We present an architecture for organizing the programmable sub-systems necessary to configure and operate software defined radio modems. To aid in the architectural organization a classification system for modem components is developed. The architecture also addresses the organization of software components registered within the system. Library modules are developed to aid in the component management and organization. The architectural model addresses the of life-cycle issues of managing, configuring and operating programmable radio components.

Keywords: Software Defined Radios, Cognitive Radios

1. Introduction

Software defined radios (SDRs) integrate several advanced technologies to bring agility to radio communications. These technologies include wide-band radio frequency components, programmable radio frequency components, high performance analog/digital converters, and significant signal processing capacity. Through these advanced technologies, SDRs enable opportunities to develop better spectrum utilization, new markets in spectrum resources, and enhanced services. With this new flexibility comes a set of challenges in how to configure and operate SDRs so as to avoid interference, operate within a regulatory system or systems, and provide robust services. The primary challenges we have identified in this paper are: (1) assembling hardware and software components into an operational radio; (2) controlling the radio during operation; (3) insuring the radio operates within physical and regulatory constraints; (4) self-configuration of radio components; (5) presentation of common interfaces to applications and management software; and (6) managing component libraries and radio configurations for use on multiple platforms.

A proposed software architecture for SDRs is the Software Communications Architecture (SCA) [1] from the Department of Defense Joint Tactical Radio Systems (JTRS) [2] program. The SCA is designed to provide an open architecture that enables radio designers to combine hardware and software components into an operational radio. In one sense, the SCA is an “operating system” defining how software components of an SDR interact. However, the SCA approaches SDRs from a software perspective and not from an implementation, operational, or life-cycle perspective. Although the SCA approach enables flexibility in many of the software components of a SDR, many issues and challenges in implementing the software components and operating SDRs are not addressed. The architecture presented in this paper addresses those issues and challenges.

2. Component Classification

The SCA specification provides a starting point when addressing the first primary challenge we have identified - assembling a hardware and software components into an operational radio. The goal of the SCA from the specification reads: “to ensure the portability and configurability of the hardware and software and to ensure interoperability of products developed using the SCA”. From this view the main theme of the framework is focused on the portability of the radios and maximizing the reusability of the components in the system. For future radios which have a large dynamic framework, such as cognitive or environmentally aware radios, the focus of the architecture needs to move from being portable and reusable to having a solid framework to support the dynamic changes. Portability and reusability are still needed to create an operational cognitive software defined radio, but developing dynamic software radios where components are changing frequently causes issues with component management that need to be addressed.

From a hardware perspective, most independently developed software radios use similar hardware for the RF and modem sub-systems. For example, most radio systems include a modulator, an antenna system, a mixer, and an amplifier. From this assumption we can group common hardware and software components into specific classes. From these classes, APIs can be developed to provide a basic function set for the individual classes. Providing the developers of hardware and software components with a component classification system not only promotes the portability of the components by
forcing components to provide basic functions, but allows the cognitive radio to clearly identify a basic set of services provided by the component by simply identifying the component’s classification.

Such classifications include: amplifier, mixer, demodulator, modulator, antenna system, filter, and several others. The set of classification types must be a standardized set in which developers must agree to classify their components into. Essentially a simple API is developed for each classification. This API includes only the basic functions and attributes that every component with the specific classification must provide to the radio system. Table I below provides an example API for an amplifier component classification:

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dutyCycle</td>
<td>attribute</td>
<td>Specifies duty cycle for specific amplifier.</td>
</tr>
<tr>
<td>maxGain</td>
<td>attribute</td>
<td>Specifies the maximum operational gain for specific amplifier.</td>
</tr>
<tr>
<td>minGain</td>
<td>attribute</td>
<td>Specifies the minimum operational gain for specific amplifier.</td>
</tr>
<tr>
<td>gain</td>
<td>attribute</td>
<td>Current operating gain.</td>
</tr>
<tr>
<td>setGain</td>
<td>function</td>
<td>Function provided to set the current gain.</td>
</tr>
<tr>
<td>getGain</td>
<td>function</td>
<td>Function provided to get the current gain.</td>
</tr>
<tr>
<td>getDutyCycle</td>
<td>function</td>
<td>Function provided to get the duty cycle.</td>
</tr>
<tr>
<td>getMaxGain</td>
<td>function</td>
<td>Function provided to get the maximum gain.</td>
</tr>
<tr>
<td>getMinGain</td>
<td>function</td>
<td>Function provided to get the minimum gain.</td>
</tr>
</tbody>
</table>

In addition to the control functions the components provide, the attributes that allow external control must have operational limits. As Table I shows, each controllable attribute has a maximum and minimum limit. These limits must be provided to ensure the hardware components do not operate outside of its operation range which could possible damage itself permanently. These limits may also be provided to ensure the radio components stay within the legal limits according to the geographical location of the radio system. FCC policy would be read into the system allowing the maximum and minimum operating limits to be set accordingly as the components are instantiated. The control path structure used to allow the interaction between FCC rules and the components is covered in Section 3.

These functions and attributes would be specified in addition to the general API provided by the SCA specification which includes interfaces for querying, starting and stopping the component.

In addition to the required function and attribute set to define a fully functional hardware or software component, the developer may want to include additional functionality to take advantage of special component specific features. For example, in addition to the amplifier functions provided above, a developer may want to include functions which allow the system to query the operating temperature of the amplifier, or an attribute to identify the physical size of the component. These additional features are needed to take full advantage of the components features if needed. Component specific functions and attributes may be defined within an API profile for the software or hardware component. This API profile can be embedded in the SCA specified SCD for the component.

Component classifications enable software radios to become more cognitive by increasing the self-awareness of the radio services. Providing an API improves presentation of common interfaces to both applications and management software.

Currently similar work is being done by the Object Management Group (OMG) [3] which has developed a Platform Independent Model (PIM) [4] which provides UML profiles for software radios. The UML profiles are developed to allow automatic generation of the SCA XML component descriptor files. The OMG’s work focuses on not only the components and devices in a radio system but also the core infrastructure and application interfaces. A set of stereotypes is provided that provide a simple classification system for hardware components. The PIM does not specify functions to query system attributes or provide any mechanisms to provide limits on the operational parameters.

3. Software Architecture

To demonstrate the classification of the SDR components, an SCA based architecture model is developed. The model was built around the SCA specification and is used to identify components needed when implementing an actual cognitive radio system framework. One component used in this model is a library management module to provide an organized administration component for the multitude of files needed for an SCA application. The model also demonstrates the use of the component classification system defined in section 2. The API classification provides a more descriptive component blueprint for component developers to follow to allow a component to interface with the architecture. This extended blueprint improves the self-awareness of the system which in turn improves the cognitive abilities of the system by allowing the radio system to know what services are available from the component and how to access them simply by knowing the classification type. Note that the components developed using the classification APIs are
still compliant with any SCA architecture because the classification APIs are simply an extension to the device API defined in the SCA specification.

The architecture uses a “Smart Controller” to parse the control from the application layer. From the SCA perspective the smart controller also contains the domain manager functionality. Control data flows from the application to the smart controller where it is checked against a rules guide to ensure the radio system is operating within the physical and legal limits of the environment. The rules guide can be generated from the operational limits of the components currently active in the system. These parameters are retrieved from the components using the getXXX function calls. The Smart Controller knows which functions are available because it knows the types of components currently active. The Smart Controller also contains a security module to check the applications access level. Without some basic type of security module, an application could create malicious software modules, remove all modules in the system or could operate the system outside of the legal operating limits.

The architecture contains two device managers. The device managers are responsible for the management of logical devices and their services. One device manager is responsible for only the digital software modules, such as an FPGA module or encryption modules. The second device manager is responsible for only the RF components in the system, such as the antenna system or an analog amplifier device. The separation between the RF and digital devices allows the RF device manager to manage the hardware components and the digital device manager to manage the software components. Although the device managers provide the same functions to the devices, the separation seems intuitive.

The device managers create an operational radio system by reading the software application description (SAD) file. The SAD as defined in the SCA specification defines how radio components are interfaced to create the actual waveform. Dynamic radio systems may contain several SADs to allow the system to be compatible with multiple waveforms. These files are managed by the software application configuration manager module. Using this module improves the organization of the system along with improving the ease of reconfigurability of the radio system. The SAD configuration manager provides functions to add, remove, or query the SADs. The bulk of the functionality comes from the query function which allows the smart controller to discover the components currently active by querying the active SAD.

The SAD is parsed and the logical devices are set up by the device manager for each component specified in the application description. The digital logical devices create instances of the software components using information stored in the component library. The RF logical devices create an instance of the hardware driver used to communicate to the RF component. This information to create an instance of the driver is also stored in the component library. This library holds the registered components available to the radio system. The library is managed by the library component manager which is responsible for the addition and removal of components in the system. Applications may also query the library manager to receive a list of available components and default component information.

The component library is not only used to provide a better organizational framework for a software radio system, but also to improve the autonomy of the radio. The interface between the library component module and the Smart Controller allows the Smart Controller to determine the components and services available to it. In a cognitive radio, the component and service availability information and the user preferences is enough information for the cognitive engine to determine the most efficient waveform setup to complete a task. Without some sort of library management system, adding, removing, and querying the components in the system would consist of manually listing the files in a directory and determining this from the file name. In an SCA compliant system each device uses 3 XML files to completely describe itself. Complex radio systems could contain from 10 to 100 devices for one waveform. Unfortunately, the relationship between the complexity of the radio system and the amount of descriptor files is linear, causing a need for some type of organizational module to provide a simple interface for the addition, removal and informational querying of components.

The logical devices control their respective devices through a CORBA interface as defined in the SCA specification. Using the API classifications, the logical device only needs to know the type of device it is representing in order to access the basic function set for the device. This allows a logical device representing an amplifier type device to control any amplifier as long as the device driver for the amplifier uses the amplifier classification API. For a specific component, with the base functions only being provided by the developer, the
logical device would be a simple template that would not be needed to be generated real-time. With the additions of features, only the extra features are added to the base template when the component is instantiated.

From the architectural model previously defined, we can demonstrate an example scenario that illustrates how the radio system can take advantage of the component classification system.

In this scenario Radio A and Radio B are communicating over a wireless link. Radio A is a cognitive software radio with all components developed using the component classification API. A decision is made by the Radio A cognitive engine that the link quality is too low and could be raised by increasing the gain of the system. The first problem is how to determine which components need to change to simply change the gain. The second is what functions need to be issued to control these components. The classification system solves the second problem and only this second problem has been addressed in this paper.

5) The logical device recognized the command and sends a setGain(highGainVariable) command over the CORBA middleware to the amplifier device driver.

Once the radio system has determined the problem and solution, the component classification is used to provide an efficient communications interface between the components of the waveform and the core system components. This simple classification system eliminates the need for the core radio components to discover the correct component function calls for the most basic features of the component.

4. Conclusion

This paper develops an SCA-based model to demonstrate the improvements component classifications provide for cognitive radio systems, and to provide a model for assembling the hardware and software components of a cognitive radio architecture. The model also introduces a component management module and a configuration management module to aid in the management of the large amount of files needed to define SCA radio components.

APIs are developed by taking advantage of common classifications of components. These APIs give a more descriptive layout of what radio component interfaces should include, and provide the radio system with a core set of functions needed to operate components in their most basic state. This in turn, allows the radio to configure itself without the need for component function descriptors, and allows for standard control functions to be developed for each classification type. Using these APIs in addition to the SCA defined device interfaces improves the cognitive awareness of the radio systems by allowing the radios to become more aware of the services available to them by simply knowing the classification type of the components active in the system.

5. References