# RESOURCE ALLOCATION IN SOFTWARE RADIOS USING CCMs BASED ON THE SCA

Srikathyayani Srikanteswara (Mobile and Portable Radio Research Group (MPRG), Virginia Tech, Blacksburg, VA; swradio@vt.edu);
James Neel (MPRG); Jeffrey Reed (MPRG); Shereef Sayed (MPRG)

## ABSTRACT

Reconfigurable hardware enables hardware paging, which in turn adds an additional dimension to the resource allocation problem. Software radios that use non-allocable hardware like general purpose processors attain flexibility mainly through software, which can sometimes lead to a non-optimal hardware implementation for the system in use. Reconfigurable hardware like FPGAs and Configurable Computing Machines (CCMs) provide the potential for further optimization of the hardware to meet system requirements. In addition, CCMs allow hardware configurations to be rapidly paged in and out of the system so the hardware can be continually optimized for performance providing the opportunity for performance that dramatically exceeds traditional processing solutions. However, the SCA does not currently support CCM implementations. In this paper we outline the process required to include CCMs into the SCA. Although we suggest some modifications to the Core Framework (CF), these changes are straight-forward to implement and impose a minimal impact on the existing CF components. Since CCMs present an important class of reconfigurable hardware that are especially well-suited for software radios, their inclusion in the SCA, the emerging standard for software radios, will provide a major breakthrough for designing efficient and flexible radios.

## 1 INTRODUCTION

The SCA is emerging as the unifying standard for software radios, bringing them a step closer to commercialization and widespread use. The SCA is based on an Open Systems Architecture and sound object oriented design principles that facilitate modular paging of systems and algorithms within the radio. Presently, this paging is performed only in software. However, a more optimal software radio solution should be able to tune its hardware to the system in use, thereby optimizing power, silicon area, and hence form factor. Such optimization should help overcome the major hurdles for SCA implementations in the handheld domain, which have especially stringent size and power requirements. Since such optimal systems are still hard to develop, the designer's goal is to maximize the number of supported radio functions with a minimum amount of hardware. Hardware paging presents an elegant solution to this problem, by allowing algorithms to be implemented on physical hardware as needed. While DSPs and FPGAs, which are both currently supported by the SCA, provide different levels of reconfigurability and aid in implementing flexible designs, CCMs, the latest in reconfigurable technology, is the only class of processor that can currently provide real-time paging of algorithms on hardware. Since CCMs are not currently considered by the SCA, in this paper we introduce a framework for handling CCM technology within the SCA. To ease this integration, this is performed in a manner similar to the techniques currently employed for other types of processing hardware. The following sections present an overview of the hardware configuration mechanisms in the SCA and demonstrate how CCMs can be integrated into the SCA with minimum modification to the CF.

## 2 THE SOFTWARE COMMUNICATIONS ARCHITECTURE

Shown in Figure 1, the SCA is an object-oriented architecture where the functionality of the radio is implemented through the use of objects. These objects are used to implement the software components of a waveform and to control, manage, and model hardware components. To provide the links to build these objects into a functioning software radio, the SCA defines critical interfaces and services in the core framework. To manage the instantiation and destruction of these objects, to facilitate the communication between the objects, and to allow objects written in different languages to interact, the SCA utilizes the services of the Common Object Request Broker Architecture (CORBA).

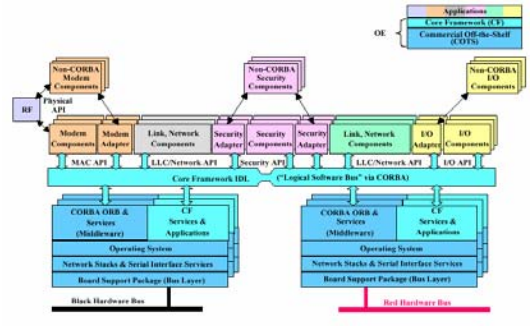


Figure 1 SCA Software Architecture

## 2.1 The Role of CORBA in the SCA

Standardized by the Object Management Group (OMG), CORBA is a software architecture used to enable the communication between objects and to allow objects to transparently use the services of other objects in a distributed processing environment. The structure of CORBA is built around four key components: an object request broker (ORB), object services, common facilities, and application objects [5]. CORBA implements communications on a peer-to-peer basis wherein a client object requests the services of a server object. It is the responsibility of the ORB to abstract the interface of the server object, so the client object need not know the server's location, implementation, execution state, or the communication mechanism used to relay information between client and server [4]. The object services component is used to support the basic functions of object implementations. Examples of objects services include Life Cycle Services used for creating, deleting, copying, and moving objects, and the Naming Service used to retrieve the services of a specific object. A reduced version of the Naming Service is used within the SCA for purposes such as ensuring that Application Factories build the correct components. Application objects are the actual objects that use the services of the ORB. These are identical to the application level objects of the SCA illustrated in Figure 1.

To aid in the implementation and abstraction of the objects in CORBA, the interfaces to all of the application objects are implemented in Interface Description Language (IDL). IDL is an implementation-neutral way of describing the interface of an object that is not specific to any programming language. For implementation purposes, CORBA uses language mappings between an object's native language and IDL in order to handle objects. As shown in Figure 2, objects can then communicate with an ORB through IDL or directly through the ORB interface. Within the SCA, all CF interfaces are written in IDL.

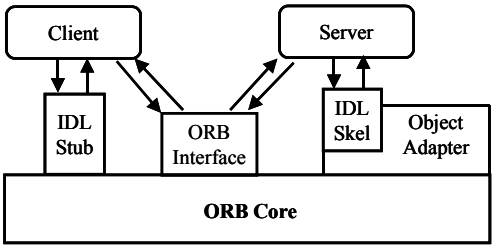


Figure 2 CORBA Messaging

As many domains are not able to support the processing overhead of a full CORBA implementation, the SCA only requires an implementation of minimumCORBA. MinimumCORBA is an attempt to reduce the overhead to an acceptable level while still retaining the most desirable features of CORBA. For instance, in minimumCORBA, all of the dynamic invocation operations are not included. Thus the implementation repository and interface repository that enable client objects to search for objects by type instead of name are greatly reduced. Additionally, the number of ORB interfaces are reduced, and the dynamic skeleton interface, dynamic invocation interface, and dynamic value management are eliminated [5].

## 2.2 Hardware Resource Management in the SCA

To aid in the management of hardware, the SCA introduces a hardware class hierarchy. The hardware class structure hierarchy ranges from the highest level of abstract hardware, such as the *SCA Compliant Hardware* class, to classes dedicated to specific types of hardware components like a *Processor* Class or GPS *Class*. Each hardware class is uniquely defined by its attributes. The interfaces to manage the components, attributes and their corresponding values define the functional requirements of an SCA complaint hardware component.

The SCA hardware class hierarchy is shown in Figure 3. The *Chassis* class defines the attributes for form factor, backplane, interconnection of the hardware modules and similar functions. The *Chassis* class is also responsible for routing data correctly through the software radio. The *Hardware Modules* class contains various subclasses that define specific types of hardware. For instance, the *RF* hardware class contains the software abstraction of all the RF hardware available to the software radio. Similarly, the *I/O* class is responsible for defining the I/O interfaces to the software radio. Generally, each hardware class inherits attributes from a parent class to refine the attributes that are unique to that type of device.

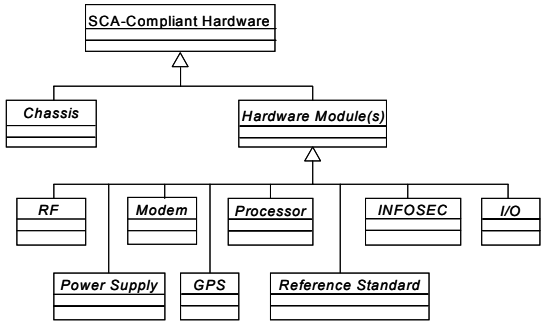

Figure 3: Current SCA Hardware Class Hierarchy

The hardware devices that correspond to these classes will have values for the relevant attributes and will be selected based on a platform's physical requirements. The device attributes that are used in the creation of waveform applications and provided in a Device Profile readable by CF applications.

This hardware class hierarchy is accompanied by a hardware rule set that stipulates certain conditions for a piece of hardware to be considered SCA compliant, such as requiring the device vendor to provide a domain profile for the hardware component. With the proper application of the rule set and with procurement direction, hardware modules that are common to multiple domains can be identified.

To use these hardware devices, the CF specifies the use of special software classes, managers, and interfaces. Two critical software classes are *Resource* and *Device*. A *Resource* is the fundamental component in the SCA and provides certain basic interfaces, such as those required for port and lifecycle operations. An application, itself, is a *Resource* and may consist of multiple *Resources*. A *Device* is a *Resource* that serves as a software abstraction of a physical device and provides interfaces for directly controlling and using the device. The *Device* class is further refined by the *LoadableDevice*, *ExecutableDevice,* and *AggregateDevice* classes that provide interfaces for loading information, executing code, and handling devices. *ModemDevice*, *I/ODevice*, and *SecurityDevice* are examples of *Devices* defined in the SCA. The physical hardware device is represented by a software abstraction that can directly control the physical hardware through the *Device* interface. Actual applications are then implemented on devices through the use of a *DomainManager* and *DeviceManagers*.

A domain manager, interfaced with by a *DomainManager* class, manages the software applications, application factories, hardware devices and device managers within the system. Note that each of these components would also have an associated interface class. The resources being managed by the domain manager are CORBA objects implementing the *Resource* interface. The domain manager is also responsible for allocating the hardware devices to an application based on various factors known by the domain manager, such as the current availability of hardware devices, behavior rules of a resource and the loading requirements of a resource.

In order to use a device, certain capacities (e.g., memory, performance, etc.) must be obtained from the device. To aid in this process device managers are used. As the capacity properties will vary among devices and are described in the device's Device Profile. A device may have multiple allocable capacities, each having its own unique capacity model. It is the responsibility of the device manager to keep track of what capacities are available and what capacities have been consumed by resources.

Figure 4 shows the interaction between various components of the CF and the hardware elements. The interactions shown in the figure focus on the steps needed to program the hardware for an application. When an application includes a device, it means that specific parts of the application have to be implemented on a particular type of device. The domain manager requests the device manager to program devices based on anapplication's request. The domain manager allocates the hardware based on the device profile and information provided by the device managers. When the application does not request a particular hardware device, the domain manager is responsible for allocating the hardware devices for the application.

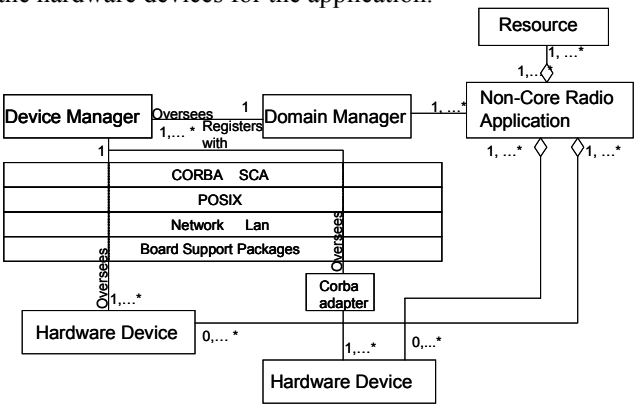

Figure 4 Hardware Resource Management in the SCA

## 2.3 The Role of Adapters in Controlling Hardware in the SCA

As minimumCORBA only supports language mappings of IDL to C++ and Java, legacy code written

in other languages would not be directly supported by the SCA. Additionally, processing elements that do not support the interfaces specified by the CF would be unusable by CORBA enabled Resources. To solve this problem, the SCA uses *Adapters* which are specialized *Resources* or *Devices* that translate the interface of a non-CORBA capable devices or code to a CORBA usable interface. For the CORBA enabled *Resources*, this process is transparent and the *Resources* can continue to use CORBA as a messaging service. The adapter concept is illustrated in Figure 5.

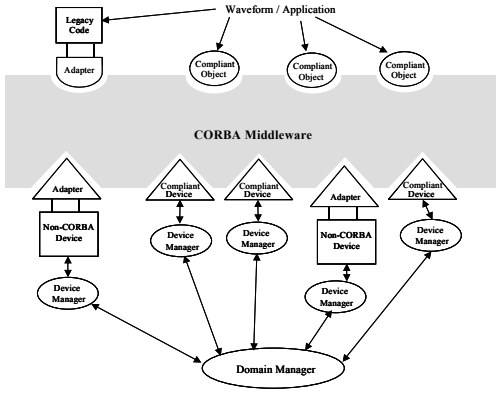


Figure 5 Adapters in the SCA [6]

## 3 CONFIGURABLE COMPUTING MACHINES

The development of Configurable Computing Machines (CCM) allows designers to achieve flexibility in hardware without sacrificing power or silicon efficiency. The CCM concept tries to retain the desirable characteristics of FPGAs and ASICs for a particular application or a suite of applications while achieving a flexibility to rapidly switch between applications. CCMs have static hardware for frequently used cores like multiplication, which result in efficient radio designs. Other features that are typically used to enhance the CCMs could be strategically placed shift registers, circular buffers, large number of I/O pins for good data throughput, etc. These components and the connections between the components are programmable to allow for reconfiguration. Conceptually, CCMs are customizable ASICs or FPGA with a coarser granularity that is better suited for signal processing applications. Some CCMs have a scaleable architecture that enables algorithms to be scaled across multiple CCMs. This extends the capabilities of the CCM architecture and provides for an additional level of flexibility in the use of the CCM hardware.

Virginia Tech has developed a scalable CCM called Stallion that is specifically suited for flexible, high-throughput, low-power computations and is based on stream-based processing [1]. Stallion supports fast run-time reconfigurability, which makes it attractive to soft radio applications. Stream based processing uses a common bus for data as well as programming packets with a header that indicates the nature of the stream packet. Further, the stream packets also indicate the next module where subsequent programming packets should be forwarded. Thus each programming stream is independent and self-steering,. This permits partial reconfiguration as one stream can be reprogrammed as other streams continue processing.

Figure 6 shows the architecture of Stallion. It consists of two meshes of 8x4 functional units and multipliers connected by a crossbar, and a network of flags and busses. Stallion has six dataports that can be dynamically programmed to be either input or output ports. From the dataports, the stream proceeds through the crossbar and then to either mesh of functional units. After being processed in the functional unit, the stream can continue through the mesh, go back through the crossbar to any other part of the chip, or off the chip via a data port.

The output of the functional unit can be routed locally to adjacent functional units or multipliers sent elsewhere on Stallion through the crossbar or the skip busses. The skip bus is a special kind of bus that sends data to any other functional unit in a mesh within one clock cycle. Finally, each functional unit is capable of generating and processing an array of flags, which can be used to control and monitor the functional unit's operation.

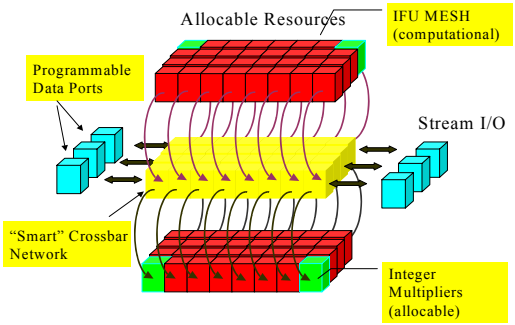


Figure 6 Stallion Architecture

## 4 PROPOSED APPROACH TO INTEGRATING CCMs INTO THE SCA

Integrating CCMs into the SCA requires that the CF include methods for programming, downloading data and interacting with the CCM. Ideally, these operations should be done with a minimal impact on existing SCA components and support the operations described in the previous sections. As a CCM will not in general be CORBA compliant, the first step that must be taken is to

introduce an *Adapter* resource to provide an interface that is usable by the other resources in use in the radio.

The first step to managing the implementations on a CCM is to introduce a hardware class to describe the allocable attributes. As the CCM will be used to perform processing operations, this class will be easiest to include as a child of the *Processor* class. The *Processor* class defines all the digital processing hardware used by the software radio. Traditionally processing hardware has consisted of DSPs, FPGAs and , which are included in the SCA architecture hardware class definition. In this paper, we introduce another class of processing hardware for Configurable Computing Machines (CCM), the *CCM* class. The CCM class definition can be included as another extension to the *Processor* class as shown in Figure 7.

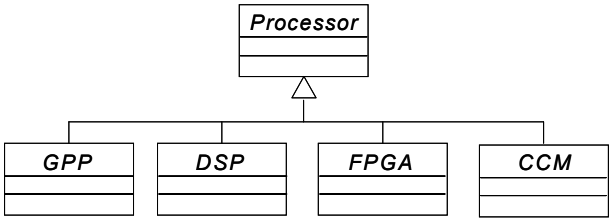


Figure 7: Suggested Processor class definition

Each of a CCM's processing cores can be considered an executable device, thus the *Device* created for the CCM should be an *AggregateDevice* which is an aggregation of *ExecutableDevices*. It would be the responsibility of the adapter to ensure that the information intended for a specific device within the CCM arrives at the device. By treating the CCM as an aggregate device, partial reconfigurability is extended further and more easily supported within the SCA. Each vendor supplying a CCM would also be required to provide a Device Configuration Profile for that CCM. This profile should contain the type and number of rocessing cores that are available on the CCM, the type and number of I/O ports, the amount of available memory, and whether or not the CCM is scaleable. For applications that indicate operation on a CCM, those applications should also indicate the amount of resources that application consumes with respect to the aggregated devices on the CCM. A device manager would then be instantiated for the CCM according to the *DeviceManager* class interface. This device manager would specifically be responsible for monitoring the capacity of the CCM and apprising the domain manager. In the following section, we present an example of inserting a CCM into a SCA compliant radio.

### 4.1 Example Implementation

In this Section, we present an overview of the process to include the Stallion CCM in a SCA compliant radio. A similar procedure can be followed for other CCMs. First an Adapter would need to be created to map the messages received from the device manager into the streaming format that Stallion uses. This adapter would also be responsible for ensuring that individual functional units are programmed as specified by the device manager. The Layered Radio Architecture developed at Virginia Tech [7] could also server this purpose where the tasks of the adapter are separated into the Soft Radio Interface Layer and the Configuration Layer. Next a Device Configuration Profile would have to be constructed for Stallion and should appear as shown below in

**Figure 8**.

```
- <xs:element name="Device">
  - <xs:complexType>
    - <xs:sequence>
        <xs:element name="numDataPorts" type="xs:int" />
        <xs:element name="numUnits" type="xs:int" />
        <xs:element name="numMultipliers" type="xs:int" />
        <xs:element name="numMemUnits" type="xs:int" />
      </xs:sequence>
      <xs:attribute name="isScalable" type="xs:boolean" />
  </xs:complexType>
</xs:element>
```

Figure 8 Partial Listing of Stallion Device Configuration Profile

A device manager class for Stallion should also be created that inherits the *Device* interface. A class diagram for this class is shown in

**Figure 9**. Notice that the Stallion Device Manager contains the Stallion configuration as well as a list of already registered services that are available. The Stallion Device Manager is also responsible for

maintaining the current status of Stallion and available capacity for use by the domain manager. The device manager also responds to requests from the domain manager to implement algorithms on Stallion through the Stallion Adapter.

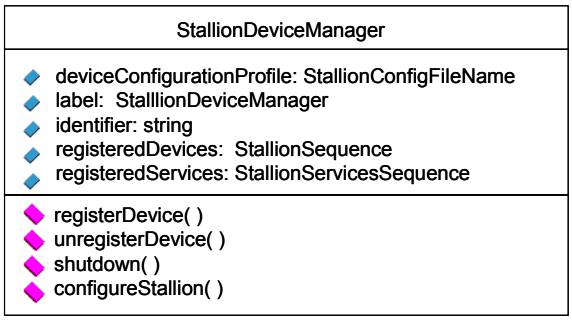


Figure 9: Device Manager UML for Stallion Device Manager

Figure 10 and

Figure 11: Stallion CCM configuration scenario

depict the sequence of steps needed to initialize and configure Stallion. Upon startup, the Stallion Device Manager is responsible for initializing and registering each of the Stallion devices in its profile. The Device Manager interacts solely with the Stallion Adapter so that the actual interactions with Stallion are transparent to the remaining components in the SCA. Once the Stallion Device Manager successfully launches a Stallion Device and registers its services, the Device Manager registers the CCMs with the Domain Manager. The Domain Manager uses this information when an application requests the use of a Stallion Device. The Domain Manager queries the Stallion Device Managers to obtain an updated list of available devices and allocates devices to the application based on the CCM's capacity and registered services.

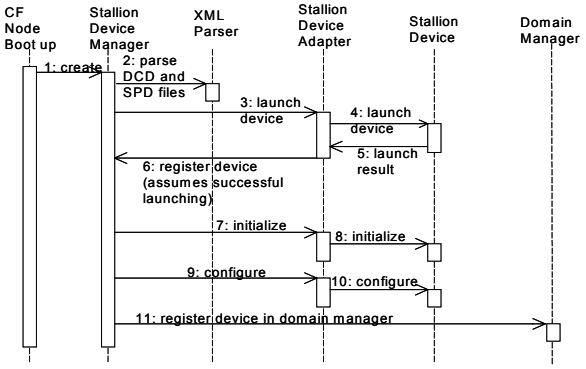


Figure 10: Stallion CCM startup scenario

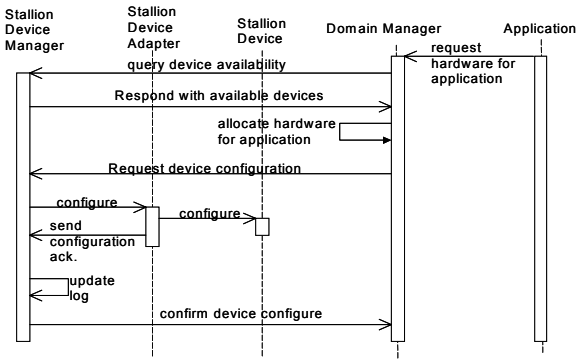


Figure 11: Stallion CCM configuration scenario

## 5 RECOMMENDATIONS AND CONCLUSIONS

CCMs provide an added dimension of flexibility by enabling hardware paging and partial reconfiguration. They represent an important class of processing hardware and are poised to become a major component of software radios. Their inclusion in the SCA is necessary for exploiting the latest hardware developments for software radios. In this paper we have described methodologies for including CCMs in the SCA with minimum modification to the existing standard. The primary steps for including CCMs in the SCA hardware are summarized as follows:

1. Define a new class called the *CCM* class as an extension of the *Processor* class.
2. Define attributes in the device manager for the CCMs used. Include these attributes in the CCM's Device Configuration Profile that is shared with the domain manager.
3. Develop a CCM interface class that satisfies the *AggregateDevice* interface.
4. Develop an Adapter for each CCM to be included.

Stallion, the CCM developed at Virginia Tech is used as a candidate CCM to demonstrate the integration of CCMs into the SCA. However, the steps described here are applicable to other CCMs and can be implemented along the same lines.

## 6 REFERENCES

[1] Software Communications Architecture Version 2.2.
[2] J. Mitola, *Software Radio Architecture: Object-Oriented Approaches to Wireless Systems Engineering*, John Wiley, 2000.

[3]Object Management Group Document: formal/02-06-33: "The Common Object Request Broker: Architecture and Specification," Version 3.0. **www.omg.org**
[4] S. Vinoski, "CORBA: Integrating Diverse Applications within Distributed Heterogeneous Environments," *IEEE Communications Magazine*, Vol. 35 Issue: 2, Feb. 1997. Page(s): 46 -55.
[5] Object Management Group Document formal/02-08-01: "Minimum CORBA Specification" **www.omg.org**.
[6] J. Reed, *Software Radio: A Modern Approach to Radio Engineering.* Prentice Hall, 2002.
[7] S. Srikanteswara, "Design and Implementation of a Soft Radio Architecture for Reconfigurable Platforms," Ph.D. Dissertation 2000 Virginia Tech.