

# 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.

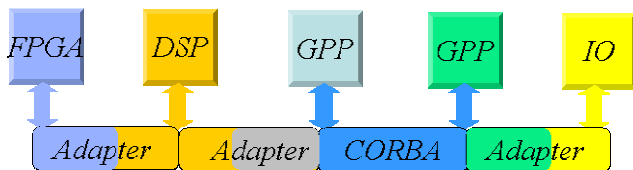


Figure 1. Distributive Communication Approaches

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







