WAVEFORM PORTABILITY AND REUSE ACROSS OPERATING ENVIRONMENTS: AN EXPERIENCE REPORT

Jerry Bickle (Chief Scientist SDR Products of PrismTech Corporation
6511 Constitution Avenue, Fort Wayne, IN 46804; jerry.bickle@prismtech.com)

ABSTRACT

In the past, there have been many papers discussing the proper separation or isolation of waveform implementations across varying SDR Operating Environments (OEs) such as the Software Communications Architecture (SCA) [1,2] OE (Core Framework (CF) Common Object Request Broker Architecture (CORBA®), and Portable Operating System Interface (POSIX®) Application Environment Profile (AEP)). Many of these papers also discuss the reuse and portability benefits reaped by such approaches. This paper presents an experience report on the application of such techniques to provide portability and re-use across disparate physical OE platforms.

This paper describes how an SCA waveform model can be used to produce an SCA-compliant waveform implementation and then describes how the same waveform source code can be compiled and deployed across multiple processing environments with differing operating systems and middleware implementations. The result of this demonstration illustrates the reality that the source code is Real-Time Operating System (RTOS) and Object Request Broker (ORB) neutral.

This paper also describes how the same waveform model can also be used to create an Object Management Group™ (OMG™) SWRadio [3] waveform implementation and how the same waveform logic (e.g., algorithms, coders, etc.) can be used across SCA and OMG SWRadio compliant waveform implementation.

1. INTRODUCTION

In the SCA specification, the basic building block of a waveform application is the Resource component as shown in
Figure 1. A Resource component realizes or supports the Core Framework Resource (CF::Resource) interface. This interface provides the generic management operations for configuration, testing, lifecycle management, connectivity and component control. A Resource component in addition to supporting the CF::Resource interface can support additional interfaces (functionality) by offering services via its “provides” ports and can use services offered by other components via its “uses” ports.

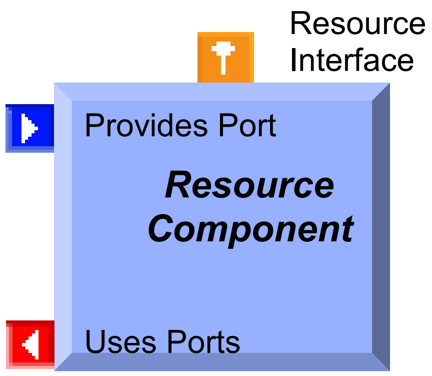


Figure 1. Resource Component Illustration

These ports allow Resource components to be connected into assemblies of components that provide waveform functionality.

The implementation of a Resource component, as shown in Figure 2, can be viewed as having three distinct logical partitions: a Component Container, an SCA Component Infrastructure, and a Component Implementation. A Resource Component Implementation is constrained not to violate the SCA’s AEP as such, the Resource component implementation remains portable amongst operating systems that support the POSIX-based AEP.

The Component Container for the SCA offers an “entry point” (main program or function) that handles:

- The entry point arguments,
- The middleware setup (ORB initialization, ORB Portable Object Adapter (POA)),
- The creation and activation of a Resource component,
- The binding of the Resource component object reference to a CORBA Naming Service, and
- A blocking function that waits on operating system signals to terminate the entry point.

The SCA Component Infrastructure itself can deal with all SCA requirements and CORBA mechanisms associated with the CF Resource interface:

- PropertySet – configure and query properties
- LifeCycle – initialize and release object
- TestableObject – executing specific test properties
- PortSupplier – get Uses and Provides Ports.

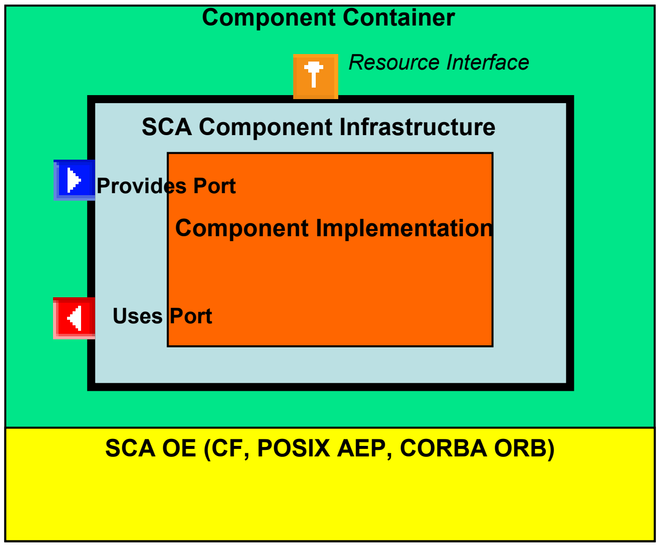


Figure 2. Component's Implementation Logical Partitions

The SCA Component Infrastructure can also deal with all the SCA requirements and CORBA mechanisms for the CF::Port interface that each "uses" port implements. The CF::Port interface provides the interfaces required to connect and disconnect "provides" interfaces.

The Component Implementation contains the implementation of the component (its business logic such as encode, decode, modulate, demodulate, filter, etc.). The Component Implementation processes information coming in from its "provides" port interfaces and after processing the information can send the transformed information out a "uses" port to another component for further processing.

The following sections describe the practicality of the reuse and portability of the Component Container, SCA Component Infrastructure, and Component Implementation across SCA/SDR OEs.

2. SOFTWARE REUSE

Software reuse is the use of previously existing software artifacts to build new artifacts software. Software that is reusable typical has the following traits: modularity, loose coupling, high cohesion, information hiding, and separation of concerns. These traits allow the software artifact to be isolated and repackaged in future software development activities. Component Implementations must strive to achieve these features if they are expected to be reused in future waveform component assemblies. Using modern Model Driven Development™ (MDD™) tools it is possible to generate a Platform Specific Model (PSM) from a Resource Component Platform Independent Model (PIM) that binds to needed technologies such as the C or C++ language [4]. The PSMs produced from a MDD tool will vary and some of the outcomes are as follows:

- A Resource Component Implementation is coupled and unique to an OE as depicted in Figure 3 by being coupled to CF and/or ORB unique features, therefore providing no reuse. A Resource Component Implementation is tied to CF implementation by using features other than standard CF files or usage of non-standard ORB features (e.g., Vendor's exception macros).

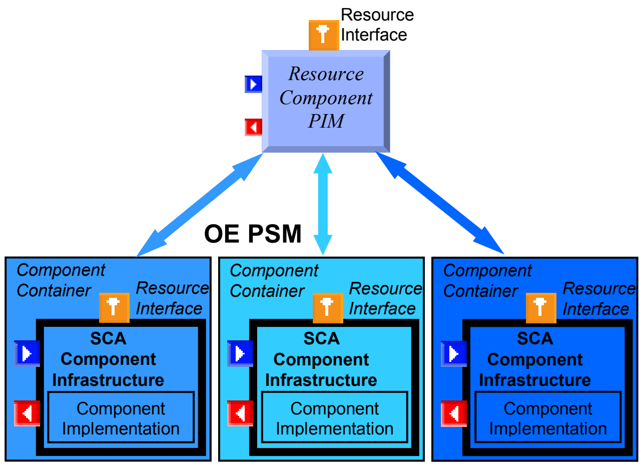


Figure 3. Component to OE PSM Illustration

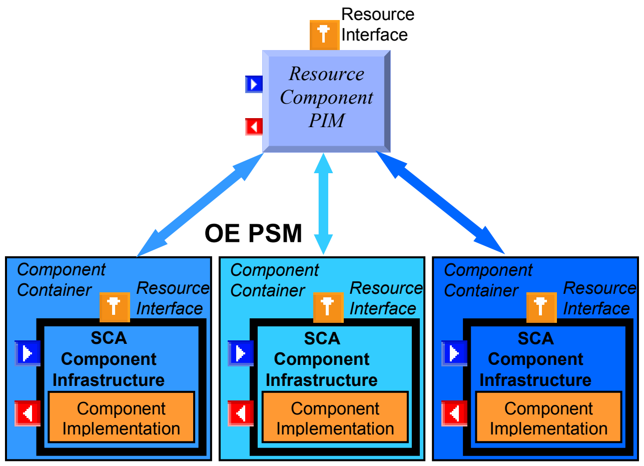
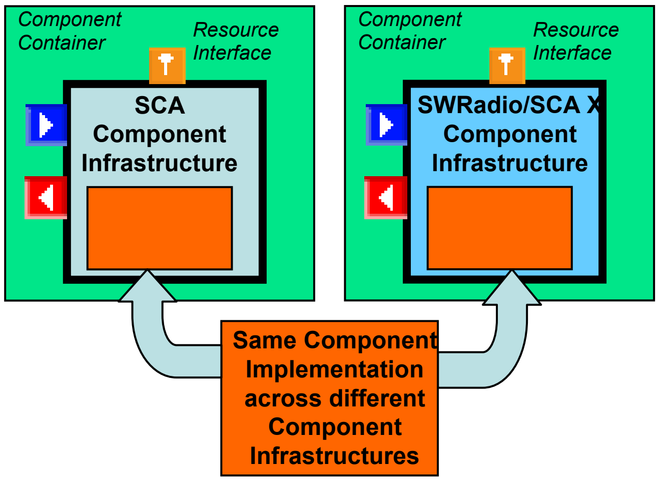


Figure 4. Reusable Component Implementation Across OEs Illustration

- A Component Implementation is decoupled from the OE and SCA Component Infrastructure as shown Figure 4 above. The SCA Component Infrastructure and Component Container in this case are specific to the OE (e.g., ORB, CF). By having the Component Implementation separate from an OE, this allows the Component Implementation to seamlessly evolve with the SCA CF or OMG SWRadio CF as shown in Figure 5. The benefit here is the developer only has to understand the design of one Resource Component PSM that is applicable across compatible OEs.



Component Container
Resource Interface
SCA Component Infrastructure
Component Container
Resource Interface
SWRadio/SCA X Component Infrastructure
Same Component Implementation across different Component Infrastructures

Figure 5. Component Implementation across Component Frameworks.

Other considerations for a component's reusability are:

- Adherence to Language Standards (C/C++/HDL) and elimination of Non-Portable C/C++ Language Features
- The processor, language, compiler and ORB commands and flags used to compile and link the component implementation for a specific processing environment.

3. SOFTWARE PORTABILITY

Software Portability is the ease with which one system or component can be transferred from one hardware or software environment to another [4]. In the case of an SCA-based SDR, porting may refer to the porting of a waveform application onto an SCA/SDR platform or the porting of a Resource Component onto another SCA/SDR OE. The goal of some of the SCA/SDR standards [1,2,3] are to provide open system specifications. An open system [6] is a system that implements sufficient open specifications for interfaces, services and supporting formats to enable properly engineered applications software:

- to be ported across a wide range of systems (with minimal changes)
- to interoperate with other applications on local or remote systems
- to interact with users in a style which facilitates user portability .

The goal of application or component portability is to minimize the cost and effort that are known and economically reasonable to port an application or component onto another platform. Ideally one would simply prefer to recompile an application for a new platform but this is usually not the case for various reasons (e.g. the referenced architecture has different hardware processing elements). As shown in Figure 6 the same component PSM can be portable across multiple OEs. Portability does not require significant re-engineering of the application. There are several dimensions to application portability [6]:

- Program Portability: Will the code run successfully on all intended platforms?
- Data Portability: Are the data structures or files used in the application portable or available on all platforms?

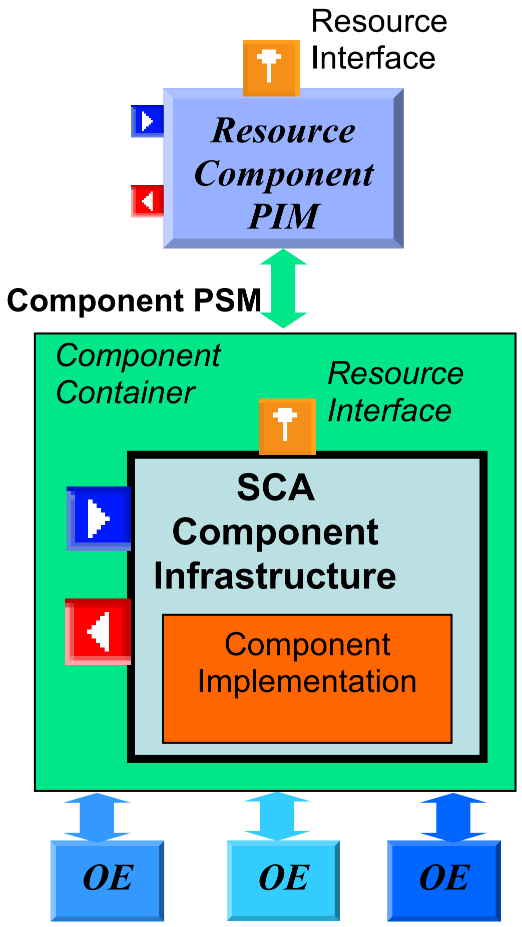


Figure 6. Portable Component PSM

- End-User Portability: Since re-training of users can be a relatively expensive exercise, it is often required that an application will need to have the same look and feel across several platforms.
- Developer Portability: The use of a standard set of interfaces and services across the entire target platforms minimizes the re-training of developers.
- Documentation Portability: Users of applications on different platforms often have different expectations regarding the type of documentation they receive especially in the context of on-line help facilities. Here we will mainly concern ourselves with the first of the above considerations. However, there will frequently be significant overlap with the other aspects.

3.1 Program Portability

Program Portability for a Resource Component‘s implementation may be impacted at the Component Container level. Potential impacts on the Component Container related to RTOS POSIX and ORB compliancy include:

- The entry point name is usually “main” for POSIX compliant RTOSes but for flat address space RTOSes the entry point may be a different function name. This issue can be handled using compile time directives for specifying the entry point name. This allows the same source code to be used for POSIX and non-POSIX RTOSes.
- Entry Point arguments may not conform to “argc/argv” format without special processing.
- Setting up of the ORB transports plug-ins used (e.g., shared memory) between processors or within a processor.

The Component Infrastructure and Component Implementation should be portable across platforms for the same language (such as C or C++). Other considerations for a component’s portability include (by no means a complete list):

- Adhere to coding language standards and the elimination of non-portable or compiler vendor specific language features. For example, avoid “pragmas”, use of bit-fields, use of native types, etc.
- Avoid CORBA ORB vendor specific functions.
- Strive for commonality between similar languages. For example, strive to make header files compatible with both C and C++ whenever possible.
- Use common operating systems libraries across languages. This makes porting code from more resource constrained environments (DSP) to less resource constrained environments (GPP) feasible (see Figure 7).

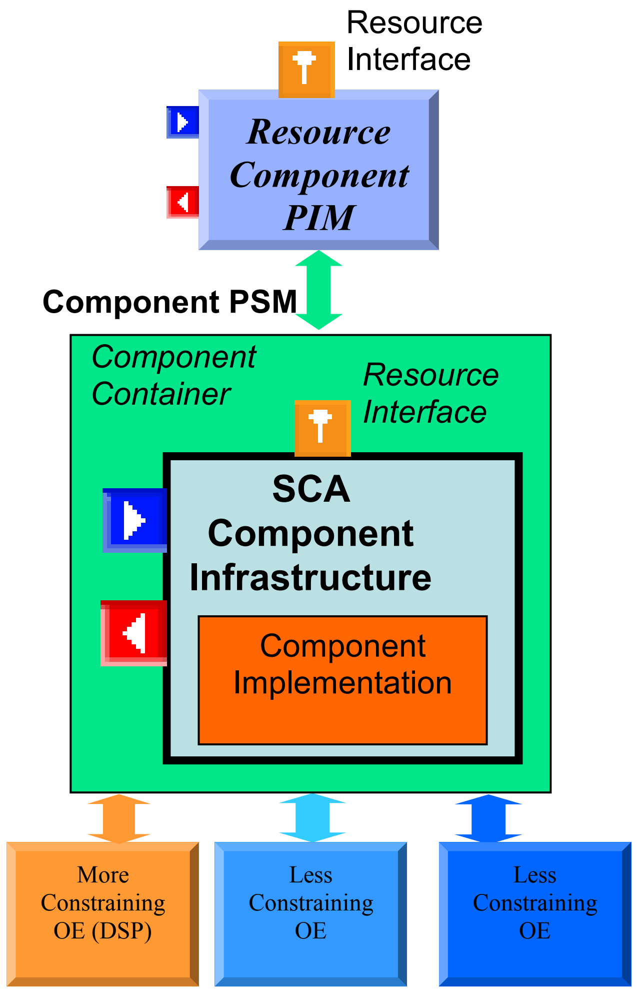


Figure 7. Different Constraining but Compatible OEs Illustration

3.2 Developer Portability

Developer Portability (also known as Platform Portability) has two aspects:

1) Common platform service components that abstract common radio functionality promote waveform application portability. These common services can be standardized and offered across radio platforms developed my differing vendors thereby increasing the reuse and portability of waveform applications.
2) Platform capability that is offered by the OE and the physical communication channels of the radio hardware.

Currently, industry standardized SDR platform service components are severely lacking. Some interfaces and service component definitions defined and offered to industry by the US DoD's Joint Tactical Radio System (JTRS) program [7] while some others are in the process of being standardized by the Software Defined Radio Forum and the Object Management Group (such as smart antenna [8] and digital IF interfaces [9]). Although this handful of common platform service definitions goes a long way to increase waveform application portability additional standardization in this area is sorely needed.

The degree to which a waveform application is portable to a particular radio platform is determined by the evaluation of waveform application PIM against the platform service components offered as well as the mapping of waveform component implementations onto the radio OE and physical radio platform.

In addition to basic software portability, a waveform application must also be able to reproduce its intended behavior on the radio platform to which it is being ported. In order to do this a waveform application must capture as part of its deployment properties:

1) The QoS requirements between the waveform application's components
2) The QoS requirements between the waveform application components and platform service components, and
3) The components processing requirements.

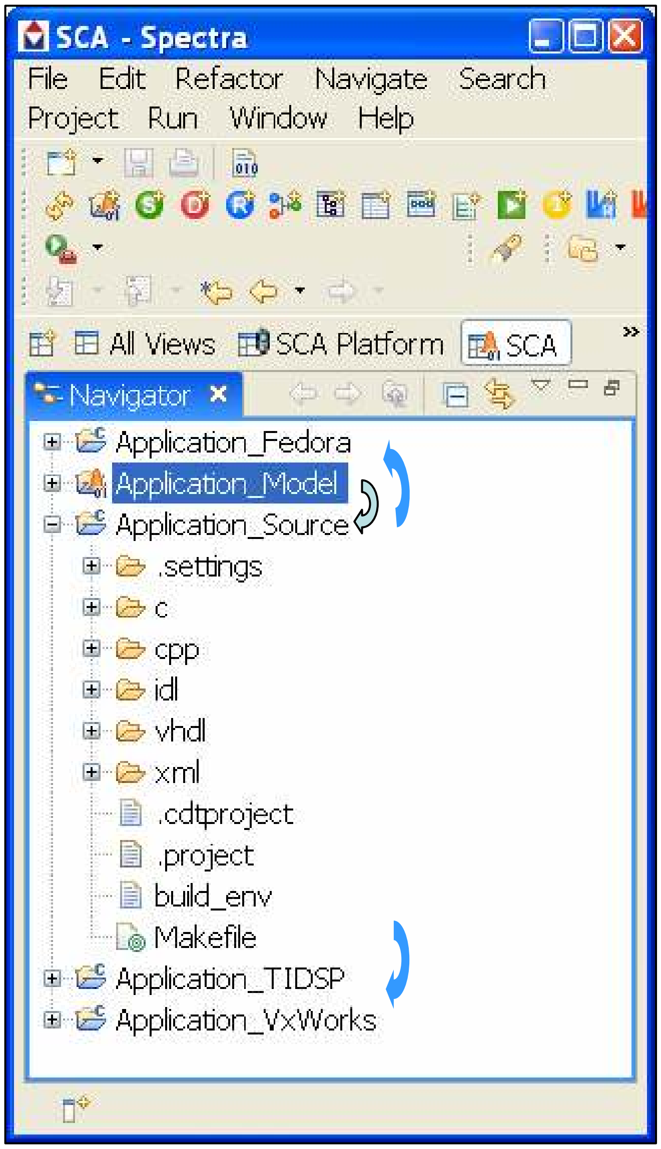
4. EXPERIENCE REPORT

As shown in Figure 6, common Resource Component implementations for Component Container, Component Infrastructure, and Component Implementation can currently be automatically generated using industry standard MDD tools. The PrismTech Spectra SDR Power Tools [10], currently used on numerous SDR programs, is an example of such a MDD tool. Also shown in Figure 7, tools such as PrismTech's Spectra SDR Power Tools provide the ability to generate optimized Resource Component implementations written in C instead of C++. This can provide significant reduction in memory footprint for resource constrained environments (such as those running on DSPs). Figure 8 illustrates these two concepts in PrismTech's Spectra tool where an Application PIM is transformed into multiple languages (C, C++, VHDL). That same source code is then used to create multiple target PSMs (in the case of the illustration Fedora Linux®, VxWorks®, and TI™ DSP).

The PrismTech SDR Power Tools also support seamless Component Implementation integration as shown in Figure 5 (including differing versions of the JTRS SCA or OMG SWRadio Resource Component definitions). Additionally, PrismTech SDR Power Tools apply the same techniques for creating reusable and portable platform service components (Even though the SCA does not dictate reuse and portability for platform service components).

5. CONCLUSION

In summary, with modern MDD techniques and existing tools, such as PrismTech's Spectra Power Tools, it is currently possible to generate one reusable and portable component implementation that is RTOS and ORB neutral. This allows the implementation to be used across a wide range of platform OEs. The only exception to this is the component implementation's Component Container code since this is the only piece of code that may not be totally reusable for different ORBs since the ORB transport plug-ins are ORB-specific at this time.



```
Application_Fedora
Application_Model
Application_Source
    .settings
    c
    cpp
    idl
    vhdl
    xml
    .cdtproject
    .project
    build_env
    Makefile
Application_TIDSP
Application_VxWorks
```

Figure 8. PrismTech Spectra SDR Power Tools Illustration

6. TRADEMARKS

- CORBA®, IIOP™, OMG Interface Definition Language (IDL)™, OMG Systems Modeling Language™, Model Driven Development™, MDD™, OMG™, Object Management Group™, Unified Modeling Language™, UML®, XMI® and the OMG logo® are either registered trademarks or trademarks of Object Management Group, Inc. in the United States and/or other countries.
- Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries
- POSIX® is a registered trademark of IEEE.
- Spectra, Spectra SDR, Spectra Power Tools are are trademarks or registered trademarks of PrismTech in the United Kingdom, United States and/or other countries
- TI™ is a trademark of Texas Instruments Incorporated
- VxWorks® are registered trademarks of Wind River Systems, Inc.

7. REFERENCES

[1] Software Communications Architecture 2.2, http://sca.jpeojtrs.mil/downloads.asp

[2] Software Communications Architecture 2.2.2, http://sca.jpeojtrs.mil/

[3] Platform Independent Model (PIM) & Platform Specific Model (PSM) for Software Radio Components (also referred to as UML Profile for Software Radio) v1.0, OMG formal/2007-03-01, http://www.omg.org/technology/documents/formal/swradio.htm

[4] MDA Presentations and Papers, http://www.omg.org/mda/presentations.htm

[5] Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY: 1990.

[6] ISO/IEC 14252:1996 [IEEE Std 1003.0-1995] Guide to the POSIX® Open Systems Environment

[7] Software Communications Architecture (SCA) and JTRS Application Program Interfaces (APIs), http://sca.jpeojtrs.mil/

[8] PIM and PSM for Smart Antenna RFP, sbc/06-12-10, http://www.omg.org/cgi-bin/doc?sbc/2006-12-10

[9] PIM and PSM for Digital Intermediate Frequency Interface RFP, OMG sbc/04-08-15, http://www.omg.org/cgi-bin/doc?sbc/2004-8-15

[10] PrismTech Spectra SDR Power Tools, http://www.prismtech.com/section-item.asp?id=305&sid=18&sid2=54