a complexity factor for a particular waveform. For example the complexity of an APCO-25 basestation is significantly greater than that of an APCO-25 subscriber.

Waveform complexity as shown in Figure 2 ranges from very simple waveforms such as AM/FM to more complex waveforms such as Common Data Link. The waveform complexity may range from the digital signal processing required, to the demands placed upon the Transmitter or Receiver for tuning time and speed. If SWAP-C were not considerations then there would be no need to classify SDRs, since a single SDR class could execute all waveforms. However since SWAP-C are significant elements in determining the ability of an SDR to meet user requirements it is imperative that SDRs have a classification system that allows the user to understand its current and future capabilities as well as allowing SDR vendors to target appropriate market segments. For example, a vendor desiring to enter the commercial single channel handheld market is not going to need cosite interference management whereas a vendor entering the Fixed basestation market could use their cosite interference management service as a major market distinguisher. For the purposes of this paper SDRs, are classified in terms of the waveforms they are intended to host, Security type, and number of channels in a package. The following classes are defined:

Class I: Single Channel, Handheld/ManPack, nonType I, narrowband data and voice
Class II: Single Channel, Handheld/ManPack. Type I, narrowband data and voice
Class III: Single Channel, Handheld/ManPack, Type I, wideband data and voice
Class IV: 2 Channel, Handheld/ManPack, Type I, wideband data and voice
Class V: 2 or more Channel Vehicular/ Airborne/ Maritime/Fixed, nonType I, wideband data and voice
Class VI: 2 or more Channel Vehicular/Airborne/ Maritime/Fixed, Type I, wideband data and voice
Class VII: 1 or more Channel Multi-Hundred Megabit Data, Type 1

Table I is a listing of potential services for a SDR and a mapping to SDR classes which must implement the service. An X indicates the class is required to implement the service. Items in **bold print** indicate proposed groupings of devices/services used to organize the services by function.

Table I: Devices/Services required by SDR Class

Devices/Services / SDR Class	I	II	III	IV	V	VI	VII
External RF							
Antenna						X	X
Antenna Coupler					X	X	
Amplifier					X	X	X
RF Switching					X	X	X

Table I: Devices/Services required by SDR Class							
Devices/Services / SDR Class	I	II	III	IV	V	VI	VII
External RF Control IO					X	X	X
Internal RF							
Transceiver	X	X	X	X	X	X	X
Cosite Mitigation					X	X	X
Digital to Analog	X	X	X	X	X	X	X
Analog to Digital	X	X	X	X	X	X	X
Signal Processing							
Modem	X	X	X	X	X	X	X
Black GPP Processing	X	X	X	X	X	X	X
Type 1 Security							
Key Management		X	X	X		X	X
Crypto Services		X	X	X		X	X
Red GPP Processing	X		X	X		X	X
NonType 1 Security							
Key Management	X				X		
Crypto Services	X				X		
Networking			X	X	X	X	X
Audio IO							
Codec	X	X	X	X	X	X	X
Vocoder	X	X	X	X	X	X	X
Transcoding					X	X	X
Speaker	X	X	X	X	X	X	X
Microphone	X	X	X	X	X	X	X
Push-to-Talk	X	X	X	X	X	X	X
Digital Data IO							
Serial IO	X	X	X	X	X	X	X
Ethernet IO				X	X	X	X
1553 IO					X	X	X
System Management							
System Control	X	X	X	X	X	X	X
Preset Management	X	X	X	X	X	X	X
Software Download	X	X	X	X	X	X	X
Timer	X	X	X	X	X	X	X
Fault Management	X	X	X	X	X	X	X
Configuration Management	X	X	X	X	X	X	X
Virtual Channel Management	X	X	X	X	X	X	X
Performance Management	X	X	X	X	X	X	X
Network Management				X	X	X	X

At this high level of the Service/Device decomposition it is apparent that not all classes of SDRs require the same services. Even in the services/devices that appear to be common the same functionality is not required. For example all classes required a transceiver, however, the control for a transceiver in a Class I SDR is much simpler than that required for a Class III and above. Thus, for those services/devices that indicate they are required for all classes further decomposition may required to establish a level of capability that must be provided. For some services/devices such as Black GP Processing no further definition is required.

The organization of the services/devices by function or some other method must be standardized or it will lead to significant effort during waveform porting.

3. API NAMING CONVENTION

Crucial to the achievement of software portability is the use of a standard naming convention and a data dictionary. The use of a standard naming convention includes not only the IDL definitions but also the XML profiles. The use of standard naming conventions also extends to the structure hierarchy of the IDL interface definitions. The hierarchy structure of the IDL ultimately defines the particular name of a class or function. In the transformation of IDL to the implementation language binding, the Module, Interface and Operation names are used differently depending upon the implementation language. The C++ mapping for IDL maps an IDL module to a C++ namespace and maps an IDL interface to a C++ class provided the C++ environment supports Namespaces. If Namespaces are not supported in the C++ environment but the C++ environment supports the use of nested classes then modules are mapped to C++ classes as well as the interfaces. If the C++ environment does not support Namespaces or nested classes then the mapping is defined following C language mapping of concatenating identifiers using an underscore ("_") as a separator. [4,5] Thus in defining an API, it is not sufficient to specify the interface name, its operations, and its attributes but also the module and its scope in which a particular interface is being defined must also be specified.

A data dictionary is used to describe the data elements being used in a particular system/program. Developers use the data dictionary to determine if a data element that they need has been defined previously or if they need to define a new data element and add it to the dictionary. The following are some simple naming convention rules [6].

Module and Interface names: 1st Letter uppercase and the 1st letter of concatenated words upper case. No underscores allowed.

Operation and Attribute names: 1st Letter lower case and 1st letter of concatenated words upper case. No underscores allowed.

The prohibition of underscores in the naming convention provides the IDL portability across implementation languages. As mentioned earlier some implementation languages mappings require the use of underscores to concatenate the IDL identifiers to form implementation language identifiers.

4. API CONTENT

To be useful an API must provide sufficient information to allow a third party to implement a component that uses the Service/Devices for which the API was defined without the support of the service/device developer. If it is too difficult to use an API, developers will implement their own version of the service, thereby developing redundant services, which lead to, increased cost and schedule.

The contents of an API should be more than a simple IDL listing of its interface name, operations and attributes. As indicated earlier in this paper an API must also be defined in a "context" by including the IDL module name within which the API is being defined. Additionally the API should include quality of service in terms of latency and jitter. Many datalinks have latency and jitter requirements and as a part of system design these latency and jitter requirements must be allocated to the SDR hosting the datalink. Thus the services/devices provided by the SDR must be characterized in terms of the latency and jitter to provide systems engineers the necessary information to make a determination whether the services/devices provided by a SDR are adequate to support a particular waveform or data link. In addition to quality of service, the performance of devices (MIPS, FLOPS, memory, etc) and whether the device supports sharing among waveforms and Management Infrastructure or if the device may only be exclusively allocated to a single waveform/application needs to be documented. The performance of the devices must be specified in terms of performance available to a waveform/application. Thus, for example, if the device is a general purpose processor (GPP) the performance available to the waveform/application is the GPP performance decremented by the overhead of the OE and the memory available is decremented by the memory consumed by the OE.

5. API DOCUMENTATION SET

The SCA API supplement specifies a structure for API documentation, requires the API be specified in IDL, but use of UML is optional. It is recommended that an API in addition to the IDL definition must also be documented using UML to model the dynamic and static characteristics of the API. The required UML documentation consists of the following diagrams: Class Diagrams; Sequence Diagrams, State Transition Diagrams; and Object Collaboration Diagrams. In addition as mentioned above the API documentation should include a quality of service description that enables a system engineer to determine whether the provided service/device is adequate for a particular waveform or data link. For a service performance specification, the performance should be stated in terms of the particular device hosting the service as well as in terms of the processing requirements of the service. The processing requirements of the service are of interest to determine if an existing service can be ported to a different device.

- Service/Device Description
- IDL Listing
- UML
  - Class Diagrams
  - Sequence Diagrams
  - State Diagrams
  - Object Collaboration Diagrams
- Quality of Service Characterization
  - Quality of Service Requirements
  - As implemented on Device
- Performance Characterization
  - Service Requirements
  - As implemented on Device

Figure 3. A Standardized API Documentation Set

5. API EXAMPLE

The Serial IO Package defined in the PIM and PSM for Software Radio Components [1] is used as the starting point for the specification of the Serial IO API. Figure 4 is taken from reference [1].

Reference [1] also contains a brief description of the interfaces and their attributes. Reference [1] does not specify any quality of service or performance requirements for the Serial IO. Reference [1] also does not specify any

naming conventions to enable a consistent extension of the PIM

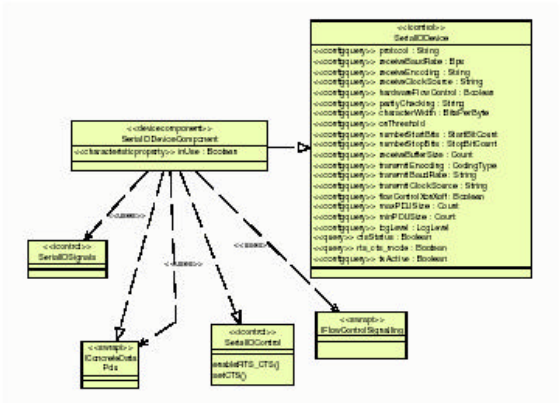

Figure 4. Serial IO UML Class Diagram

Figure 5 is taken from reference [1] to illustrate an IDL listing. In this listing SerialIO is declared as a module subordinate to the DfSWRadio and PhysicalLayer modules.

```
//File:DfSWRadioPhysicalLayer.idl
….
module DfSWRadio {
  module PhysicalLayer  {
    interface IOSignals {
      oneway void signal_RTS ();
   };
   module SerialIO {
      interface SerialIOSignals : IOSignals {};
      interface SerialIOControl {
        void enableRTS_CTS (
          in boolean enble
        );
         void setCTS (
           in boolean cts
        );
      };
  };
```

Figure 5. Example SerialIO API IDL Listing

The IDL in Figure 5 is used to illustrate the impact of the IDL structure and naming conventions on waveform portability. For example, given a C++ environment in which Namespaces and nested classes are not supported, the SerialIO module would be mapped to the following name DFSWRadio_PhysicalLayer_SerialIO [4]. However, if the implementer decided to partition their system differently, as indicated in Table 1, the SerialIO would be mapped to DFSWRadio_DigitalDataIO_SerialIO. At porting time, all of these naming issues have to be resolved. If the C++ environment supports Namespaces a similar issue at porting time would arise in reconciling the Namespace and the scope of the namespace.

For SerialIO, the state transition diagram is trivial since the SerialIO can transition from idle to transmit or receive; from transmit to idle or receive; and from receive to idle or transmit as shown in Figure 6.

The sequence and object collaboration diagrams are also trivial for SerialIO and are not provided.

The performance and quality of service for SerialIO should be stated in terms of the latency and jitter associated with the SerialIO device and the processor requirements to host the SerialIO device.

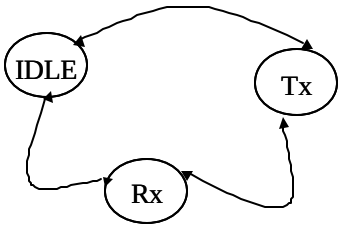

Figure 6. SerialIO State Transition Diagram

6. SUMMARY

The definition and promulgation of unencumbered APIs is essential to achieve the goals of the Software Communications Architecture. The acceptance of a Radio Set view for API definition is essential to waveform portability since a radio is expected to support a multiplicity of waveforms that exist now and in the future.

REFERENCES

[1] PIM and PSM Software Radio Components, Object Management Group, dtc/04-05-04, May 2004

[2] JTRS JPO Application Program Interface Supplement to the Software Communications Architecture Specification, JTRS-5000API V3.0, August 27, 2004

[3] D. Szelc, "API Position Paper" SDRF-I-030-V1.0, SDR Forum, June 18, 2003

[4] C++ Language Mapping Specification V1.1, Object Management Group, formal/03-06-03, June 2003

[5] C Language Mapping Specification V1.0, Object Management Group, June 1999

[6] E. Christensen, "Application Programming Interface (API) Discussion", SDR Forum, November 15, 2000