THE DEPLOYMENT OF SOFTWARE COMPONENTS INTO HETEROGENEOUS SCA PLATFORMS

Steve Bernier (CRC Ottawa, Canada; steve.bernier@crc.ca);
Juan Pablo Zamora Zapata (CRC Ottawa, Canada; juan.zamora@crc.ca);

ABSTRACT

This paper examines the JTRS goal of maximizing reusability of common software and hardware for different software defined radios. It does so by analyzing the different aspects of reusability through portability of SCA applications. It looks at binary reuse as well as at source code portability. For the latter, the paper describes how the SCA enhances portability when dealing with aspects such as Operating Systems, Middleware and Platform Devices. The paper also describes the different mechanisms provided by the Software Communications Architecture (SCA) to deploy software components. In doing so, it provides a detailed discussion on how the SCA addresses operating environments that support multiple processes as well as those that don't. Finally, the paper provides guidelines on how to map the SCA Loadable and Executable Devices to the different Computational Elements (GPPs, DSPs and FPGAs) that can be found on an SDR system,

1. INTRODUCTION

The Joint Tactical Radio Systems (JTRS) is a US DoD program. Its objective is to provide the DoD with needed communications capabilities through a family of affordable radios. The JTRS radio sets are Software Defined Radios (SDRs) that can be used to quickly field new communications capabilities. The JTRS program achieves its goal by maximizing the reusability of common software and hardware for different radios. This paper examines the different aspects of reusability through portability of SCA applications. It concentrates on the portability of the assembly implementation and underlines the role of the platform provider in portability. In essence, this paper provides guidelines for designing SCA devices for platforms with different kinds of computational elements.

The remaining of the paper is structured as follows: section 2 describes the different aspects that impact reusability and illustrates how the SCA deals with them. Section 3 talks about the need to map different computational elements (GPPs, DSPs and FPGAs) to SCA containers. Section 4 presents the conclusions of the paper. Section 5 provides the references used in the elaboration of this document.

2. REUSABILITY THROUGH PORTABILITY

An SCA application is defined as an assembly of interconnected software components. The level of reusability for an application is a function of the level of portability of its software components and of the assembly description. For the latter, the SCA addresses the issue by having all of the information related to the assembly of the application components stored in a number of descriptors defined by the SCA as the XML Domain Profile. The descriptors define how each component of the application must be instantiated, connected, and configured. They contain the requirements that must be met by the operating environment in order to run each component of an application.

Portability of the application implementation is addressed in the following subsections. In section 2.1 we first explore the reuse of binary implementations without modifications. Section 2.2 then considers a second approach that aims to enhance the portability of source code.

2.1 Implementation Reuse

Ideally, every component of an application would have a single implementation, which would be reused and run on any processor. A way of achieving this goal is to use an interpreter or a virtual machine. This solution is not very common in embedded systems since it usually cannot provide the level of performance required by waveform applications. Another way of achieving this goal is to use a family of processors that are binary-compatible with each other. This option implies that platform vendors would need to standardize on a single processor family, which would severely constrain SDR developers.

2.2 Source Code Portability

Considering that reuse of binary implementations are handicapped by either the existence of performance requirements that cannot be met, or because implementations require homogenous hardware environments (which is not probable), an alternative and more realistic approach is to port the source code of components. Porting source code implies the creation of new

implementations of a software component using a different tool chain (compiler, linker, etc.) specific to a different processor and operating system. The extent of changes that must be performed in the source code in order to create the new implementations defines the level of portability. The more modifications are required, the less portable the source code is considered.

Thus the goal of the SCA specification is to achieve the highest level of portability for the source code of applications. The target here is to reuse as much of the original source code as possible. The SCA provides a means for developers to write portable source code. It does this by enabling the source code to have a high level of independence regarding the operating system, middleware, and devices it uses. In the following subsection we explore each of those areas

2.2.1 Operating Systems

Every software component must use operating systems calls for things like threads, synchronization, file access, and such. The SCA specification requires that application software be restricted to the use of a standard set of system calls defined in the POSIX Application Environment Profile document [1]. This set of system calls must thus be provided with every operating environment.

2.2.2 Middleware

The SCA also requires the use of a standard middleware for communications between components. SCA applications are made of several interconnected components that must interact with each other. Therefore, each component contains a significant amount of code for inter-component communications. Standardizing the middleware helps achieve a greater level of portability. The SCA provides the choice of two communications schemes: CORBA [2] or MHAL [3]. CORBA is a middleware used for components usually hosted by a general purpose processor (GPP). Recently, CORBA also started to become available for Digital Signal Processors (DSPs) and Field Programmable Gate Arrays (FGPAs) [4, 5]. MHAL is a low-level communications layer that is required for communications between components when CORBA is not available.

2.2.3 Platform Devices

Waveform applications must interact with physical devices to get access data and/or relevant information about the system (e.g., radio signals, time). This is normally done using device drivers, which often provide different APIs from one manufacturer to another. Therefore, in order to maximize portability, the JTRS requires that every operating environment provide a standardized set of APIs for access to hardware components. So far, the JTRS program has standardized the APIs for a number of devices such as an Ethernet device, a Serial port, and a Vocoder [6]. The Software Defined Radio Forum also has a number of working groups developing standard APIs for devices such as antennas and transceivers.

2.3 Installation, Instantiation and Connection

When an application is ported to a new operating environment, a fair amount of time can be spent changing the way a component will be installed, instantiated, connected, controlled, etc. For instance, different operating systems can offer distinct ways of creating a process. As a matter of fact, some operating systems support the concept of a process while others don't. Different platforms can also support different file system technologies, which will have an impact on how an application is installed and instantiated.

The SCA Core Framework specification [7] addresses this issue by defining a deployment engine that must be part of every operating environment. The deployment engine standardizes how applications are handled. The specification also defines how component connections are to be established, including 5 different alternatives on how to locate (direct or indirect) the source and target components of such connections [8].

The Core Framework specification describes a deployment model by which software components are hosted by containers offered by the platforms. The Core Framework deployment engine uses the different containers of its operating environment to deploy application components.

The SCA defines two types of containers: the ExecutableDevice and the LoadableDevice. In section 3 we discuss in depth how the SCA can map those containers to computational elements.

3. MAPPING THE SCA CONTAINERS TO HARDWARE

SDR hardware platforms can provide three types of Computational Elements (CEs): General Purpose Processors (GPPs), Digital Signal Processors (DSPs), and Field-Programmable Gate Arrays (FPGAs). But the SCA Core Framework deployment engine uses only two types of software containers to deploy applications. Consequently, a relationship must be defined between the SCA containers and the different CEs of a platform. This process is called mapping the SCA to a platform.

The mapping consists in creating a software abstraction layer for the hardware. Each CE must be represented by an SCA container, which acts as a proxy to the hardware. These SCA containers along with the SCA Devices described in

section 2.1.3 are sometimes referred to as an SCA Board Support Package (BSP). A software component can be made of a number of artifacts (e.g. shared libraries, configuration files, main executable). Those artifacts are deployed into SCA containers by the SCA deployment engine.

In the remaining of the section we address the characteristics of the two types of SCA Devices, and how they can be mapped to the three types of CEs.

3.1 SCA Containers Characteristics

The key in developing an abstraction layer is a good understanding of the core characteristics of the two SCA containers. This helps identify which type of SCA container should be a proxy for a specific CE.

The SCA LoadableDevice is a type of container that is used by the deployment engine to load artifacts of software components. This container is capable of loading new artifacts, but not capable of launching new tasks (also known as processes). This type of container is therefore not capable of instantiating a component.

The SCA ExecutableDevice is a type of container used by the deployment engine to load and execute artifacts of potentially many software components. It is capable of creating several tasks at the same time without having to reboot to launch a newly loaded task. This type of container is therefore capable of instantiating a component. The deployment engine is responsible for the life-cycle of every component it instantiates. The life-cycle includes how a component is launched, the value of input arguments for launching, when the component is initialized, what property values are to be used as the initial configuration, and when the component is started. The life-cycle also includes the tear-down of a component which controls when a component is stopped, disconnected, and released. The SCA specification requires that an application be made of at least one SCA-compliant component. Therefore an SCA platform needs to provide at least one ExecutableDevice

The deployment engine cannot use an SCA LoadableDevice for launching components that implement the SCA standard life-cycle. This implies amongst other things that the deployment engine cannot feed such a component with a variable initial set of configuration properties and that it cannot connect it with other components. In essence, using an SCA LoadableDevice implies some level of hardcoding that negatively impacts the level of portability.

Ideally, all the containers provided by an SCA platform would be of type ExecutableDevice to maximize portability. The problem is that most often ExecutableDevices are mapped only to GPPs, and very seldom used with the other more powerful computational elements like DSPs or FPGAs often used in SDR platforms.

There are two key features provided by computational elements that can be used to differentiate a LoadableDevice from an ExecutableDevice: incremental loading and multi-tasking.

Incremental Loading is a feature that allows new binary code to be loaded into execution memory during runtime without rebooting the CE. Most operating systems that provide this feature do so through what is called a loader [9]. For instance, the POSIX standard [10] provides access to a loader through the ‘exec()’ system calls.

Multi-tasking is a feature that allows multiple tasks to share a single CE. It allows several tasks to run concurrently without requiring a reboot every time a new task is launched. Most operating systems implement this feature through what is called a task scheduler [9].

3.2 Mapping SCA Containers to Computational Elements

3.2.1 Mapping GPPs

GPPs are always represented as SCA ExecutableDevices since they provide support for both incremental loading and multi-tasking.

3.2.2 Mapping DSPs

DSPs are almost never represented as an SCA ExecutableDevice because they are not traditionally used with a loader. As well, most DSPs are used without multitasking support. Therefore DSPs are most often represented as SCA LoadableDevices.

Even when a DSP is used with a multitasking scheduler (e.g. DSP/BIOS), it can only be represented as an SCA LoadableDevice when it does not support incremental loading. That is because it can’t be used to load new artifacts to create new tasks. In other words, it cannot launch new components. It can only create new tasks using the binary code that was originally loaded. In this case, reloading the DSP will inevitably change its behavior. In other words, reloading the DSP implicitly starts a new task and that is why there is no need for the SCA ExecutableDevice API.

However, it is possible to represent a DSP as an executable using an operating system that provides both incremental loading and multi-tasking [11]. Nevertheless, in most cases, DSPs are not used with an operating system. As a result, the level of portability of the code it runs is decreased.

3.2.3 Mapping FPGAs

Since most FPGAs don't provide support for incremental loading, they are usually represented as SCA LoadableDevices. But, it is possible to use FPGAs as ExecutableDevices. The Xilinx Virtex FPGAs support incremental loading through a concept called partial reconfiguration. It allows the FPGA to be partially reprogrammed during runtime without rebooting. ISR Technologies has used a Xilinx FPGA mapped as an ExecutableDevice [12]. Also, even if FPGAs don't come with an operating system, they natively support the concept of multi-tasking through parallelism.

4. CONCLUSION

In this paper we have outlined different aspects that play a role in software reuse, mainly associated with application portability. We have described how the SCA specification enhances application portability. We have also described characteristics of the different computational elements being used in SDR platforms, and how they are traditionally mapped to SCA containers. By identifying the key concepts that differentiate SCA containers (LoadableDevices and ExecutableDevices), we have described the limitations that LoadableDevices pose for SCA application portability. Finally, we have provided guidelines of what it is required for computational elements traditionally mapped as LoadableDevices, to be mapped as ExecutableDevices instead.

5. REFERENCES

[1] JPEO JTRS, "SCA APPLICATION ENVIRONMENT PROFILE", http://sca.jpeojtrs.mil/, May 2006.

[2] "Minimum CORBA Specification version 1.0: OMG Document formal/02-08-01", http://www.omg.org, August 2006.

[3] JPEO JTRS, "Joint Tactical Radio System (JTRS) Standard Modem Hardware Abstraction Layer Application Program Interface (API)", http://sca.jpeojtrs.mil/, May 2007.

[4] L. Wirbel, "SDR looks to new middleware", DSP Design Line, January 2007.

[5] J. Jacob, M. Ulm, "CORBA for FPGAs: Tying together GPPs, DSPs, and FPGAs", http://DSP-FPGA.com.

[6] JPEO JTRS, "Introducing the JTRS Public APIs", http://sca.jpeojtrs.mil/, May 2007.

[7] JPEO JTRS, "Software Communications Architecture Specification", http://sca.jpeojtrs.mil/, May 2006.

[8] F. Lévesque, C. Auger, S. Bernier, H. Latour, "Jtrs Sca: Connecting Software Components", SDRF'03 Technical Conference, Orlando, November 2003.

[9] J. R. Levine, "Linkers and Loaders", Morgan Kaufmann Press, January 2000.

[10] "IEEE Std 1003.1, 2004 Edition", http://www.unix.org.

[11] Talon™ DSP RTOS, http:// www.blackhawk-dsp.com.

[12] J. Belzile, "Putting it all together – Objectives and Challenges", SDRF'05 Technical Conference, 2005.