NEXT GENERATION SCA OPERATING ENVIRONMENTS

Jerry Bickle
Chief Scientist SDR Products of PrismTech Corp., Burlington, Mass.
jerry.bickle@prismtech.com

ABSTRACT

Joint Tactical Radio System (JTRS) Software Communications Architectures (SCA) implementations have been branded by many as being slow or large because of the underlying use of technologies such as the Common Object Request Broken Architecture (CORBA) and the eXtensible Markup Language (XML). Some of this branding has also occurred because of CORBA's initial usage in enterprise systems using TCP/IP. However today's embedded CORBA middleware, designed and standardized for use in real-time, resource constrained, distributed systems makes the building of small and fast SCA implementations viable across General Purpose Processors (GPPs), Digital Signal Processors (DSPs) and Field Programmable Gate Arrays (FPGAs).

The paper begins with a brief discussion about SCA perceptions and technologies that offset these perceptions. The paper additionally discusses SCA distributive communication approaches: *adapters* along with their short comings and new alternatives which are architecturally consistent and use CORBA throughout the radio set. Finally, the paper discusses the capability of SCA operating environments on DSPs and FPGAs.

1. INTRODUCTION

Past, and even some current, SCA/SDR implementations have decided to artificially limit SCA/SDR component framework implementations to operate only on General Purpose processors (Pentiums, Xscale, PowerPC) and to use adapter technologies on GPPs to interface with non-CORBA DSP and FPGA components (as shown in Figure 1). This however, need not be the case with newer alternative solutions that allow the Operating Environment (OE) to support a larger array of SDR hardware processing elements on GPPs, DSPs, and FPGAs.

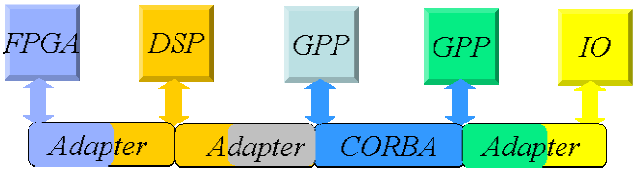

Figure 1. Distributive Communication Approaches

The SCA framework was never intended to be limited to only GPP as many believe. The goal of the architecture (as in many good architectures) has always been to remain implementation technology neutral and to extend beyond the GPP boundary. The goal of SCA is to extend the architecture as close to the antenna as possible to reap the maximum benefits of reuse and portability. Yet, the pace of acceptance within industry has been slowed by certain perceptions about the SCA and its associated implementation technologies. These perceptions include:

- The SCA Operating Environment (CORBA ORB, XML, POSIX) is large and takes up valuable system resources such as memory.
- The CORBA is low performance and adds too much overhead for simple data transfers. This perception is based on TCP/IP being used as the transfer mechanism, which is the default behavior for a CORBA implementation.
- XML parsing is too slow and overkill
- There is no SCA Commercial Off The Shelf (COTS) solution for communicating with waveform components implemented on devices such as DSPs and FPGAs

Other causes of slow acceptance are initial investment in a technology such as in the SCA and the reluctance to use CORBA technology in signal processing solutions. This is a paradigm shift for developers building signal processing software and such paradigm shifts are often shunned by skeptics.

From a historical perspective, these issues are similar to those faced in the transition from low level programming languages such as assembly language to higher level programming languages such as C, C++, and Java. For example, there was much resistance to the C language initially because assembly code is faster and takes less memory space than C. Additionally, C compilers and emulators had many problems associated with them.

Experience with the C language has shown that there are two items needed to make a transition to a new level of design abstraction. The first, and most important, item is the business case. In a competitive market, organizations will strive to implement a new paradigm if there are financial rewards associated with it. In the example of C, the driver was cost savings through portability, reuse and maintainability. Assembly language had speed and size advantages, but it needed to be rewritten every time an application was to be run on a new processor. In contrast C

source code could be written once and then quickly ported to various platforms.

While the business case drives the industry to work towards such paradigm shifts, the new imposed levels of abstraction associated with them are not initially accepted until technological breakthroughs enable practical use. Thus, the second item that is required for abstraction acceptance is the technical enabler. In the case of C, the enablers were breakthroughs in both hardware and software technologies and included such items as

- increases in processor performance,
- increases in memory size and density,
- smaller software footprint,
- more efficient software processing tools (compilers),
- system elements that were bundled together in a single package with no need to make elements from different vendors work together (operating systems to handle low level hardware interface), and
- high level tools to remove complexity from user (emulation and debugging environments).

The combined effect of these enablers allowed the C language to achieve wide industry acceptance and be used in many applications.

Thus, acceptance occurs when the overhead of high-level abstractions no longer has a significant impact on system performance. Assembly is still used today in applications that have strict performance requirements. However, it is known that this comes at the cost of portability needed to support future design modification.

Turning now to the current state of the SCA and CORBA, we find the recent development of similar types of enablers for this field such as:

- Advances in performance, size and density continue to be made for processors, memory and FPGAs.
- Advances in embedded real-time Object Management Group (OMG) CORBA specifications and CORBA profiles for resource constrained system.
- The sizes of CORBA and SCA implementations have been reduced dramatically in recent years.
- CORBA Products for DSPs and FPGAs such as PrismTech's e*ORB for DSP and Integrated Circuit ORB (ICO) for FPGAs and ASICs.
- SCA Operating Environment for GPPs, DSPs and FPGAs such as PrismTech's entire Spectra OE middleware for embedded GPPs and DSPs residing on radio processing platforms once thought too limited in memory and processing power to contain them.
- High level tools to remove complexity from SCA development such as PrismTech Spectra.

The sum total of all the enablers discussed above brings CORBA and the SCA to the edge of a new era of component middleware technologies and development techniques/tools for software radios. These breakthroughs enable practical use of CORBA and SCA for software radios.

2. DISTRIBUTIVE PROCESSOR COMMUNICATION APPROACHES

With the advent of CORBA being used in Software Defined Radio systems such as JTRS, the result is that additional burdens and complexities have been placed on the JTRS platform and waveform developers attempting to handle communications between radio functionality executing on GPPs and that found on Digital Signal Processors (DSPs) and FPGAs. In addition, there have been no standard mechanisms for handling this complexity. As such, techniques have emerged which, in trying to deal with these complexities, actually fail to maintain the architectural consistency that the JTRS Software Communications Architecture (SCA) tries so hard to achieve. To make matters worse, it is this very architectural consistency that drives the portability and re-use of JTRS applications.

There are basically two approaches for SCA component to component distributive communication: Adapter Design Pattern and COTS Middleware such as CORBA.

2. 1 ADAPTERS

The adapter approach is needed where COTS middleware solutions are not available. In the past, this approach was used to support DSP and FPGA components as shown in Figure 1. The Adapter can be at the component level or generic non-component level that can accommodate any component. An example of the component level adapter is a Hardware Abstraction Layer (HAL) as shown in

Figure 2. The HAL approach places the responsible of messaging formatting and processing at the component level at the both CORBA GPP and non-CORBA DSP and FPGA components that are communicating. This behavior as illustrated in

Figure 3 is similar to using IP sockets in the sense the component need to format messages that are to be sent using the HAL interface, and to process and un-format messages received from the HAL interface.

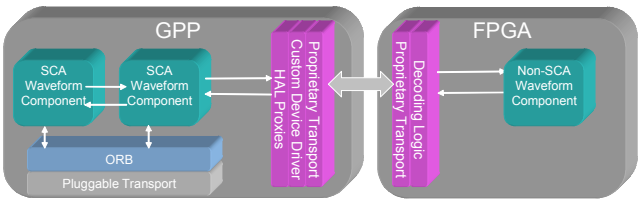

Figure 2. HAL Architecture.

The drawbacks of using the component-level adapter approach are as follows:

- The component is performing low messaging processing that has nothing to do with the component's implementation logic.
- Reuse and portability of this component is limited since it is coupled to a HAL.
- Integration is more time consuming.
- The Client Component is not unaware that the Server Component is a non-CORBA component.
- Not up keeping with the SCA architecture.
- Increase Cost and Schedule for development

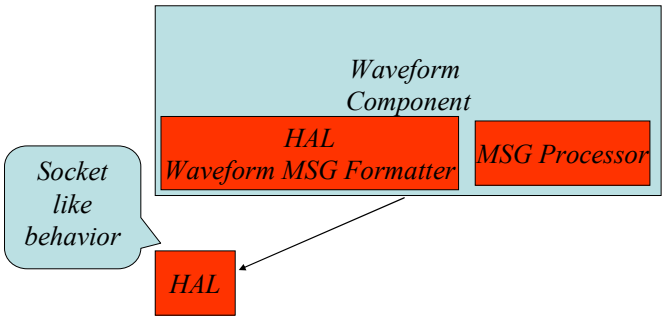

Figure 3. Component's HAL Responsibility

In the SCA, logical device components are used to load and execute software on to a physical device. In the case of the generic non-component level adapter approach, this is usually handled by another component such as an SCA executable device component thereby creating a proxy component that fronts for the non-CORBA component as shown in Figure 4. The proxy component provides some benefits over component-level adapters such as:

- The client component to the proxy component does not need to perform low level messaging processing that has nothing to do with the component's business logic.
- Reuse and portability of the client component since it is compliant with SCA architecture.
- The Client Component is unaware that the Server Component is a non-CORBA component

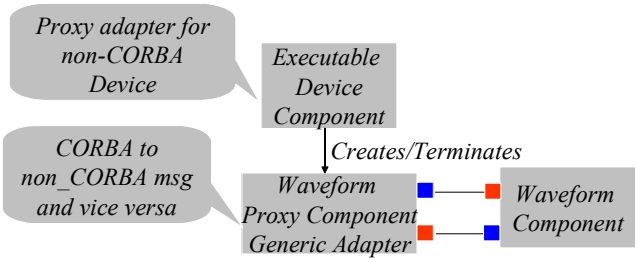

Figure 4. Generic Proxy Illustration

This proxy component usually has the disadvantage of being constrained on its interface capability on what it can support since it has to work for any component connected to it. More or less this type of approach tends to be a "one size fits all" approach.

With either adapter approach, the adapter passes data to the local transport via a driver. The data is transferred over the local transport, such as a system bus, to a transport interface on the non-CORBA processor. The transport interface performs address decode and passes the data to the desired waveform object or function.

Other short comings of both these approaches are:

- Both approaches are based upon non-standard middleware solutions usually proprietary in nature, which results in no COTS tools support to offload the development of SCA components.
- In addition, there is less reuse and portability of the waveform components unless one strictly uses the same hardware architecture on another radio thus making it hard for technology insertion and evolution.
- Lastly, a waveform design is usually captured in a platform specific model and the interfaces between CORBA components and non-CORBA components are hidden in the details. Thus, one is unable to capture the design of the waveform in an independent method such as a SCA Model Driven Development tool.

2. 2 NEXT GENERATION CORBA

Next generation CORBA solutions allow a standard software bus throughout the radio thereby achieving the vision of SCA/SDR. This software bus allows components to seamlessly communicate with one another without knowing what processors they are executing on as shown in Figure 5. The component's communication paths could be on the same processor or not. Additionally, the CORBA Extensible Transport Framework provides for the development of standard and efficient transport mechanisms that support embedded communication. This allows the flexibility to implement other protocols above and beyond TCP/IP (CORBA default) for real-time systems (such as

highly optimized shared memory performance transports with zero copy behavior over RapidIO and cCPI buses).

Technology now exists that provides the realization of the SCA throughout the radio. ORB technology such as PrismTech's OpenFusion e*ORB for GPPs and DSPs and PrismTech's OpenFusion ICO for FPGAs/ASICs. These types of technologies provide greater flexibility in selecting processor architectures for SCA/SDR implementations. A GPP is no longer required since CORBA is available on other processor types.

CORBA ORBs are available for C and C++ implementations, and have been highly optimized for embedded environments such as DSPs. In fact DSP ORBSs have been used to support SDR implementations going back to early in the year 2000 on the Digital Modular Radio (DMR) program.

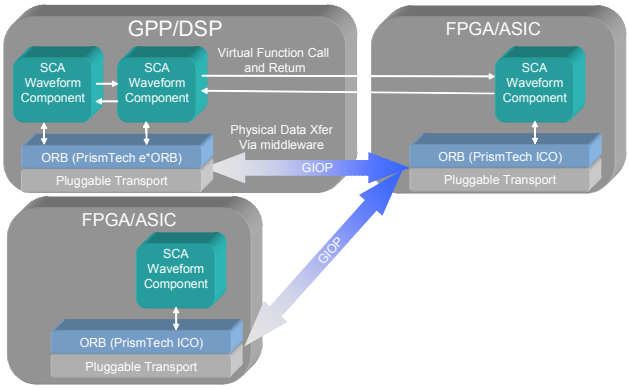

Figure 5. Next Generation CORBA Illustration

Hardware based ORBs are also now emerging (such as PrismTech's Integrated Circuit ORB (ICO)). Hardware elements of a radio system may now be made CORBA compliant and reap the benefits of software portability. In addition to supporting general purpose CORBA communications, the hardware based ORB has also been tightly integrated into PrismTech's Spectra development tool suite for Software Defined Radio (SDR). This brings the portability of the Software Communications Architecture (SCA) onto silicon devices.

Hardware based ORB is a hardware implementation of a CORBA ORB. It supports a general subset of CORBA functions that will support the SCA architecture. While hardware based ORBs may be used to provide SCA compatibility, it is primarily a CORBA core and may also be used in pure CORBA applications with no SCA requirements. For SCA applications, additional functionality may be added via SCA development tools (such as PrismTech Spectra) to implement the SCA component. The hardware based ORBs are written in portable VHDL that can be synthesized onto any FPGA or ASIC platform.

A hardware based ORB design environment usually consists of:

- A hardware based ORB core,
- IDL to VHDL compiler,
- SCA Modeling Tool,
- The optional SCA waveform component.

3. NEXT GENERATION OPERATING ENVIRONMENT

The SCA Operating Environment (OE) as shown in Figure 6 along with the next generation CORBA ORBs provides the added flexibility that allows for SCA Core Frameworks (CFs) to be implemented on DSPs and FPGAs besides or in addition to GPPs as depicted in Figure 7. For example, one could have a complete CF implementation on a DSP or a partial CF implementation such as a SCA executable device component. For a partial configurable FPGA, one may also have a SCA loadable device component.

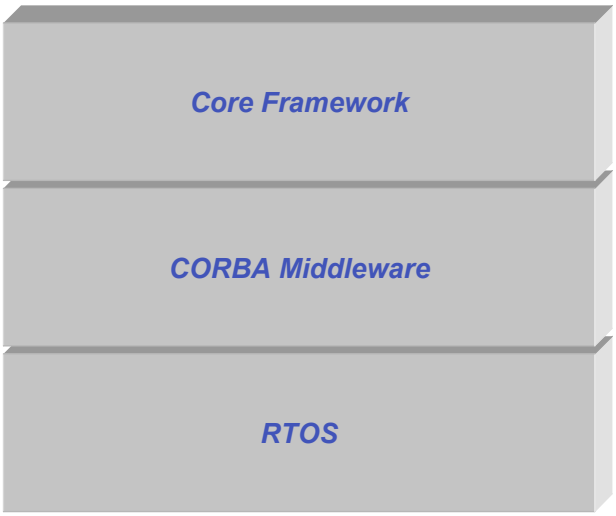

Figure 6. SCA Operating Environment

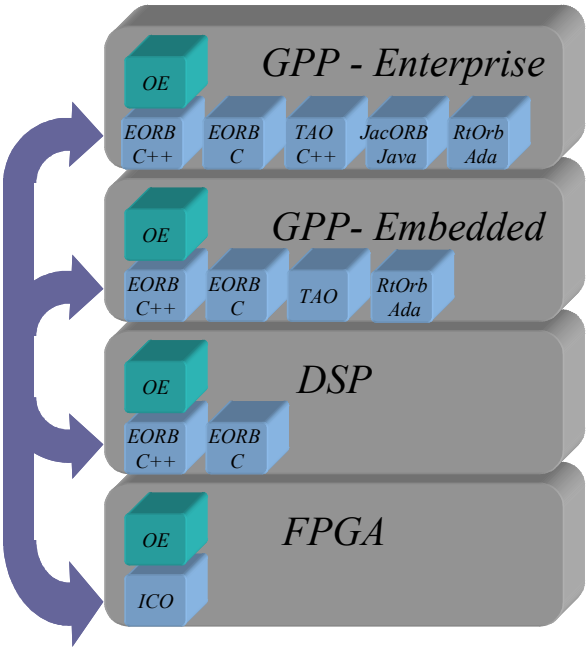

Figure 7. Next Generation OE

4. CONCULSION

The next generation CORBA enablers mentioned above along with SCA development tools like PrismTech Spectra usher in a new era of SCA/SDR development. These breakthroughs allow waveform and platform developers to concentrate on their radio implementations as they did before the inception of SDRs, thus allowing developers to concentrate on waveform business logic instead of building their own software middleware solutions.

The perceived shortcomings of CORBA have been over come by:

- small and fast CORBA implementations to efficiently support communication across the entire signal processing chain, including FPGA and DSP environments. Products such as PrismTech's e*ORB require less than 90KB.
- Efficient CORBA implementations, such as e*ORB, impose little overhead on top of the underlying performance of the transport.
- Choice of transports in ORB is critical to meeting performance criteria. ETF allows for custom transports to be easily supported and allows for multiple transports to be configured in and used in the same system.

The language neutrality of CORBA allows SCA OE to be written in C (very low footprint) but still support waveforms written in other languages such as C++ and Ada. The sizes of SCA OE implementations have decrease dramatically in recent years. These ultra compact SCA OE implementations are now available and running on embedded GPPs and DSPs residing on radio processing platforms once thought too limited in memory and processing power to contain them.

If CORBA is not used throughout the radio then one is on the road to a *poor man's* middleware implementation. One still has to solve the same issues in a proprietary manner. Transports, message formats, marshalling/demarshalling of types, and call dispatch all still needs to be addressed. Finally, the standardized benefits of CORBA are substantial:

- CORBA facilitates implementation of portable waveforms. A key goal of the JTRS program
- The use of standards based middleware like CORBA and SCA enables greater tool integration, supporting faster development through Model Driven Development and generative programming techniques.