Software Defined Radio Forum

API Position Paper

System Interface Working Group

Document Number: SDRF-03-A-0005-V0.00

July 19, 2003

Goal of this Document

The System Interface Working Group (SIWG) feels it is necessary to provide input on the importance and development of APIs for SDR to the general community. We also would like to influence the API development occurring on the Joint Tactical Radio System (JTRS) program within the US Department of Defense, as this very large program is working to set standards for the entire SDR community. We propose a number of recommendations to the SDR community.

Introduction and Background

For modular, open-system, Software Defined Radios (SDR) well defined, commonly used, or standard, Application Programming Interfaces (APIs) are crucial.

An Application Programming Interface (API) is an agreement of services provided and required behavior among related software and/or hardware modules. It does not prescribe an implementation; is not a section of code or a program or an application. In many ways, this can cause confusion because an API is an abstraction and not a physical entity.

To allow the successful replacement of modules in different implementations (i.e. replacing a software-based module by a hardware version), the interfaces which define the boundary of the module must be designed in such a way so that the interface does not imply or exclude a potential implementation. If it does, then its appeal and its ability to attract components that conform and thus be reused will be limited.

The Need and Benefit of an API

1. Application Portability - An API is a defined set of function invocations that provide a level of abstraction between the application and the operating environment, including the operating system and/or other (Hardware) platform services. This enables portability of software applications among different platforms that provide the implementations of the same APIs.
2. Upgrade/Enhancement - A well defined set of APIs provide the syntax and semantics of message formats, procedure calls, and/or global data shared between entities. Software applications can be easily ported between systems that use common APIs. Furthermore, if the APIs are well known and understood, open to the community for use, then anyone can write code that can be hosted on hardware platforms that employ these APIs and thus can be easily added to numerous and diverse platform implementations.

What makes a good API?

A good API will have the following characteristics:

- It should be written so that it is understandable and testable, unambiguous and have only one interpretation
- It should be extensible and allow the use of proprietary implementations.
- It should have reasonable granularity within the system partitioning
- It should identify behavior under failure conditions
- It should support interface revisions and enhancements.
- It should adjust to match the available resources.
- It should describe the operation and its parameters.
- It should be independent of the transport mechanism.

What must an interface or API not do?

- It should NOT have two or more APIs or interfaces controlling a single resource without resource sharing management.

- It should NOT allow applications to go through APIs directly to lower levels.
- It should NOT be under-specified.
- It should NOT have knowledge of the implementation or any APIs below it.

Level of Detail of an API

The desired degree of application "portability" determines the number and granularity of APIs. Without a low enough level of granularity, the opportunities for multiple sources of components that can easily work together will be lost. A fine-grained approach provides developers with the freedom to mix, match, and combine modules like building blocks. The finer-grained the approach, the more overhead required.

Structure and Format of an API

The preferred method of encapsulating discrete functional elements in an Object Oriented domain is via "components" – a logical and/or physical (and replaceable) part of the overall system. A component's APIs should be defined using a graphical representation, such as the Unified Modeling Language (UML).

Waveform Portability Issues

APIs are found throughout a system at all levels where modules or components interface, hardware and software. Many APIs have been addressed to date and adopted into documents, such as the Joint Tactical Radio System (JTRS) Software Communications Architecture (SCA), to achieve standardization in implementations. Examples include POSIX for the operating system and minimumCORBA as a middleware. These serve to establish a common operating environment upon which to build a platform.

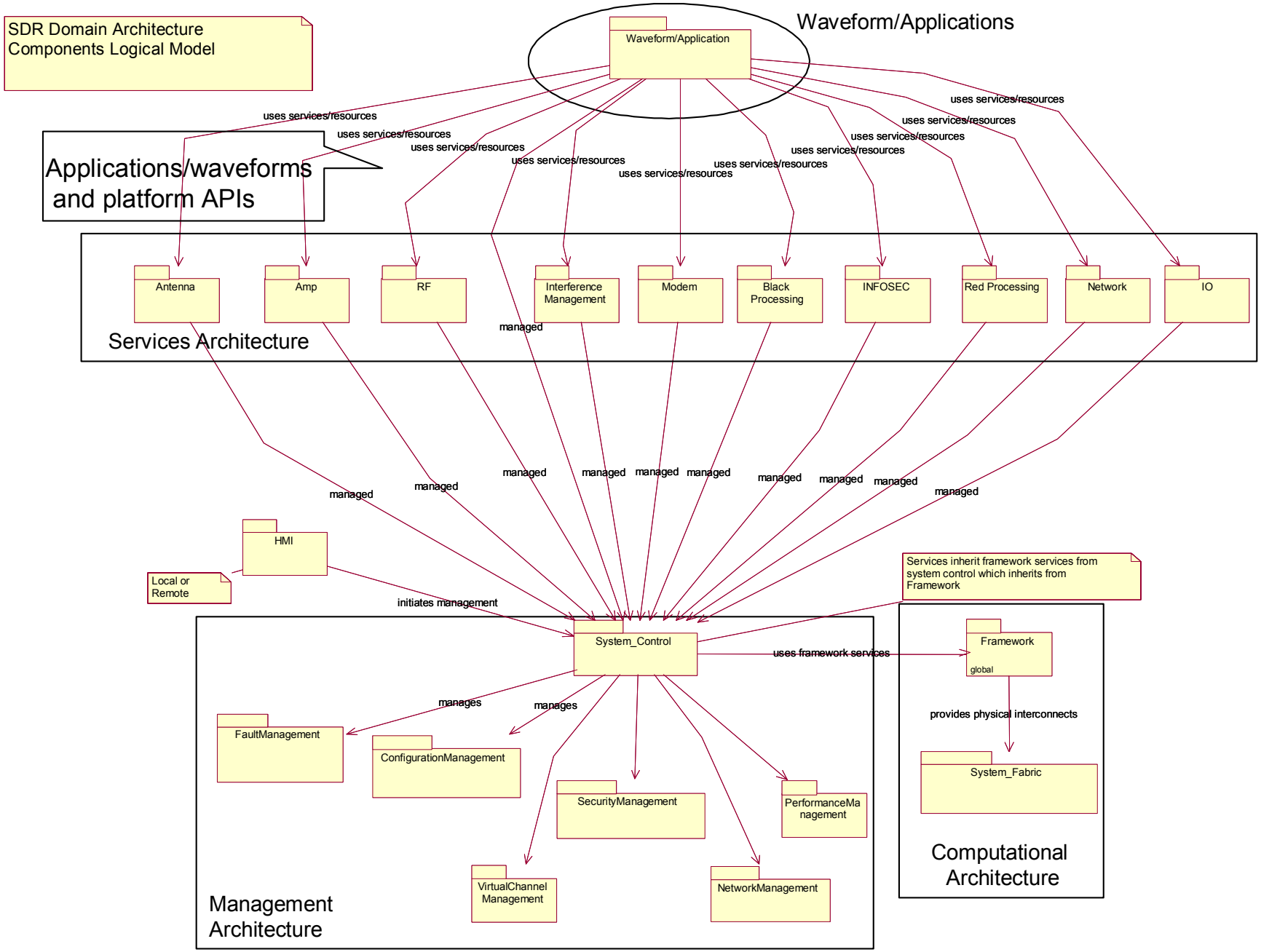
The JTRS SCA is a specification that separates the waveform "resources" from the platform "devices". The resource/device boundary defines the waveform API. The SCA allows developers to move most of their waveform functionality into the "devices" - such that the "resources" become very simple. Therefore, the Waveform API's could result in relatively simple API's, with much of the waveform functionality hidden within the "devices" severely limiting the portability. Furthermore, the API's are built from SCA API "building blocks" which are like templates. The API development procedures should require that developers create classes from the building blocks with consistent semantics and naming conventions applied across all waveforms. This is not currently being done for the JTRS program.

Recommendation #1

We recommend that waveform developers create a platform independent model of the waveform before the implementation is decided to avoid pushing all functionality into the devices thus limiting portability. When the platform specific implementation is created the developer should then provide rationale for the allocation of functionality to devices.

API Development

The SIWG has taken on the challenge to address the APIs that focus on the portability of applications. The focus is defining those APIs between an application and a platform. For reference the SIWG is using the diagram below to define the specific interfaces addresses. This model was taken from the Telecommunications Information Network Architecture (TINA) system reference model which describes, from a system perspective, a generic system architecture.


SDR Domain Architecture
Components Logical Model
Waveform/Applications
Waveform/Application
Applications/waveforms
and platform APIs
uses services/resources
Antenna
Amp
RF
Interference
Management
Modem
Black
Processing
INFOSEC
Red Processing
Network
IO
Services Architecture
managed
HMI
Local or
Remote
initiates management
Services inherit framework services from
system control which inherits from
Framework
System_Control
uses framework services
Framework
global
provides physical interconnects
System_Fabric
Computational
Architecture
manages
FaultManagement
ConfigurationManagement
SecurityManagement
PerformanceMa
nagement
VirtualChannel
Management
NetworkManagement
Management
Architecture

Recommendation #2

The SIWG has developed a draft SDR Interface Definitions document. The SIWG is also coordinating with the Object Management Group (OMG) Software Radio (SWRADIO) Domain Special Interest Group (DSIG) through a formal liaison relationship and participation of the common membership. We recommend that the SDR community in general and the JTRS JPO specifically, review both the SDR Interface Definitions document and the OMG SWRADIO submission as a starting place for the development of a common set of APIs. The SDR Interface Definitions document has been provided as input to the OMG submission for anticipated voting at the September 2003 meeting of the OMG.

SDR Dictionary

To ensure maximum understanding and utility of APIs, there should be an agreed upon set of terms and a naming convention.

There is currently no standard dictionary of radio terms for developers to reference. For instance, the SCA uses XML tags to control software deployment, but does not specify standard terms for waveform properties. There are several groups who could contribute to a standard set of terms for SDR:

- Mercury has proposed an XML dialect to the OMG SWRADIO DSIG;
- The OMG's SWRADIO DSIG has a Platform Independent Model(PIM) document which contains a set of terms for the SDR environment;
- South West Research Institute has proposed an XML dialect under DARPA MoBIES.

Tool vendors such as Mathworks (MATLAB/SIMULINK) and Elanix (Systemview) would be interested in such a dictionary, since they already have their own versions for radio terms.

The System Interface Working Group, in cooperation with the SWRADIO DSIG, will develop a naming convention and dictionary of SDR terms. This common set for SDR could be used across the industry. In the development of naming conventions the practice of using long terms promotes clarity, terms should rarely be abbreviated. The SDR Forum will publish a dictionary that defines common and agreed upon names.

Recommendation #3

We recommend that the SDR community adopt a common naming convention and dictionary of terms. For the JTRS program this could be added as an annex to the SCA or through a reference to a current standard dictionary.