SDR SIGNAL PROCESSING DISTRIBUTIVE-DEVELOPMENT APPROACHES

Dominick Paniscotti (VP SDR Products of PrismTech Corp., Burlington, Mass.; dominick.paniscotti@prismtech.com); Jerry Bickle (Chief Scientist SDR Products of PrismTech Corporation
6511 Constitution Avenue, Fort Wayne, IN 46804; jerry.bickle@prismtech.com)

ABSTRACT

The implementation of Software Defined Radios (SDRs) involves the development of software on various signal processing environments including General Purpose Processors (e.g., Intel® XScale™, IBM® PowerPC®), Digital Signal Processors (DSPs) (e.g., TI™, Analog Devices) and Field Programmable Gate Arrays (FPGAs) (e.g., XILINX, Altera).

JTRS Software Communications Architecture (SCA) [1,2] based waveform components developed for GPPs typically communicate with each other using CORBA® middleware, generally use the C++ language in their implementation and are layered on various POSIX®-compliant Real-Time Operating Systems (RTOS) (e.g., GreensHill®, VxWorks®, LynxOS®) as described by the SCA Application Environment Profile (AEP). However, this approach has not historically been used when such waveform components are targeted for DSPs.

The paper will discuss past, current, and advanced approaches used in the development of these waveform components in non-GPP based applications.

1. INTRODUCTION

Past (and even some existing) SCA/SDR development activities have decided to artificially limit SCA/SDR component framework implementations to operate only on General Purpose Processors (e.g., Pentium, XScale, PowerPC) and to use Adapter design patterns [4] on GPPs to communication with non-CORBA based DSP and FPGA component implementations (as shown in Figure 1).

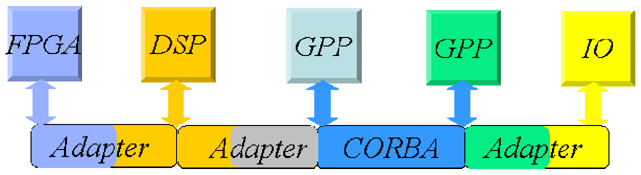

Figure 1. Adapter Illustration

However, for some time, Commercial Off The Shelf (COTS) implementations of CORBA have existed that support a larger array of SDR hardware processing elements (including DSPs, and FPGAs). Such COTS CORBA implementations obviate the need for these adapters (as illustrated in Figure 2).

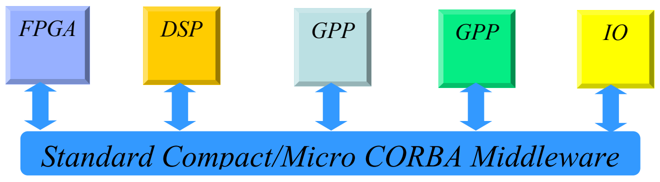

Figure 2. Standard Distributive Middleware Illustration

2. DISTRIBUTED PROCESSOR COMMUNICATION APPROACHES

Regardless of the type of distributed communication approach one takes, certain characteristics need to remain constant. These characteristics are as follows:

Information Requestor

- The request is encapsulated into a message protocol.
- The request identifies the intended recipient(s).
- The request is sent to the intended recipients (using a communications transport of some type)

Information Recipient

- The incoming request is retrieved (from the communications transport)
- The incoming request is identified as being for the recipient.
- The incoming request's message protocol is processed.
- The contents are handed off to the intended recipient for processing.

Depending on the Distributed Processor Communication approach there will also be some differences. These include but are not limited to:

- Synchronous communication capabilities being offered
- Asynchronous communication capabilities being offered
- The Quality of Service (QoS) associated with communication path(s) and its management
- The degree of development burden placed on the application developer when making and handling distributed requests.
- The extent to which the communication middleware provides isolation of the developer's application source code from the distributed processing implementation
- The memory size and performance footprint of the middleware implementation
- COTS development tool support for the communication middleware

In addition, as SCA/SDRs take a component-based approach additional distributed communication characteristics are important:

- The degree to which the SCA/SDR component framework can be implemented using the distributed processor communication chosen
- The extent to which the choice of distributed processor communication implementation interoperates with components on GPPs that use CORBA middleware, as well as the risk and development cost associated with such choices.
- The ability to maintain architectural consistency across disparate hardware
- The degree to which the SCA/SDR components are able to be migrated to other hardware targets (to support future technology insertion)
- COTS development tool support for component development.

As stated earlier there are two approaches for SCA/SDR component-to-component communication: Adapter Design Pattern or a distributed middleware technology (such as CORBA). The following sections will consider these approaches in detail.

3. ADAPTERS

The adapter approach is typically used where middleware solutions are not available or not practical. During the initial implementation of the SCA specification (circa 1999), there were few COTS CORBA implementations available for either DSPs or FPGAs. As such, most early developers used Adapter-based approaches to communicate between GPP-based SCA components and functional implementation of waveform behavior residing on DSPs and FPGAs (see Figure 3). Even though these early implementations chose to use Adapters to minimize risk, the SCA specification itself continued to depict solutions that envisioned a common middleware solution throughout the entire radio with the goal of achieving architectural consistency throughout.

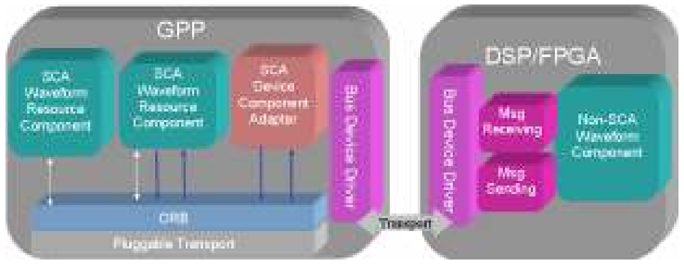
GPP
SCA Waveform Resource Component
SCA Waveform Resource Component
SCA Device Component Adapter
Bus Device Driver
ORB
Pluggable Transport
Transport
DSP/FPGA
Bus Device Driver
Msg Receiving
Msg Sending
Non-SCA Waveform Component

Figure 3. Adapters Commonality Illustration

3.1 HAL ADAPTER

The SCA does specify a particular abstraction called a "Device" where the use of the adapter pattern is warranted. These Device abstractions are used to provide isolation between SCA waveform components and the hardware facilities (e.g., serial, audio, digital if, antenna, crypto, gps, etc.) they depend on. This is typically referred to as a Hardware Abstraction Layer (HAL). These HALs isolate the particular physical hardware implementation from the abstract functionality that hardware offers. As such, physical hardware elements can be replaced with differing hardware offering the same functionality without affecting the waveform components that depend on that functionality. These SCA Devices perform functions such as:

- Receiving CORBA requests from GPP SCA Waveform Resource Components, and sending them on to behavior found in physical hardware elements
- Receiving requests from the physical hardware and sending a CORBA request to a GPP SCA Waveform Resource Component in response to that.
- Loading and executing waveform component software on a physical device (GPP, DSP, or FPGA)

Note that the implementation of these Devices were mandated by the specification to be reusable. Its goal is to isolate the waveform components from the physical hardware thereby allowing their portability/reuse.

3.2 COMPONENT ADAPTER

The Component Adapter approach involves a CORBA-based SCA waveform component communicating with a non-CORBA, non-SCA based "component" using an

Adapter. The non-SCA based "component" is not truly a component at all as it does not adhere to the SCA definition of a waveform component. It's simply a software or firmware (in the case of FPGAs) function running on some processor in the radio to which an SCA-based waveform component needs to communicate.
SCA Device's are often used to facilitate building these kinds of Adapters. The Component Adapter approach extends the behavior of the Device to support SCA component to non-SCA component communication. As such, the SCA Device no longer serves as a Device HAL (as described previously) but rather becomes part of the waveform application itself. In doing so, it breaks the waveform application portability tenets of the SCA. This Component Adapter approach can be implemented as a Component Level Adapter or as a Proxy Component Level Adapter.

3.2.1 COMPONENT LEVEL ADAPTER

With Component Level Adapters, the waveform component developer is saddled with the burden and responsibility of message formatting and processing for all component types (CORBA and non-CORBA) as shown in Figure 4. The Component Level Adapters are often mislabeled as HALs in industry.

The JTRS Modem HAL (MHAL) [5] is a classic example of a Component Level Adapter. Although its name would imply that it provides the abstraction of a radio MODEM, in actuality it acts to facilitate the communication between SCA waveform components and non-SCA waveform components. Its implementation needs to format messages that are to be sent from SCA-based components using CORBA mechanisms to non-SCA based components that typically use proprietary communications standards. The message formats are typically waveform specific with the Component Level Adapter acting as the conduit which encapsulates and routes the messages to their destination non-SCA waveform functions. To further complicate matters, the waveform application itself is responsible for tagging the data being sent so that it arrives at the proper non-SCA waveform function.

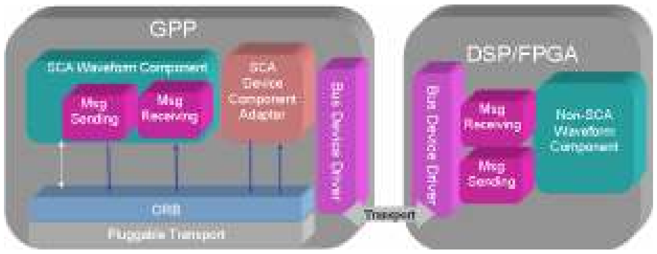

Figure 4. Component Level Adapter Illustration

As no standards exist to guide the development of these Component Level Adapters, synchronization behavior between the Adapter and the non-SCA waveform functions is generally ill-defined (which leads to further portability concerns). Lastly, as the non-SCA functions are not guided by the rules of the SCA, developers themselves are free to choose the degree of isolation they wish to apply between message communication function and waveform logic functions (as no communication middleware enforces this isolation). The impact is that this non-SCA based software may actually stifle future bus and transport choices, thereby hampering technology insertion.

3.2.2 PROXY COMPONENT LEVEL ADAPTER

In contrast to the Component Level Adapter, a Proxy Component Level Adapter dynamically launches new Component Level Adapters as appropriate to support communication between SCA-based components and non-SCA based software (as illustrated in Figure 5). In this approach, the message formatting and processing is not the responsibility of the waveform component developer but rather falls to the developer of the Proxy Component Level Adapter implementer. Proxy Component Level Adapters are usually implemented using the SCA ExecutableDevice abstraction. Their implementation typically has apriori knowledge of the waveform application being executed and as such launch the appropriate Component Level Adapters to support communication between a particular set of SCA waveform components and their associated non-SCA functions.

Regardless of the Adapter approach chosen, many similarities are found between them.

- The data is transferred over a local transport, such as a system bus, to a transport interface on a non-CORBA processor. The transport interface performs address decode and passes the data to the desired waveform function.
- The non-CORBA processor handles requests and sends requests.
- Neither Adapter approach is reusable or portable to different processor/transport technologies.
- The use of these Adapters involve extra layers of communication that impact performance.

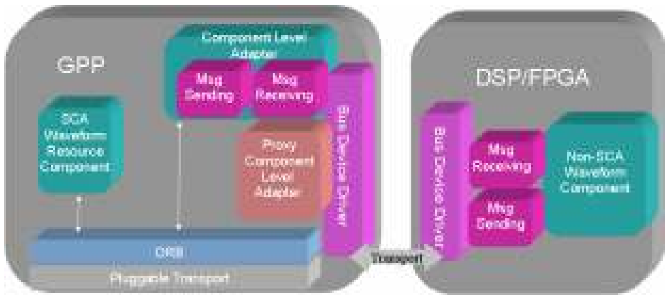

Figure 5. Generic Component Proxy Illustration

4. NEXT GENERATION CORBA

CORBA provides a standard facility for communications between heterogeneous platforms where the requestor is unaware of the recipient location and the requestor is isolated from the communication middleware details as shown in Figure 6. The recipient's Interface Definition Language (IDL™) files define the recipient's interfaces. The IDL is compiled by a CORBA IDL compiler to define the client side interface ORB behavior and the recipient's ORB server side behavior. The generated client and server side interface performs the ORB messaging behavior marshalling/un-marshalling of General Inter-ORB Protocol (GIOP) requests at the ORB level. GIOP message types are communicated over any connection-oriented transport. This combination of GIOP and the underlying transport provides the fundamental vehicle for the CORBA communications infrastructure. Most notably, the CORBA Inter-ORB Protocol (IIOP) specifies how GIOP is implemented over TCP/IP - all ORBs claiming conformance must implement at least IIOP. For resource embedded constrained environments IIOP is not the ideal choice due to size and performance. GIOP uses a simple and efficient Common Data Representation (CDR) to represent the binary layout for IDL types assembled for transmission. The CORBA CDR provides an encoding enforced by the IDL definitions used, aligns primitive types to natural boundaries, and supports byte ordering.

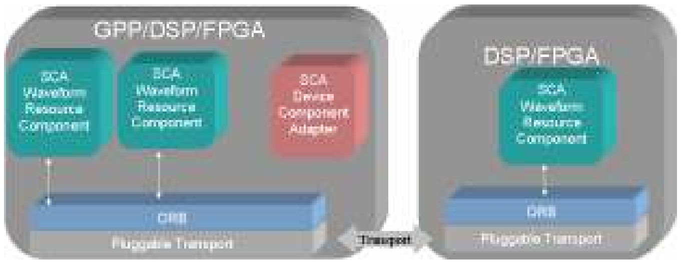
GPP/DSP/FPGA
SCA Waveform Resource Component
SCA Waveform Resource Component
SCA Device Component Adapter
ORB
Pluggable Transport
Transport
DSP/FPGA
SCA Waveform Resource Component
ORB
Pluggable Transport

Figure 6. Standard Software Bus Illustration

The component's communication paths could be on the same processor or not. The communication is direct, with no additional mechanisms required (as are found in Adapters).

New CORBA specifications have emerged over recent years targeted at the real-time and embedded domain (RT/E). These include:

- CORBA/e - introduced to address the most demanding requirements of size and performance-contrained embedded applications without forfeiting the interoperability, portability and platform independence which SDRs benefit from the use of CORBA.
- The Real-time CORBA specification - an optional extension that provides facilities that support deterministic behavior by promoting end-to-end predictability in distributed systems. The Real-time specification is part of the CORBA/e profiles.
- Another key technology utilized is the CORBA Extensible Transport Framework (ETF). This framework allows the development of standard and efficient protocols to support optimized communication between ORBs. The ETF allows the flexibility to implement protocols other than TCP/IP (the CORBA default) for real-time systems, including highly optimized shared memory performance transports with zero copy behavior over RapidIO and compactPCI. PrismTech's family of embedded ORBs, including ICO™, supports ETF.

New disruptive technologies are often met with skepticism and reluctance to adopt them. Technologies such as higher-order languages and compilers were initially scoffed at as were operating systems for the embedded domain by software practitioners who were concerned by the overhead they introduced. Time (and a combination of Moore's Law [6]) has proven these technologies to reap far more benefit than the performance impact they introduced. Similar arguments are made by SDR developers when the use of COTS middleware is proposed to solve the issues discussed in section 3.2. In fact many of these same concerns were raised when CORBA was chosen as the middleware technology choice for the JTRS SCA.

4.1 DSP ORBS

The principal challenges arising from the adoption of standards-based solutions on such platforms relate entirely to the limited resources that are typically available in the application domain where such devices are used. These problems have been largely offset by the various initiatives that have developed in the CORBA space, including low-footprint CORBA profiles, real-time, and the capability to externally adapt the ORB core to support native data transports. In addition, ORBs supporting the IDL to C language mapping provide a particularly fit-for-purpose technical solution in a domain where the use of object oriented languages (C++) is scarce, and tool-chain support is limited. CORBA ORBs are available for C and C++ implementations, and have been highly optimized for embedded environments such as DSPs. In fact DSP ORBs have been used to support SDR implementations going back to early in the year 2000 on the Digital Modular Radio (DMR) program.

For PrismTech's OpenFusion™ C e*ORB™ [7] for a compact profile has basically 63K for the client ORB and 87K for the server ORB. A micro profile C e*ORB would be smaller yet. A C++ ORB would be about 4 times larger than this.

4.2 HARDWARE ORBS

Hardware based ORBs are also now emerging (such as PrismTech's Integrated Circuit ORB (ICO™)) [7]. Hardware elements of a radio system may now be made CORBA compliant and reap the benefits of software portability. This brings the portability of the Software Communications Architecture (SCA) onto FPGAs and ASICs.

While CORBA can be hosted on an FPGA using a soft processor core and a conventional software ORB, greatly improved performance can be achieved using a hardware ORB. A hardware ORB is a CORBA ORB written in Very High Speed Integrated Circuit Hardware Description Language (VHDL) and designed specifically for FPGAs. Implementations such as PrismTech's patent-pending OpenFusion Integrated Circuit ORB (ICO) provide a subset of CORBA functions required to support the most commonly used communication patterns. While specifically targeted for use in high performance SDR applications and can be used to help ensure compatibility with the Software Communications Architecture (SCA), it is primarily a CORBA IP core and can also be used in applications with no SCA requirements.

Operating at hardware data rates and without the unnecessary overhead of a GPP proxy object to communicate with an FPGA, a hardware ORB can provide significantly better performance than a software ORB. A hardware ORB can process a message in a few hundred nanoseconds, hundreds of times faster than a conventional software ORB. In sustained tests, a hardware ORB can typically process well over a million CORBA messages per second.

5. TOOLS

Developing these CORBA-based SCA components can be quite a difficult task. Tools have emerged over the past several years that automate the development of these components using Model Based approaches. The initial focus of these tools was to develop SCA/SDR components for GPPs using C++. Recently, support has begun to emerge for the C language and VHDL (to support DSPs and FPGAs) As an example, PrismTech's Spectra development tool suite for Software Defined Radio (SDR) offers support for C++, C and VHDL and supports several RTOSs including WindRiver VxWorks, GreeHills Integrity, LynuxWorks LynxOS and several Linux distributions. These types of tools bring the portability aspects of the Software Communications Architecture (SCA) to DSP and silicon devices (such as FPGAs). Using PrismTech's Spectra SDR Power Tools [8] a component developer can model a waveform application and transform those models into C++, C, and/or VHDL.

6. CONCLUSION

Technology now exists that provides the realization of the SCA/SDR 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. Finally, in modern multi-processor systems composed of GPPs, DSPs and FPGAs the overall throughput of the system can be improved by using DSP ORBs and hardware ORBs (on FPGAs) as they obviate the need to add Adapters and their associated proprietary transport mechanisms.

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.
- IBM® and PowerPC® are registered trademarks of International Business Machines Corp
- INTEGRITY® and Green Hills® are the registered trademark of Green Hills Software, Inc.
- Intel® and XScale™ are registered trademarks of Intel Corporation
- Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries
- LynxOS® is the registered trademarks of LynuxWorks, Inc.
- Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries
- POSIX® is a registered trademark of IEEE.
- e*ORB, Spectra, Spectra SDR, Spectra Power Tools, ICO, and OpenFusion 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] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley Professional Computing Series

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

[6] Moore's law, http://www.intel.com/technology/mooreslaw/

[7] PrismTech OpenFusion CORBA Products, http://www.prismtech.com/section-item.asp?id=570&sid=18&sid2=10&sid3=251

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