Next Generation Architecture for Heterogeneous Embedded Systems

S. Murat Bicer, Frank Pilhofer, Graham Bardouleau, Jeffrey Smith
Mercury Computer Systems, Inc.
Chelmsford, MA, U.S.A.

May 5, 2003

Abstract

The Software Communications Architecture (SCA), a mandatory specification for Software Radio implementations by the Joint Tactical Radio System (JTRS), defines a Common Object Request Broker Architecture (CORBA) based component model for building portable applications in a heterogeneous environment. The Object Management Group's (OMG) CORBA is an accepted architecture for distributed systems that recently added a component model to its suite of standards. The authors' effort in leveraging the strength of CORBA by reusing OMG standards within the SCA, and improving OMG standards to match JTRS expectations, yields synergies that will broaden the vision of SCA as well as easing implementations and improving scalability within the SCA framework. A case study shows seamless integration of FPGA components into an SCA application. The Streaming Component Environment (SCE), a Mercury product that provides this kind of flexibility within our current high-performance embedded systems, is being extended to comply with the OMG and SCA specifications.

1 Software Communications Architecture

Today's rapid pace of technological advances make communication devices obsolete shortly after they are produced. To keep up with this pace, communications systems must be designed to maximize the transparent insertion of new technologies at virtually every phase of their lifecycles. When these new technologies are inserted, the upgraded devices should still be able to communicate with each other and with legacy systems.

The term *software defined radio* was coined in 1990s to overcome these problems. A software defined radio is a communications device whose functionality is defined in software. Defining the radio behavior in software lets one add new functionality without hardware alterations during a technology upgrade.

In order to maintain interoperability, the radio systems must be built upon a well-defined, standardized, open architecture. Defining an architecture also enhances scalability and provides plug-and-play behavior for the components of a radio.

The Software Communications Architecture (SCA) is an open architecture defined by the Joint Tactical Radio System (JTRS) Joint Program Office (JPO). The SCA has been published to provide a common open architecture that can be used to build a family of radios across multiple domains. The radios built upon SCA are interoperable, can use a wide range of frequencies, and enable technology insertion. The SCA also supports software reusability.

The SCA defines an Operating Environment (OE) that will be used by JTRS radios. It also specifies the services and interfaces that the applications use from the environment. The interfaces are defined by using the Common Object Request Broker Architecture (CORBA) Interface Definition Language (IDL) and graphical representations are made by using Unified Modeling Language (UML)

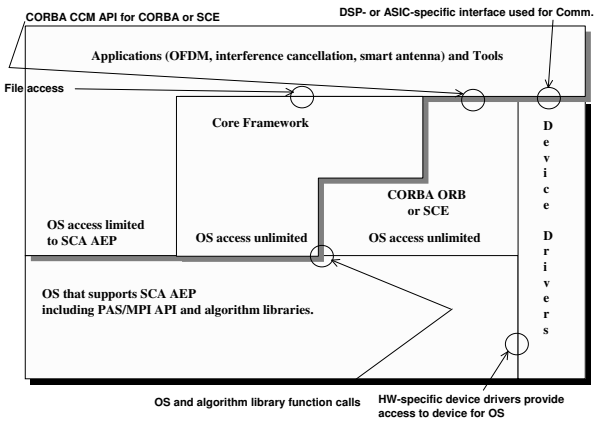

Figure 1: SCA Framework

[1].

The OE consists of a Core Framework (CF), a CORBA middleware and a POSIX-based Operating System (OS). The OS running the SCA must provide services and interfaces that are defined as mandatory in the Application Environment Profile (AEP) of the SCA. The CF describes the interfaces, their purposes and their operations. It provides an abstraction of the underlying software and hardware layers for software application developers. An SCA compatible system must implement these interfaces. The interfaces are grouped as Base Application Interfaces, Framework Control Interfaces and Framework Services Interfaces.

The CF uses a Domain Profile to describe the components in the system. The Domain Profile is a set of XML files that describe the identity, capabilities, properties, inter-dependencies, and location of the hardware devices and software components that make up the system [2].

Although the SCA uses the CORBA middleware for its software bus, the application layer can reach the OS by other means. CORBA adapters can be used to wrap the legacy software components. Figure 1 shows the relationship between the AEP, the application and the OE.

2 SCA Issues

Application portability versus performance is a common trade-off. SCA suffers the same dichotomy. The vendors implement waveforms using Application Specific Integrated Circuits (ASIC) and Field Programmable Gate Arrays (FPGA) in their platforms and still be SCA compliant with the help of CORBA adapters. This approach hurts portability and results in platform specific implementations. Waveforms expect a lightweight component run-time environment that is not yet commercially available.

SCA is currently *out-of-synch* with CORBA Component Model (CCM) and services for embedded environments. These specifications were not ready when SCA was written but they do cover the embedded profiles now and SCA can evolve by including pointers to these specifications.

Neither CCM nor SCA addresses scalable embedded multiprocessing (where a component has a data-parallel implementation) or reusable/interoperable wideband dataflow. By defining extensions based on experience and working them through standards, reusable definitions for Digital Signal Processor (DSP) and FPGA codes, data parallel embedded computer support and wideband interoperable dataflow can be made a part of SCA [3].

In SCA, components are connected to each other through *ports*. One of the main obstacles in developing interoperable SCA applications is the loose definition of ports in the specification. Components can have *provides* and *uses* ports. The CF creates component assemblies by connecting the provides port of a component to the uses port of another component. The communication model that will be used after this *connect* operation is not defined which could lead to interoperability problems when connecting components from different vendors.

Component implementation spanning processors can be supported by SCA by including Data Parallel CORBA [7] features to the standard e.g. partial objects and parallel servers. Striping, distributing and scattering streams across multiple processors for streams from I/O ports to processors and among processors can be supported by allowing alternative assemblies of components based on QoS and hardware. Mercury's leadership in developing the Data Parallel CORBA specification was based on years of internal research and realized by the proprietary middleware *Streaming Component*

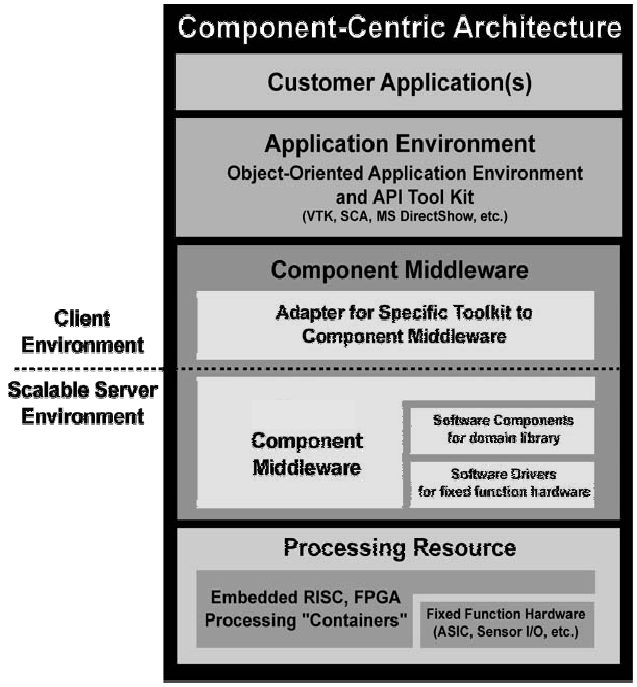

Figure 2: Mercury Component Centric Architecture

Environment (SCE) that supports seamless integration of FPGA components into SCA compliant systems using high-speed interconnects (see Figure 2).

3 Component Architectures

3.1 CORBA

CORBA [4] is an open, vendor-independent infrastructure for distributed computing, published by the OMG, a consortium of more than 800 members from industry and academia. CORBA is based on the object oriented model; objects are implemented in a server and can be addressed in a location independent manner using opaque object references. A declarative language, IDL, is used to describe an object's public interface. A language specific compiler translates an IDL description into programming language constructs that hide the potential remoteness from the developer. In the C++ language, interfaces are translated into classes. Remote objects appear as local proxy objects ("stubs") that automatically send the request, mediated by the Object Request Broker (ORB), to the server. After processing, a reply is sent back the same way. Object Request Brokers communicate across hosts using the efficient General Inter-ORB Protocol (GIOP) [6]. Within the same hosts, ORBs may use other transports (like shared memory). If the target object exists in the same process, the "remote" invocation is usually as efficient as a virtual method call in C++. The standard protocol and the variety of standard language mappings ensure cross-vendor and cross-language compatibility: Java clients easily interoperate with C++ servers or vice versa.

In addition to CORBA, the OMG has defined numerous add-on standards for common services like the Naming, Event or Transaction service, tie-ins like Secure Interoperability and Fault Tolerance, and also profiles for e.g. real-time systems.

3.2 CORBA Component Model

The CORBA architecture is based on a centralized approach to computing: requests are sent to a server for processing, assuming monolithic, standalone servers. This model is suitable for a wide range of applications, including three-tier (Frontend, Server, Database) architectures. However, it does not fit modern peer-based architectures very well, as it does not describe *interaction* between servers on a peer to peer basis.

In 2000, the OMG adopted CCM [5] as an extension to the object model. CCM advocates a concept in which components, in their IDL description, advertise *ports*. CCM defines interface and event ports; a component can provide ports ("facets") or use ports ("receptacles"). A component implements the interfaces of provided ports, but at the same time it enables the component to express dependencies on other interfaces that must be supplied by other components. At deployment time, when a component-based application is to be executed, the various components are first instantiated and then interconnected according to an *assembly* description.

By avoiding the monolithic approach of centralized servers, CCM is more attractive to fine grained distributed systems. In combination with the ability to assemble components into an application,

CCM better promotes code and component reuse: the smaller components are, the better the chances that they can be reused.

Small software components also promote parallel programming; ideally, components will execute in parallel, exchanging data and messages for synchronization.

3.3 Lightweight CORBA

In the past, CORBA has been criticized for its complexity; the size of existing ORB implementations made CORBA unattractive to embedded applications. The OMG addressed the issue with the Minimum CORBA specification, but there was a chicken and egg situation. Small ORBs are feasible at the cost of a limited feature set, but as long as there was no customer demand for small-footprint, all vendors offered were full-featured, large, ORBs.

This situation has changed. Tighter integration of heterogeneous hardware in the same embedded system created a demand for vendor-independent standards like CORBA, instead of proprietary protocols. At the same time, loosened memory and performance constraints closed the gap to low-footprint ORBs that are now commercially available.

The authors are actively involved in tailoring OMG specifications for the embedded domain. This includes Lightweight Logging, Lightweight Services and Lightweight CCM. The recently adopted Lightweight Logging Service defines a minimal-footprint logging service where distributed log messages are time stamped and recorded for later retrieval. Lightweight Services is a profile of the common Naming, Event and Time services with the goal of removing redundant features from the "full" services. For example, while the original Event Service defines both push and pull event models, Lightweight Services removes the pull model.

Lightweight CCM also removes some of the CORBA Component Model's features that are redundant or not required in the embedded domain, such as support for persistency. It is anticipated that the Lightweight CCM specification will enable CORBA components to be implemented with as little footprint as current CORBA servers.

Initial submissions for the Lightweight Services and Lightweight CCM have been received and are currently being discussed within the OMG, adoption is anticipated before the end of the year.

3.4 Deployment and Configuration of Component-based Applications

A shortcoming in the current CCM is its limited facilities for assembly and deployment of component-based applications. The authors are working on a new add-on standard that will greatly improve the situation. The submission is based on several important concepts and features:

- Components can be implemented monolithically, or by an assembly of interconnected subcomponents.
- Multiple alternative implementations of the same component can be packaged and distributed as a single file.
- Implementations describe their requirements. These are matched against resources in the target environment.
- Standard interfaces and data models ensure interoperability across vendors.

The result will be a powerful specification that will greatly enhance the CCM.

In the current CCM, assemblies are limited to a single level of composition. Our specification allows assemblies to be recursive. Rather than having a separate identity, assemblies always implement a concrete component interface, enabling composition and decomposition at any level: components can be glued together into an assembly, or they can be transparently replaced by an assembly that implements the same interface. In an assembly, a component's "surface features" (attributes and ports) are mapped to attributes and ports of subcomponents.

Allowing to package alternative implementations addresses the heterogeneity in a distributed system. Distributors can then for example include equivalent Linux, Windows and Java versions in

the same package; an implementation can then be chosen based on the target environment and user preferences.

A data model is introduced to model available resources, such as basic features (e.g. CPU and OS type), sharable but limited hardware (such as CPU, Memory), non-sharable hardware (e.g. a microphone), and interconnect bandwidth. Monolithic component implementations describe their requirements, and assemblies describe the requirements of interconnections. During a "deployment planning" process, concrete implementations are selected among the alternatives so that resources match or exceed requirements.

Interfaces are defined in IDL, data models are defined in both IDL and XML to transport data between vendors both at runtime and offline (in files). This allows for different parts of the infrastructure to be supplied by different vendors.

In particular, hardware vendors can support their specialty hardware by supplying only two pieces. The first is a means to implement a component, resulting in a component implementation that is packaged according to the standard data models. The deployment infrastructure treats component implementations as opaque. They are just passed to a well-defined entity in the target environment that is responsible for instantiating a running component from the implementation. This "node manager" is the second piece that needs to be supplied by the vendor. All the deployment infrastructure needs to know about a component implementation is described in well-defined XML files.

This allows for a wide range of concepts, from on-demand compilation to scripted implementations to DSP code to VHDL code that is to be loaded onto an FPGA.

The vision is that, by virtue of the Deployment and Configuration specification, users need to acquire only a single deployment infrastructure, plus the (multiple) vendors' development and execution chains for the hardware in their environment. In combination with other features of the specification, users are enabled to implement their distributed, component-based applications in a standard, vendor-independent way. The ability to include multiple implementations in a single package ensures portability, and the possibility to add new pieces of accelerated hardware, to port only the "hot components," or to selectively decompose components into finer-grained assemblies, allows for present and future application scalability across a wide range.

Using the OMG standard Model Driven Architecture (MDA) approach, the specification is expressed in terms of a Platform Independent Model that does not rely on a concrete component model. Instead, the model is then mapped into one or more Platform Specific Models. So far, mappings for both the CCM and the SCA are included.

A final candidate of the specification has been submitted, adoption is anticipated before the end of the year.

4 Next Generation SCA

The SCA predates the CCM, but its component model of Resources that are interconnected via ports is comparable. The SCA also defines some other services that are not just relevant to the Software Radio domain, but useful to embedded systems in general. Most notable is the logging service; for the common naming and event services, the SCA defines a subset in order to cut to size both servers and clients.

This was one of the premises that started the aforementioned "lightweight" CORBA specifications: the parts of the SCA that are not limited to the Software Radio domain should be separated into standards of their own. The SCA specification can simply include pointers to the separated standards.

The authors expect several synergies from this process. The SCA specification will become smaller. SCA Core frameworks can then be implemented and used with more existing, standard pieces of software. On the other hand, CORBA will benefit from stand-alone, lightweight services for the embedded domain.

The specification about Deployment and Configuration should make a special impact. The deployment facilities that are defined by the SCA are not well specified and a frequent cause for confusion. Deployment "happens" by way of an Application Factory, but the distribution of Resources

among Devices is largely undefined. The SCA is also limited to a single level of composition.

The SCA mapping of the Deployment and Configuration specification includes a much more powerful deployment concept and defines clear boundaries between the core framework, the deployment system, devices and the component implementation. When adopted into the SCA core framework, the Deployment and Configuration specification will enhance the SCA standard with its support of heterogeneous systems, FPGA and DSP systems and the ability to transparently integrate new hardware environments.

5 Streaming Component Environment

At Mercury Computer Systems, Inc., an enabling technology called SCE is being developed based on a combining of software and hardware component architectures.

The SCE is a component based middleware that utilizes CORBA for the control plane and an in-house developed data plane. The key requirements for the SCE technology are reuse, portability and scalability while not significantly affecting performance when compared with a hand-coded application performing the same function. An additional area of interest was the reduction in time to develop an application using the SCE product.

The amount of reuse achievable in a component system is dependent on two independent issues; the first is the level of isolation between the component and the underlying operating system or hardware platform. Within SCE it was important to provide the maximum isolation and hence, the highest possibility for reuse. This has been achieved by performing trampolining of functions from the components and providing a dynamic loading and linking approach that makes the components completely independent of the operating environment. The second reuse issue is component granularity; a component is only reusable if the functionality that it provides is required by another application. This issue can only be addressed as a learning lesson. We have been reasonably successful in componentizing applications to maximize reuse of the components.

Component portability is achieved by reducing the number of system level dependencies. Also, the infrastructure in which the component executes must be easily ported. The SCE is a layered product, in which components execute within a container infrastructure. Once a container has been ported to a given platform/processor, components can be executed on that platform.

Given the complexity of some algorithms and the size of data sets associated with these algorithms, it is necessary to be able to distribute the processing across more than one processor or FPGA. To permit this, the SCE supports complex data reorganization and distribution techniques. A component is provided information about the segment of the data set present in a data payload. This allows a component to be written in a manner that is independent of the scaling of the algorithm.

The SCE supports a signaling mechanism that is common between all of the supported devices, including microprocessors, FPGAs and DSPs. All data and signals are transferred concurrent with processing; hence very high processor utilization can be achieved.

The design of the SCE predates both SCA and CCM. However, we are working on making the SCE compatible with these standards, so that the SCE can be used to build both SCA and CCM compatible applications.

6 FPGAs as Components

With the increasing adoption of component models for multi processing applications, especially in complex areas such as communications systems, the need to support unconventional processing resources such as FPGA and adaptive logic devices increases.

At Mercury, we have already demonstrated seamless integration of FPGA-based hardware components into SCE applications.

Software component architectures provide interfaces that facilitate communications between components. This isolates the components from many of the platform specific issues, allowing portability at the cost of some level of performance. Given

the increases in processor performance, and the increasing costs of software development, it is generally acceptable to lose some performance for increases in reusability, and hence improvements in time-to-market.

FPGA based computing is gathering pace in some segments of the computing and processing marketplaces, due to the high performance that can be achieved for some types of algorithms compared to software based implementations. Unfortunately, the majority of current systems adopt one of two approaches to integrating FPGA devices into a heterogeneous system. The first approach closely couples an FPGA device to a more conventional general-purpose processor (GPP) that is responsible for interfacing with a system via a software component system and the FPGA performs computation on the data sets. This approach is expensive, given the need for a GPP for each FPGA, and has performance issues due to the communication required between the two. A second approach is to treat the FPGA as a device requiring a proprietary device driver running on a GPP that is responsible for controlling the FPGA data flow.

There are both advantages and disadvantages to any attempt to integrate an FPGA into a component architecture. As with any component architecture, there can be significant benefits in the ability to reuse the component. This would be true for FPGA based component architectures. Any infrastructure necessary to implement the component architecture will consume resources on the FPGA device itself. This is an area that requires careful consideration, as the resources available on a given device are limited.

To maximize the reuse of an FPGA component, it is necessary to remove platform specific implementation details from the component itself through the use of well-defined interfaces. These platform specific details include memory controller physical, bus interface, DMA controller and I/O port implementations. If this can be achieved, the FPGA component can be ported to any FPGA device meeting the requirements for the component that provides matched interfaces for the abstracted functionality.

The benefits of a standard set of interfaces for internal FPGA facilities is significant from a technical point of view, however the business view may not be seen as so compelling. The ability to purchase off the shelf FPGA components that comply with standard interfaces permits the development of products with reduced in-house skill sets. It also provides a secondary market opportunity for in-house developed cores, allowing those cores to be sold externally. The standardization of the interfaces also permits FPGA board vendors to provide a framework that supports the interfaces, allowing integration of FPGA boards into existing component architectures.

To achieve a truly flexible component based infrastructure, it is necessary to provide a scheme that allows the transfer of data payloads and state signaling. With this additional functionality, it is possible to integrate software and hardware components together in a heterogeneous system. To achieve this it is necessary to provide interfaces to an FPGA component that indicates the availability of data payloads, and allows the FPGA component to indicate that data payloads have been processed and new output payloads have been created.

7 Conclusion

The existing SCA standard supporting waveform interoperability was described. In the next generation SCA, the platform independent and domain specific portions of the specification are segmented into OMG standards providing greater commercial implementation opportunity. With the advantage of hindsight, refactoring the existing SCA specification and input from the commercial and military sector, SCA revisions are addressing key scalable embedded processing issues e.g. interchangeability of software and heterogeneous hardware components, high performance interprocessor communication, lighter weight CORBA-related specifications and support for low-level embedded standards and parallel objects, making the SCA development process more transparent to the waveform developer. This specification work is further validated with a testbed demonstrating internal SCA multiprocessing research with respect to the heterogeneous component SCA/SCE middleware and sample waveforms. The evolution of this next gen-

eration SCA middleware will be demonstrated in MILCOM [8] in October.

References

[1] Modular Software-programmable Radio Consortium. *Software Communications Architecture Specification*. MSRC-5000SCA, V2.2, November 17,2001.

[2] Modular Software-programmable Radio Consortium. *Support and Rationale Document for the Software Communications Architecture Specification*. MSRC-5000SRD, V1.2, December 21, 2000.

[3] J. Smith, J. Kulp, M. Bicer, T. Demirbilek, J. Anton. *SDR - Do You Care to Buy the Softest?* In Proceedings of the Military Communications Conference - Mobile Communications and Military Transformation, Washington, D.C., March 2003

[4] Object Management Group, *CORBA 3.0.2*. December 2002.
www.omg.org/cgi-bin/doc?formal/02-12-02

[5] Object Management Group, *CORBA Component Model*. June 2002.
www.omg.org/cgi-bin/doc?formal/02-06-65

[6] Object Management Group, *General Inter-ORB Protocol*, Version 1.3. June 2002.
www.omg.org/cgi-bin/doc?formal/02-06-19

[7] Object Management Group, *Data Parallel Processing*. November 2001
www.omg.org/cgi-bin/doc?ptc/2001-11-09

[8] Military Communications Conference - MILCOM, Boston, MA, October 13-16, 2003.
www.milcom.org/2003