INTERCONNECTING SCA APPLICATIONS

François Lévesque (francois.levesque@crc.ca), Steve Bernier (steve.bernier@crc.ca)
Communications Research Centre, Ottawa, ON, Canada
Tel/Fax: 613-998-2858/613-990-0316

ABSTRACT

The Software Communication Architecture (SCA) is a component-based development (CBD) framework for embedded systems. An important aspect of CBD is the reusability of components. Components offering different levels of services can be used in multiple instances and be interconnected through ports to build assemblies. SCA applications could also be considered as components having a set of ports and therefore be aggregated to create larger applications. However, the SCA specification, in its present state, does not specify how applications can be interconnected. This may create severe limitations for radio networking, especially when interconnectivity between different communications protocols is required (e.g. FM to P25). This paper presents a simple way of interconnecting applications and thus enabling application reusability. The proposal is an extension to the SCA specification but does not requires any new XML profile, which guarantees backward compatibility with existing tools and Core Frameworks.

1. INTRODUCTION

The SCA is a framework that enables component-based development (CBD). The Software Defined Radio (SDR) industry is pioneering the use of CBD for embedded systems and so far, the SCA is unrivalled as a CBD environment for those embedded platforms. Nevertheless, several years of usage have revealed that the SCA Core Framework specification [1] lacks support for at least one important feature: inter-application connections. The lack of standardization for this type of connection has lead to the implementation of a number of proprietary solutions, which is contrary to the SCA philosophy.

This paper describes why inter-application connections are important and how they can be implemented with a high-level of backward compatibility. Section 2 proposes that reusability should not only be a property of components but should also apply to applications. It also draws a link between application reusability and inter-application connections. Section 3 explains how the concept of an aggregate application [2] can be used to add support for inter-application connections. Section 4 discusses how the support for aggregate applications may impact the implementation of a SCAv2.2.x Core Framework.

2. APPLICATION REUSABILITY

Reusability is the corner stone of CBD. This fairly recent programming paradigm elevates reusability one step above source code. It provides reusability at the executable level (e.g. the component level). Furthermore, the SCA provides an additional degree of freedom in terms of reusability thanks to CORBA. It supports interoperability between components developed for different operating environments (i.e. combination of operating system and processor).

An SCA application can be defined as an assembly of components (called *Resources*) and the specification provides the mechanism to reuse these resources in multiple applications. It is understood that the potential for reusability of a component is directly related to the level of granularity of the service it provides. Components offering a service of a finer granularity will be more reusable. For example, a component executing forward error correction (FEC) functions will find itself reused much more often than a component executing a complete modem function. However, modeling at a small level of granularity yields more components, which can lead to excessive overhead.

Nevertheless, it is possible to implement a specific functionality as one or many *Resources*. However, the SCA currently only provides one mechanism to group *Resources*; i.e. applications. In other words, an SCA application cannot be made of an assembly of sub-applications, it can only be made of one or many *Resources*.

SCA application := *Resource*$^+$

This limitation of the SCA can have a negative impact on portability of components at a fine level of granularity. In the absence of a way to create a sub-assembly from a number of smaller components, developers implement larger components. At the other end of the granularity spectrum, this limitation also prevents developer from using existing applications to create an aggregate application. For instance, it is impossible to define a cross-banding application that would be made of an AM Receiver application connected to an FM Transmitter application, and of an FM Receiver application connected to an AM Transmitter application. The only way to model this is to create a new application from the union of all the *Resource* components of the four applications.

There are a number of drawbacks associated with the creation of a large application composed of the amalgamation of *Resources* from smaller applications. First, the assembly knowledge already embedded into the pre-existing applications must be properly duplicated. This means developers must redo every connection between components, redefine each overriding property value, recreate all the uses-device relationships, and more. This task is tedious and very error prone. Second, the developer is forced to create a new assembly controller for the new extended application. This step involves duplicating the assembly control logic, which may become very difficult to do especially when the source code is not available for the assembly controllers of the applications being combined.

Fundamentally speaking, the SCA makes it very difficult to create aggregate applications because of its one-level definition of application concept. It simply does not provide the same level of reusability for applications as it does for components. Ironically, the SCA already supports inter-application connections through the concept of external ports. This does allow deployed applications to be interconnected but a Core Framework cannot establish those connections since there is no means to specify a list of inter-application connections. Inter-application connection can only be supported through proprietary extensions to the SCA specification.

3. AGGREGATE APPLICATION CONCEPT

For an SCA Core Framework to perform inter-application connections, it must be able to coordinate the launch of the applications that need inter-connections. In other words, there must be a standard means to describe an aggregate application. From a high-level point of view, the concept of an aggregate application requires a slightly different definition of the current application concept. It requires a recursive definition.

SCA application := (*Resource* | SCA application)+

This kind of recursive definition of an application is already supported in other CBD architecture such as EJB [3], the CORBA Component Model (CCM) [4] and Deployment and Configuration (D+C) [5]. One important property of this new definition is that it can be made equivalent to the original definition. Indeed, by flattening the recursion, an aggregate application ends-up being composed of *Resources* only. The key difference with the new definition is the extra knowledge it provides to describe which *Resources* are part of sub-assemblies and how they must be deployed. Thanks to this property, existing Core Frameworks can easily support aggregate applications since they are capable of deploying a group of *Resources*.

4. IMPLEMENTATION OPTIONS

There are two main approaches for implementing the concept of aggregate applications. The first one actually does not require any changes to existing Core Frameworks as it is completely supported through a modeling tool. The second approach requires modifications to Core Frameworks but makes the aggregate application concept much more elegant and versatile.

The modeling tool approach relies on the fact that the aggregate concept is defined recursively and can be made equivalent to the original SCA application definition as explained in the previous section. In this approach, a modeling tool provides a means to create aggregate applications and to convert them into a single application definition before deployment. This way, a Core Framework only ever sees an application made of *Resources* only. However, the down side to this approach is that once the aggregate application is converted, the aggregate knowledge is lost. Thus a Core Framework cannot be precise in determining which sub application might cause a deployment error. Also, monitoring tools cannot represent the application as an aggregation, which could also be useful. In addition, there are currently no standards in the SCA modeling tools industry, making it impossible to share aggregate applications across different modeling tools. Of course, standardizing the meta-data describing an aggregate application could solve the sharing problem, but it would not necessarily solve the debugging issues or the runtime monitoring tools issues. In fact, once the aggregate application meta-data is standardized, Core Frameworks can support the concept, which helps address the drawbacks of the tool based approach.

The Core Framework based approach allows developers to exploit the aggregate application concept at all levels: modeling, deploying, monitoring and debugging. This approach however requires changes to the behavior of current Core Frameworks. But, as described in this paper, those changes are not significant and can easily be made optional for Core Frameworks implementers that do not wish to support aggregate applications.

5. CORE FRAMEWORK SUPPORT FOR AGGREGATE APPLICATIONS

Currently, SCA applications are described using meta-data that can be generated by modeling tools. The file format for application meta-data is the SAD XML file [6]. The structure of the SAD file mainly allows a developer to identify which *Resource* component(s) the application requires, how many instantiation of each component must be created and how they must be interconnected.

One of the easiest ways to add support for aggregate applications is to allow the SAD file to reference other applications (i.e. sub-applications) in addition to *Resources*. Alternatively, a new XML file could be created to specifically describe aggregate applications. However, during our evaluation it became obvious that this new file would have the same structure as the SAD file. Therefore, this paper concentrates on the option that relies on a SAD file. The reminder of this section describes the impacts of supporting aggregate applications.

5.1 Impacts on the SAD file

To support aggregate applications, the SAD file must first be modified so it can contain references to other SAD files. This modification concerns the *componentfile* section of the SAD used to identify the components that are part of the application. Currently, the SCA specifies references can only be made to components (i.e. SPD files). Allowing references to an application (i.e. SAD files) does not require a structural change to the file format. It simply requires a change in the text of the specification. The main impact of this change is that Core Frameworks will have to deal with references to applications.

The *componentinstantiation* section of the SAD file is where a developer specifies how many instantiation of each component must be created. This currently applies to *Resource* components only and therefore needs to be extended to applications as well. Once again, no structural change is required to the SAD file format; only changes to the text of the specification. The same is true for the sub-elements of *componentinstantiation*. The *usagename*, the *componentproperties*, and the *findcomponent* normally apply to *Resource* instance, but will be used for an application instance. The *usagename* will be used to define the name of the *Application* instance. The *componentproperties* will be used to override the default value for the application properties. And finally, the *findcomponent* will optionally be used to define the name the application will be registered with in the naming service.

When a Core Framework encounters a reference to a sub-application, meaning a *componentinstantiation* for an application, it must instantiate an *ApplicationFactory* and use it to instantiate the sub-application. As a result, the Core Framework will obtain a reference to the *Application* object that is a proxy to the sub-application. That reference needs to be stored as it will be needed for connections and for shutdown.

Connections with sub-applications must be established through *Application* objects to preserve encapsulation of the sub-applications. The *Application* object is a proxy to an instantiated application. However, because of this, it is not possible to use all five types of connections the SCA supports [2]. In fact, this is already the case for connections between components of a node assembly. The following two types of connections cannot be supported by a Core Framework: "deviceusedbythiscomponentref" and "devicethatloadedthiscomponentref". The reason being that an *Application* object can't have "usesdevice" relationships and can't be loaded on a SCA *Device*. The domainfinder type of connection could be supported but it would require a structural change to the SAD file. The *domainfinder* element would It require a new type named "application". Finally, the componentinstantiationref and findby type of connection can easily be supported the same way they are for regular components.

5.2 Impacts on the '*Application*' object

The *Application* object is created when an application is launched. It is created by an *ApplicationFactory* and acts as a single point of entry for interactions with an application. In short, it is proxy to the application component assembly. The *Application* object also is used to provide deployment information. It does so by through six attributes. To get access to the sub-applications of an aggregate application, the *Application* object must provide a new attribute. This new attribute provides a sequence of references to *Application* objects. This way, any runtime monitoring tool can recursively explore the *Application* objects to tally all the components.

5.3 Impacts on the Domain Manager

The *DomainManager* also provides deployment information regarding instantiated applications. This information is provided through a sequence containing a reference to each *Application* object created. In other words, it contains one entry for each instantiated application. With the current SCA specification, the *Application* objects contained in that sequence are completely independent from one another. They can be created or released with no effect on other applications. This property however changes with the support of aggregate applications. Releasing a sub application would definitely have effect on the aggregate application it belongs to.

For this reason, we don't recommend adding the sub-application to the DomainManager's sequence of applications. Only the Application object for regular standalone application or for an aggregate application should be listed. Runtime monitoring tools can still expose the sub-applications through the new *Application* object attribute.

The *DomainManager's* provides another attribute that also might have been affected by the concept of aggregate application. The attribute is a sequence containing one *ApplicationFactory* for each application that has been installed. The deployment of an aggregate application leads to the creation of one *ApplicationFactory* for each sub-application. But again, adding each sub *ApplicationFactory* to the *DomainManager's* list would only add confusion and create problems if one of those sub-factories was released before the *ApplicationFactory* of the aggregate application.

The only minor impact on the DomainManager is related to application installation. Since the new application meta-data (i.e. SAD file) may reference other applications, the installation service must be modified to extend the validation to the sub-application meta-data.

6. CONCLUSION

This paper explained why application reusability is important and presented an approach to interconnect SCA-based applications. Modifications required to the SCA specification to support the aggregate application concept from a SAD file were described. Most of the changes are simply textual. No change to the SAD file is required but to connect applications using a new *domainfinder* 'application' type. However, for backward compatibility purpose, it has been suggested to make the support of aggregate applications optional to the SCA and to define related requirements in a new SCA extension document. As for SCA Core Frameworks, tools wishing to support this SCA extension would have to be slightly modified.

7. REFERENCES

[1] *Software Communications Architecture Specification*, Version 2.2.2, 15 May 2006.

[2] F. Lévesque, C. Auger, S. Bernier, and H. Latour, "JTRS SCA: CONNECTING SOFTWARE COMPONENTS," *Proceedings of the 2003 Software Defined Radio Technical Conference*, Vol. 1, pp. SW2-001, 18 November 2003.

[3] JSR-220 *Enterprise JavaBeans™*, Version 3.0, 8 May 2006.

[4] *CORBA Component Model Specification, Version 4.0, formal/06-04-01.*

[5] *Deployment and Configuration of Component-based Distributed Application Specification, Version 4.0, formal/06-04-02.*

[6] *Software Communications Architecture Specification Appendix D: Domain Profile*, Version 2.2.2, 15 May 2006