A Lightweight Operating Environment for Next Generation Cognitive Radios

Gómez, Ismael; Marojevic, Vuk; Salazar, Jose; Gelonch, Antoni
Dept. Signal Thery and Communications, Universitat Politècnica de Catalunya
ismagom@gmail.com; {marojevic;jose.salazar;antoni}@tsc.upc.edu

Abstract

It is widely known that the SDR industry campaigns component-based radio applications, which will enable fast prototyping and deployment of new radio devices and may increase manufacturing profits. Through the JTRS program, the US Dept. of Defence proposed the SCA specification as the standard for military communications. The SDR Forum is now reviewing these specifications and trying to adapt them to the commercial market. The significant differences between military and commercial communications' requirements make this migration a hazardous task. On the other hand, the SCA specification does not consider any method or procedure that enables cognitive functionalities, which would be necessary for future cognitive radio implementations. This paper therefore presents an alternative approach to SCA, introducing a low-profile operating environment for next generation cognitive radios. We demonstrate its suitability for present and future commercial radios.

1. Introduction

It is generally understood that a cognitive radio system addresses spectrum management issues and related problems [1]. Cognitive radio, however, stands for a concept of much wider scope: it introduces intelligent radio devices that may reconfigure themselves to increase the systems' capacities, users' satisfactions and operators' profits. More precisely, the flexibility introduced by software-defined radio (SDR) and a cognitive system can improve the efficient usage of radio and computing resources [8]. In such a scenario, the system would observe its operational environment and learn from its reconfiguration decisions (cognition, intelligence), while taking advantage of its reconfiguration capabilities at all system layers (SDR). This implies automatic adaptation to the momentary communication status and user requirements, that is, without user interventions. Hence, above from features related with the communication system itself, other aspects, such as the underlying infrastructure, need to be taken into account. The high dependency between the wireless system and the hardware that supports it justifies this direction.

The reconfiguration flexibility of SDR platforms is a powerful tool of cognitive radio systems. These platforms consist of general purpose processors (GPPs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs), pico arrays, networks on a chip (NoCs), multiprocessor systems-on-chip (MP-SoCs), or a mix of them. Moreover, today's reconfigurable devices, including arrays of processors, offer high computing capacities at moderate power consumptions.

Many efforts to define a software framework that is capable to efficiently manage the adaptation of waveforms have been achieved in the past decades. Only a few of these try to provide a complete development framework, integrating development tools for FPGAs, DSPs, and so forth, and address portability issues. The most relevant framework is probably the software communications architecture (SCA) proposed by the US Dept. of Defence and adopted by the SDR Forum and many research groups (section 1.2). The flexibility of such kind of frameworks entails the cost of additional resources that are needed to control the reconfiguration process. In addition, monitoring, management, and actuating elements must be incorporated for a cognitive resource management, further increasing the computational requirements of the system.

1.1. Cognitive Radio System

We suggest a cognitive radio system based on SDR and which tries to jointly optimize the different resources at all system layers. Figure 1 shows how the cognitive entity is in charge of handling every aspect related with the complete set of resources, including

radio and computing resources. The middleware or execution environment –a layer between the application and the platform– features several control mechanisms and provides support to the sensing and monitoring entity.

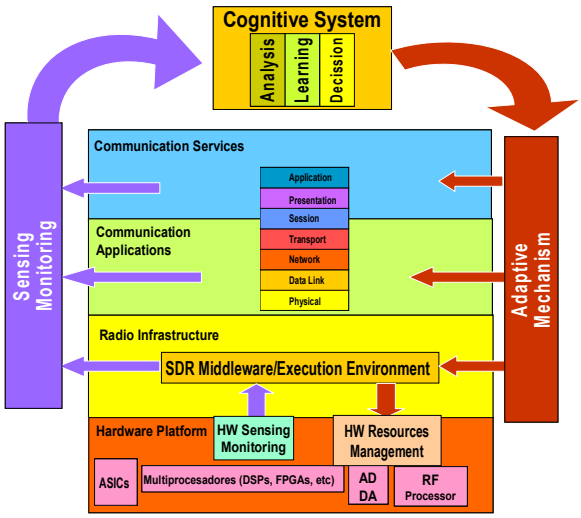

Figure 1. Full Cognitive Radio system view

Hence, a next generation cognitive radio system requires a middleware or execution environment that can fully access the hardware and provide the full functionality to reconfigure it (a part form supporting the proper execution of the radio application).

The need to operate in an efficient, reliable, and secure mode points out several requirements to the framework's management and control architecture. The following list provides some of these functionalities.

- Monitoring and detection of the available access networks;
- Support of the dynamic and easy incorporation and/or substitution of components of any protocol;
- Validation, diagnosis, and error control of the own system;
- Control and coordination of the reconfiguration of the components of the different equipments;
- Data interface management support;
- Well defined interactions and interfaces with external entities, such as network entities;
- An integrated development framework.

1.2. SCA Overview

The SCA specification defines an operating environment (OE) that will be used by JTRS radios. It also specifies the services and interfaces that the applications must employ. The interfaces are defined by using the Common Object Request Broker Architecture (CORBA) Interface Definition Language (IDL), whereas graphical representations are made using the Unified Modelling Language (UML).

The OE consists of a Core Framework (CF), a CORBA middleware, and a POSIX-based Operating System (OS) (see software stack in Figure 2). The CF describes the interfaces, their purposes and their operations. It provides an abstraction of the software and hardware layers for software application developers. An SCA compatible system must implement these interfaces.

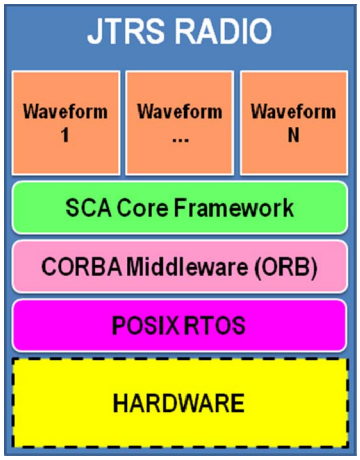

Figure 2. SCA Software Stack

The software communications architecture was introduced to reduce the overall software manufacturing costs by reducing the development time, enabling reusability (from one design to another), and portability (from one platform to another) of software pieces. This concept was inherited from the OOP (Object Oriented Programming), which has been enriched in the past decades by world-wide computer programmers. Furthermore, CORBA exploits and expands these concepts, enabling distributed computing among heterogeneous environments. This means that a software component that executes on one processor can send a message to another component running on the same or another processor; no difference will be appreciated from the point of view of the component programmer.

A natural step was to translate all this general-purpose computing knowledge and experience to the digital signal processing world. However, different arguments must be taken as premises in radio communications: timing considerations (real time and QoS constraints), event synchronization with external devices, minimum data transfer overhead, and also the fact that most of signal processing engineers are used to medium efficient level programming languages, such as plain C.

1.3. SCA Limitations

The SCA specification does not consider methods to allow the radio to be aware of its internal computing resources (processing resource occupation, remaining battery life, etc.) or the waveform's actual status (BER, noise power density, etc.). It is because the underlying POSIX-RTOS does not (and will not) support those functionalities. The SCA specification has been designed to provide dynamic reconfiguration services; no further requirements were imposed.

However, an SCA-compatible CF can be designed where these features are available: consider a tool interfacing, on one side, the SCA framework using an SCA-based interface and, on the other side, functions that facilitate gathering resource information. This solution would be platform-dependent because the methods used to gather hardware information are out of the scope of the specification and are not supported by POSIX. Thus, portability to other architectures is not ensured.

Another issue is that SCA currently does not support any features that guarantee QoS. It is rather focused on dynamic reconfigurability [9]. Moreover, it does not focus a CPU *time* (a computing resource, in CPUs) management approach, i.e. the system can not dynamically assign a certain amount of CPU time to one or another process. This makes it impossible to apply real-time oriented software mapping algorithms (e.g. t_w-Mapping [8]), which dynamically assign software components to processing resources while satisfying real-time and QoS requirements.

Besides these difficulties, others have been broadly reported, as delays suffered by the IDL interfaces [2], large foot-print of existing commercial OE implementations [4], dynamical deployment of application components [5], etc.

2. P-HAL-OE: An OE for Cognitive Radios

Considering the SCA limitations and the features required by commercial cognitive radios, a lightweight Platform-Hardware Abstraction Layer (P-HAL) has been developed for future SDR and cognitive radio devices [6]. The concept has been proved through implementations over heterogeneous platforms (Linux for GPPs, 64xx and 67xx DSPs, and a reduced version for FPGAs [7]) and the execution of various SDR applications.

The new version of the framework, the P-HAL-OE, has experimented an internal architecture reorganization (written software is still backward-compatible) in order to ease the portability to other platforms and to separate its functionalities into well-defined autonomous blocks, which can be distributed among a set of processors and thus enable a *distributed radio management*, as well as other (cognitive) features.

2.1. Framework Architecture

P-HAL-OE defines a structure for interoperability of independently developed P-HAL-OE software and for portability of P-HAL-OE routines from one platform to another. In order to achieve this, it is possible to identify a large number of tasks that do not change from one platform to another, whereas others are platform dependent. The larger the number of functions that are platform independent, the easier it is to port the P-HAL-OE from one platform to another. Conversely, the larger the platform dependent part the more difficult it is to adapt the services to a new platform as services become more complex with increasing software sizes. Therefore, the P-HAL-OE platform dependent part will define elementary services that can potentially be implemented with low cost and a low software depth. Figure 3 shows a schematic view of the different P-HAL-OE components and libraries. The top-left level of the stack depicts the application software (represented here by a single P-HAL-OE application object), which only uses P-HAL-OE functions to interact with its environment. These P-HAL-OE functions are called inside the different objects in the application and their implementation is found within the P-HAL-OE software library, being a platform independent library. The basic operations within the software library may require in-depth platform or hardware management. At this level, the P-HAL hardware library needs to keep the software library isolated from the platform. Any P-HAL hardware library uses all the necessary platform services (OS if present) and hardware and provides them to higher P-HAL layers. Hence, the P-HAL hardware library is the platform dependant part of P-HAL.

The top-right of Figure 3 shows a representation of the software components (P-HAL Software Daemons) that belong to the P-HAL-OE. They perform several tasks to successfully run the user application or to support a distributed management. The implementation of these components is platform-independent and so directly portable to other platforms (as soon as the hardware library is available). In continuation we provide a short description of this group of Software Daemons. (*MAN* is used as an acronym for *MANAGER*.)

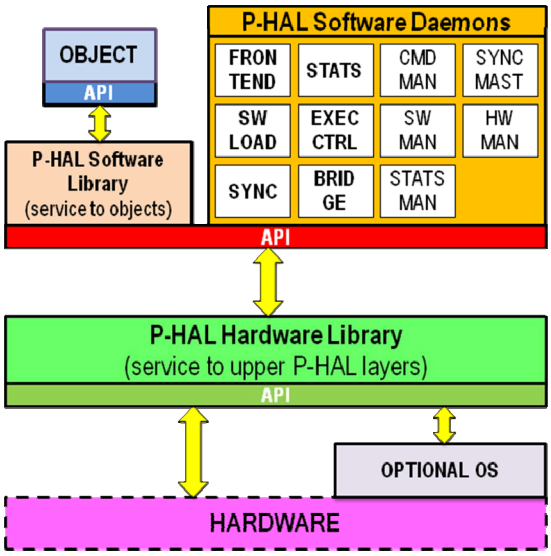

Figure 3. P-HAL-OE Components

- *CMD MAN:* Centralizes the interactions with higher level control applications and P-HAL-OE. For example: GUIs, algorithms, user text-commands, development tools.
- *HW MAN:* Performs computing resource management tasks, assigning software requirements to computing resources.
- *SW MAN:* Manages application definition (component connection graph) and component repositories.
- *STATS MAN:* Provides initialization parameters to the application objects and captures the evolution and modification of application variables.
- *BRIDGE:* Acts as a link for data transfers between processors.
- *SYNC MAST:* Provides the local time reference to remote processors.
- *FRONT-END:* Routes and bridges P-HAL-OE control packets among the rest of the daemons and gathers hardware status information.
- *SW LOAD:* Assigns local resources for loading software components and creates internal data interfaces between them.
- *EXEC CTRL:* Ensures that every software component is correctly running under the given real-time constraints.
- *STATS:* Captures and modifies an application's component variables as indicated by STATS MAN.
- *SYNC:* Synchronizes the local times with the remote time reference provided by SYNC MAST.

2.2. Cognitive Functionalities

The term *cognition* refers to the capability of knowing or being aware of the (cognitive) entity's internal or external environment. A *cognitive radio* is thus a radio that features a set of tools or procedures for detecting the users' communication needs and for providing radio (and computing) resources that are most appropriate to satisfy these needs.

In relation with such definition, P-HAL-OE can divide its cognitive functions in two groups:

Computing Resource Management: The system needs to be aware of its internal status and architecture at any time: plug-and-play network discovering (plugged/un-plugged processors must be automatically recognized at run-time), distributed processing resources (time, area, power…), processor-internal parameters (power, battery, malfunctions…), and so forth.

Application and Execution Management: This includes the application's variables acquisition and their modification; realtime execution surveillance; autonomous application and component repository management, and application execution control.

On the other hand, all cognitive functions are executed by Software Daemons. They are grouped in two types depending on the "intelligence" level:

Manager Daemons: These are intelligent elements that do not directly access environmental variables or parameters but make decisions as a function of their values and the predefined procedures or methods.

Sensor/Actuator Daemons: These are non-intelligent elements that provide a direct access to environment variables and parameters. The interaction is bidirectional, allowing capture and modification of values.

The top view of P-HAL-OE functionalities is illustrated in Figure 4. Is shows a 2D space where the vertical axis represents the level of intelligence and the horizontal axis hardware (left) to software (right) space.

The non-intelligent entities at the bottom of Figure 4 are not allowed to horizontally communicate with one another. The information they gather is only reported to its immediate manager, which governs their actions. This separation is very useful to clarify and understand the functionalities of the system and their interactions. The intelligent pieces, the managers, can communicate with other managers; this facilitates a common management approach. A higher intelligent entity (user, GUIs, debugging tools, CCRM applications, etc.) serves as a centralized interaction gate towards the whole framework, the CMD MAN Daemon. Such an interaction is currently supported through text-only commands, although, some standard API interfaces could be implemented (Java, Python, or C++, among others) to enable the development of GUIs or debugging tools.

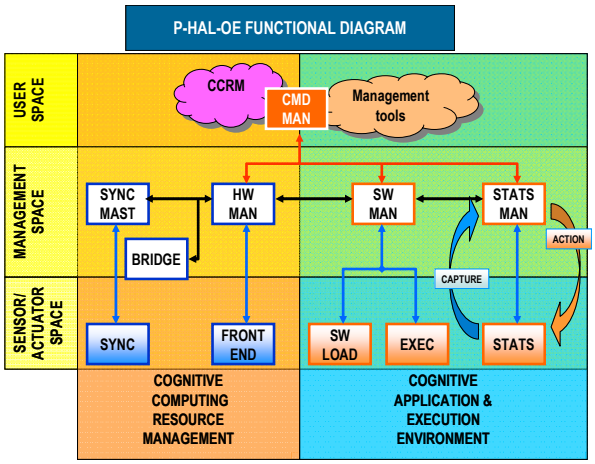

Figure 4. P-HAL-OE Functional Block Diagram

2.3. SCA vs. P-HAL-OE: Conceptual Differences

We find the following key differences between the SCA and the P-HAL-OE (see also Table 1):

- P-HAL Hardware Library is equivalent to SCA's POSIX layer. However, it has been specifically designed for this purpose, providing several advantages: (1) it is very simple and specified for this context, thus introducing low footprint and minimal overhead; (2) direct support to obtain hardware information to enabling cognitive functionalities; (3) due to its simplicity the portability to resource-constrained architectures, such as DSPs and FPGAs, is easier than designing custom POSIX-compliant support.
- P-HAL Software Daemons are equivalent to the *DomainManager*, *DeviceManager*, *ApplicationFactory* and *FileManager* entities found in any SCA CF. Nevertheless, the separation that P-HAL realizes through its Software Daemons is more strict, making future improvements, upgrades, or feature-adding easier and more consistent.
- P-HAL-OE enables a distributed management of its environment or, in other words, its functionalities are non location-constrained. This means that not every processing element must support all P-HAL-OE functionalities; for example, DSPs or FPGAs do not need to realize management tasks such as software repository access or software components mapping. These tasks should be reserved to non-constrained processors (GPPs) if available. P-HAL-OE eases this easily: Since functionalities are assigned to P-HAL software daemons, not launching one disables the corresponding set of functionalities. This reduces the footprint and other resources.
- Interactions between objects and with lower layers are completed through API's instead of using CORBA IDL interfaces. This contributes to reduce the overall run-time overhead in data transfers and in control calls [2]. Moreover, thanks to the pipelined architecture of the execution scheduling, data transfer overhead can be completely eliminated (at least in terms of time) if DMA controllers are used.

Table 1. SCA vs. P-HAL-OE Feature Comparison

Topic	SCA 2.0	P-HAL-OE
Process Scheduling (assignation to CPU resources)	Undefined. Best-effort, almost Real-Time in some cases (RTOS)	True Real-Time. Processes are assigned a determined amount of CPU time (resources)
Processor Resource Management	None. Any tool available to manage processing resources.	Yes, functions available to gather processing resources information and to manage and assign them to processes.
Deployment Mapping	Static, depends on processor family, available devices, etc.	Real-Time oriented. QoS guaranteed. Multiple constraints possible
Dynamic Computing Resources Management	Partially supported. Application definition (XML) must be reparsed and their interfaces recompiled (IDL compiler)	Yes, supports run-time data flow reconnection and module reconfiguration.
Data Interfaces	CORBA IDL	FIFO-like. Packet oriented
Run-time variables access	Same way as data interfaces (IDL)	Variables are externally accessed. Information is centralized.
Portability	Very high although requires underlying ORB middleware and platform-specific POSIX RTOS	High, requires specific Hardware Library implementation
Distributed Management	Yes, in some cases. It is sometimes difficult to isolate functionalities from one platform to another	Yes, each processor runs just the desired management tasks.

2.4. SCA vs. P-HAL-OE: Performance Comparison

After presenting key differences between both specifications in terms of functionalities, this section

provides a performance analysis. Since GPP platform's overhead (e.g., using Linux) is negligible in all aspects (time, memory, power, etc.), only the DSP (Ti C64xx) implementation will be considered. DSP-based platforms are very important for wireless communications. Their resource limitations have obliged developers to optimize their designs.

The SCA-based implementation OSSIE [2] is considered for this comparison, in particular its implementation on a TI6416 DSP [9]. P-HAL-OE has been implemented on the same DSP. Table 2 provides the comparison study.

Table 2. SCA vs P-HAL-OE DSP Implementation Performance Comparison

Magnitude	OSSIE	P-HAL-OE
Total framework memory footprint	1426 kBytes	50 kBytes
Interface packet delay	Variable. Average 6.9 cycles per 32-bit word when a packet length of 1024 words is used for internal communications.	Deterministic and adjustable. Components execution is pipelined. Delay is independent of packet size or destination location
Framework run-time overhead	Unknown. POSIX system calls time delays are unpredictable	Restricted to less than 10% of total CPU time

Besides the huge difference in the total memory foot-print (very important in SDR environments), we appreciate the time accuracy of P-HAL-OE. Pipelined process scheduling and the fact that P-HAL restricts background processing tasks to less than 10% (of the total CPU time) provides the designer with the certainty of what his application could suffer from when implementing his waveform in a component based environment, instead of an ad-hoc one (without a framework).

In addition, SCA-based radios experience from unpredictable delays because of the presence of unconstrained lower software layers (CORBA middleware and POSIX) [3]. These delays could be harmful in certain situations, such as in TDMA-based systems (e.g. GSM), where time slots must be filled at exact times.

3. Conclusions

This paper presents an operating environment which faces the challenges for next generation (cognitive) radio communications. It is able to assure real-time waveform requirements with limited computing resources. Moreover, minimum overhead as the design premise leads to a lightweight implementation, with very low memory footprint and deterministic time overhead. A comparison with SCA-based SDR frameworks has demonstrated the main advantages of our proposal. Finally, a component-based design, having cognitive functionalities and tools for the management of all system resources, will characterize P-HAL-OE as an efficient and easily extensible operating environment for next generation cognitive radios.

4. References

[1] J. Mitola III, *Cogntivie Radio: An Integrated Agent Architecture for Software Defined Radio*, PhD Thesis, Royal Institute of Technology (KTH), May 2000

[2] OSSIE, MPRG, Wireless group at Virginia Tech, http://ossie.wireless.vt.edu/trac

[3] J. Bertrand, J. W. Cruz, B. Majkrzak, T. Rossano, "CORBA Delays in a Software-Defined Radio", *IEEE Communications Magazine*, Feb. 2002

[4] D. Oldham, M. Scardelleti, "JTRS/SCA and Custom/SDR Waveform Comparison", *Military Communications Conference*, MILCOM 2007

[5] S. Kim, J. Masse, S. Hong, "Dynamic Deployment of Software Defined Radio Components for Mobile Wireless Internet Applications", *Proceedings of International Human, Society and Internet*, 2003

[6] A. Gelonch, X. Reves, R. Ferrús, "P-HAL: A Middleware for SDR Applications", *2005 Software Defined Radio Technical Conference and Products Exposition (SDR'05)*. Orange County, California. November 14-18, 2005.

[7] X. Reves, V. Marojevic, R. Ferrus, A. Gelonch, "FPGA's Middleware for Software Defined Radio Applications", *FPL* 2005.

[8] V. Marojevic, X. Reves, A. Gelonch, "Cooperative Resource Management in Cognitive Radio", *Proc. IEEE Int.'l Conf. Communications (ICC'07),* Glasgow, 24-28 June 2007, pp. 5939-5944

[9] J. Lee, S. Kim, S. Hong, "Q-SCA: QoS Enabled JTRS Software Communications Architecture for SDR-based Wireless Handstets", *Real-Time Systems*, Volume 34, Issue 1, pg. 19-35, Sept. 2006

[10] C. Aguayo, F. Portelinha, J. Reed, "Design and Implementation of an SCA core framework for a DSP platform", *Military Embedded Systems,* March 2007