SCA-based Component Framework for Software Defined Radio*

Saehwa Kim, Jamison Masse, Seongsoo Hong, and Naehyuck Chang†
School of Electrical Engineering and Computer Science
Seoul National University, Seoul 151-742, Korea
{ksaehwa, jamison, sshong}@redwood.snu.ac.kr, naehyuck@snu.ac.kr

Abstract

SCA (Software Communication Architecture), which has been adopted as a SDR (Software Defined Radio) Forum standard, provides a framework that successfully exploits common design patterns of embedded systems software. However, the SCA is inadequate as a component framework since it does not explicitly specify (1) a component model that defines how to express a component interface and how to implement it, (2) a package model that defines what and how to package in deployment units, and (3) a deployment model that defines the deployment environment and deployment process. In this paper, we propose a SCA-based component framework for SDR. Specifically, we present (1) a component model that defines a component as a specialized CORBA object that implements object management functionality, (2) a package model exploiting the existing XML descriptors of the SCA, and (3) a deployment model that defines a SCA-based deployment environment with a boot-up process to restore the deployment state and a deployment process supporting lazy application instantiation and dynamic component replacement.

1. Introduction

Component software technology for maximizing software reuse is greatly helpful to overcome the extreme complexity of embedded software and reduce time-to-market. The component software technology is aimed at creating new software systems through the combination of deployable software, as opposed to ground-up development. We adopt the OMG's (Object Management Group) [1] definition of a component: a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces [2]. Components are typically some form of shared library and, depending on the deployment environment, could be distributed as binaries, byte code, or even source files written in a scripting language.

The SDR (Software Defined Radio) forum [3] has adopted SCA (Software Communication Architecture) [4] of the JTRS (Joint Tactical Radio System) [5] as the standard software structure of SDR embedded systems. The SCA provides a framework that successfully exploits common design patterns of embedded systems software, which is typically composed of device control programs and application programs. The SCA also provides a flexible environment for integrating heterogeneous hardware and software written in various languages by adopting CORBA (Common Object Request Broker Architecture) [6] as its base middleware. However, the current SCA is inadequate as a component framework since it does not explicitly specify (1) a component model that defines how to express a component interface and how to implement it, (2) a package model that defines what is in a deployment package and how those contents are packaged, and (3) a deployment model that defines the deployment environment and deployment process.

In this paper, we propose a SCA-based component framework for the SDR. Specifically, we present (1) a component model that defines a common component CORBA interface that supports object management functionality including connecting contained objects while isolating functions that are not a part of the application domain, (2) a package model that exploits XML descriptors defined in the SCA domain profile, and (3) a deployment model that defines a deployment environment exploiting the SCA core framework interfaces, a boot-up process to restore the deployment state exploiting the SCA domain profile, and deployment process that supports lazy application instantiation and dynamic replacement of application components. An application can be selectively instantiated after its installation through our lazy application instantiation. An application also can be dynamically upgraded through our dynamic component replacement mechanism.

1.1. Related Work

CCM (CORBA Component Model) [7], EJB (Enterprise Java Beans) component model [8], and DCOM (Distributed Component Object Model) [9] are the most widely available component frameworks for distributed object computing. Among them, the CCM is the most recent standard and supports more advanced technology encompassing the others. Since the CCM can interoperate with both the EJB and the DCOM and since the SCA is based on CORBA technology, we will use the CCM exclusively to compare with our research.

However, the CCM cannot be directly applied to SDR handheld devices since the CCM is focused only on server side applications that are installed in general purpose systems [10] while SDR systems accommodate stand-alone wireless applications, as opposed to stand-by server applications.

Imposing features from the CCM, such as containers, home

* The work reported in this paper is supported by the Korea Science and Engineering Foundation (KOSEF) grant R01-1999-00206.
† School of Electrical Engineering and Computer Science, Seoul National University, Seoul 151-742, Korea.

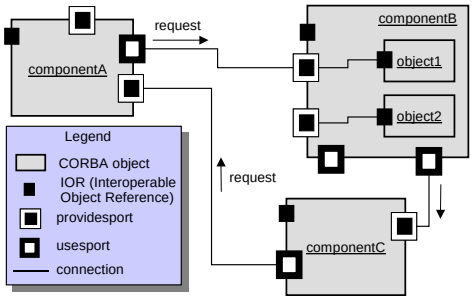

Figure 1. Connection of components via ports in our component model.

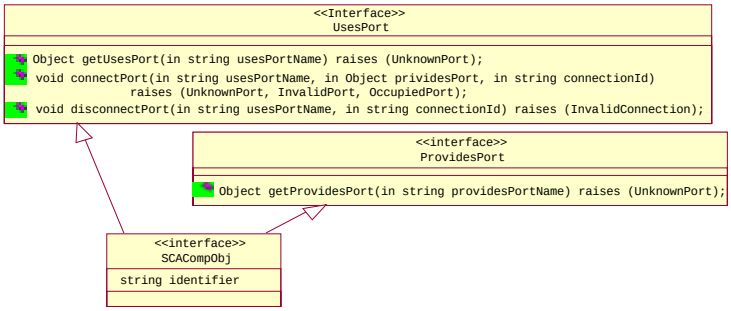

Figure 2. SACompObject interface.

executers, and reflection runs counter to the need for lightweight software in handheld devices. We should also consider how to deploy hardware device configuration information related to the deployment of software components. Additionally, we also consider how to restore the deployment state when an SDR system boots up since SDR systems may exist that have limited non-volatile storages that cannot support full check pointing or backup of run-time memory state.

The remainder of the paper is as follows. We provide an overview of the SCA in Section 2. In Sections 3, 4, and 5, we present our component framework that is composed of a component model, package model, and deployment model to complement SCA and complete component framework. We conclude this paper in Section 6.

2. Overview of the SCA

The SCA structure is composed of an application layer and an operation environment (OE) layer. The OE is also divided into RTOS, CORBA, and core framework layers where RTOS and CORBA are COTS (Commercial Off-The-Shelf) products. Since the SCA uses the CORBA middleware, application programs are basically composed of CORBA objects that conform to the SCA core framework. The SCA core framework is composed of the specification of interfaces and a domain profile. A domain profile is composed of XML descriptor files that describe the hardware and software configuration information of a SCA system domain.

3. Proposed Component Model

In this section, we present our component model that describes how to define interfaces of components and its rationale. For this, we will first explain our approach, which is isolating object-management functionality. Then, we will explain the notion of ports, which are used to describe an interface of a component. After that, we will present our common component interface.

3.1. Approach: Isolating Object-Management Functionality

A component object in our component model, an instantiation of a component interface, isolates object-management functionality and does not contribute any functionality to an application domain. A component object is responsible (1) internally for managing life cycles of its contained objects and (2) externally only for connecting its contained objects with objects in other components. To enforce this approach, we let a component interface in our component model compose CORBA object interfaces but prevent it from inheriting any CORBA object interface. This approach follows the basic principal of object-orientation that encapsulates behaviors according to divisible functions. Our approach is also justified when characteristics of embedded systems software are considered where objects are typically statically connected in the deployment phase and maintain their connections composing stand-alone applications.

3.2. Ports

The term port is used in the CCM and also in the SCA domain profile to mean a named connection point through which components interoperate. We explain the notion of ports that are needed in our component model to declare component interfaces and describe connection information between components.

Our component model supports one-way binary communication via *provides* ports and *uses* ports. A *provides* port of a component is the means to retrieve an object reference for a server object contained in the component. A *uses* port of a component is the means to retrieve an object reference for a proxy object connected with a server object contained in another component. Figure 1 shows conceptually the connection of components via ports in our component model.

3.3. Common Component Interface

We provide a standardized common component interface so that a deployment tool can construct software by composing components. In our component model, component interfaces are declared via IDL (IDL2) and Software Component Descriptor (SCD) in the SCA domain profile. Specifically, the IDL declares only a component name and the SCD declares the port information of a component. We propose `SACompObj` interface that will be used as a common component interface as shown in Figure 2. The declaration of component interface in IDL is done as follows by only declaring the component name information.

```
interface component_name:SACompObject {};
```

That is, interfaces inheriting only `SACompObj` without a body are component interfaces. As described before, the `SACompObj` is an interface whose function is to connect objects

contained in components. The SCACompObj interface implements UsesPort and ProvidesPort interfaces as shown in Figure 2. The ProvidesPort interface provides getter operations for *provides* ports and the UsesPort interface provides setter and getter operations for *uses* ports. The identifier attribute is used as a unique identifier for instances of a component interface. Programmers should implement operations of SCACompObj for each component interface according to their set of ports.

The current SCA inconsistently makes use of the concept of ports, differing between the SCA core framework interfaces and the SCA domain profile. The view of the SCA domain profile is in accordance with that of our component model. We consider ports simply named connection points between components, while the SCA core framework interfaces consider ports actual objects providing connection functionality. In our component model, the common component interface (SCACompObj) provides connection functionality, rather than using port objects to facilitate connection. Through our component model, the notion of ports becomes consistent through the whole framework. The SCACompObj makes both the Port and PortSupplier interfaces of the current SCA for connecting objects obsolete.

4. Proposed Package Model

Although a component file is the basic unit of software composition, it cannot be the unit of deployment. The deployment unit for composing component software should contain not only component files but also files like XML descriptors describing deployment information. We adopt the term *package* from the CCM for the deployment unit, which is a ZIP file format assembly of multiple files. We classify packages into (1) *component packages* that contain only one component type and (2) *component assembly packages* that contain multiple component types.

Since the SCA domain profile provides well-defined descriptors and association relationships among descriptors, it is straightforward to determine which descriptors should be packaged together in each package. A component package contains one top-level Software Package Descriptor (SPD) as well as all the other descriptors the SPD requires. A component assembly package contains either one Software Assembly Descriptor (SAD) or one Device Configuration Descriptor (DCD) as well as all the other descriptors they require.

On the other hand, packages cannot contain arbitrary combination of SCA core framework interfaces since there are precedence and collocation constraints on installing objects. Therefore, we need to determine which interfaces should be and cannot collocated in a package and categorize the types of packages. Since this is tightly coupled with the deployment model, we describe it in the next section.

5. Proposed Deployment Model

In this section, we describe the roles the participants play in the deployment process and how they interoperate. Our deployment model presents (1) a complete deployment environment, (2) a system boot-up process to restore the deployment state, and (3) a deployment process supporting lazy application instantiation and dynamic replacement of application components.

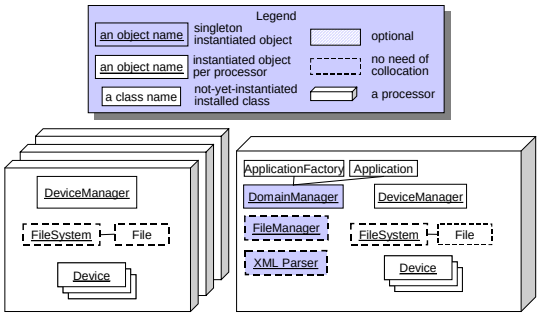

Figure 3. Deployment environment.

5.1. Deployment Environment

Figure 3 shows our basic deployment environment. Installed classes and activated objects in the deployment environment are primarily composed of the SCA core framework but also include additional objects such as an XML parser. To set up this deployment environment, we need a pre-installed component package supporting DomainManager and component assembly packages described with DCDs. This deployment environment implicitly dictates which interfaces should be and should not be collocated in a package and categorizes package types.

Each processor contains logical device objects; one logical device for the processor itself and others for hardware devices the processor manages. These proxy logical devices can act as cross loaders by managing accessible image files, loading images to target devices, and letting cross-loaded images execute on their target devices. As such, the role of logical device objects is similar to that of hardware managers in [11].

5.2. Boot-up Process

The boot-up process, the restoration of the deployment state, is mainly composed of (1) setting up the deployment environment and (2) activating applications that were executing when the system was shut down. Our boot-up process requires elements of domain profiles reside in non-volatile storage. Specifically, a DCD should be pre-installed in each node and one node should have DMD pre-installed where the DomainManager will reside. In addition, SADs also should be pre-installed in the proper nodes. Additionally, applications must be able to specify if they should be instantiated immediately after boot-up if they were instantiated at the point of shut-down. For this, we added to SAD two XML elements *instantiated* and *restore*. The element *instantiated* is set to true when its application is instantiated and is set to false when the application is shut down. The element *restore* is a static value representing whether the instantiation of its application would be restored or not. Under these conditions, each node executes a boot-up procedure by exploiting the domain profile information. The boot-up procedure is composed of (1) DomainManager configuration using DMD, (2) DeviceManager configuration using the DCD, and (3) Application configuration using SADs.

5.3. Deployment Process

The boot-up process, the restoration of the deployment state, is mainly composed of (1) setting up the deployment environment and (2) activating applications that were executing when the system was shut down.

Our deployment model provides different deployment processes according to the component types being deployed. Specifically, our deployment process supports lazy application instantiation where an application may not be activated immediately after it is installed but it may be activated selectively afterward. But components that implement control and service interfaces in the SCA framework that are always instantiated such as Device are instantiated immediately after installation. Note that the CCM deployment process, intended for server systems, installed components are instantiated immediately after installation. However in our framework for SDR embedded systems, multiple waveform applications are installed together and they are selectively instantiated on a case-by-case basis.

Application component upgrades are facilitated through a `replace()` operation we have added to `DomainManager`. Upgrading an instantiated application can be done in two ways: run-time or lazy upgrade. The lazy upgrade uninstalls the package of the target component, installs the new package, and updates the corresponding SAD descriptor. The upgrade takes effect only after the application is re-instantiated. The run-time upgrade follows the procedure of the lazy upgrade but after updating the SAD stops all the resource objects within the target application, disconnects connections between objects of the target component and others, instantiates the replaced component, restores connections related with the newly replaced component. The choice of such an upgrade strategy is dependent on the properties of components or applications. For this reason we have added to both the SAD and the SCD an XML element *`upgradetype`* which can be either *`runtime`* or *`lazy`*. Only when both values are *`runtime`*, the run-time upgrade is performed.

As such, installation process and instantiation process for application components are explicitly separated. This is from consideration of the characteristics of the SDR embedded systems, where multiple waveform applications are installed together and they are selectively instantiated on a case-by-case basis.

6. Conclusion

We have presented a SCA-based software framework supporting dynamic deployment of SDR components that is composed of a component model, a package model, and a deployment model. Our component model provides a common component interface for connecting objects contained in its component. Components are defined as specialized CORBA objects supporting object management functionality, isolating functions that are not applied to an application domain. Our package model provides deployment unit packaging that exploits the SCA domain profile. Finally, our deployment model provides (1) a deployment environment based on SCA core framework interfaces, (2) a boot-up process to restore the deployment state, and (3) a deployment process supporting lazy application instantiation and dynamic component replacement.

The main contributions of the paper are three fold. First, we have proposed the component model specialized to embedded systems with consistent notion of ports. Second, we have proposed the deployment model based on the current SDR software standard, complementing it. Finally, we have presented the component framework specialized for embedded systems addressing the characteristics of embedded systems applications with static connection management of components, the boot-up process to restore deployment state, and lazy application instantiation policies.

We are currently developing deployment tools for supporting the proposed framework to show its utility. Designing a lightweight container for embedded systems that supports such a component model also seems promising. Future considerations also include specifying non-functional QoS aspect of components such as response time, fault-tolerance, security and tools for evaluating them.

References

1. Object Management Group (OMG), http://www.omg.org.
2. Unified Modeling Language Specification Version 1.4 Appendix B - Glossary, Object Management Group, September 2001.
3. Software Defined Radio (SDR) Forum, http://www.sdrform.org.
4. Software Communications Architecture (SCA) Specification MSRC-5000SCA V2.2, Joint Tactical Radio Systems, November 17, 2001, Available at http://www.jtrs.saalt.army.mil/SCA/SCA.html.
5. Joint Tactical Radio System (JTRS), http://www.jtrs.saalt.army.mil/.
6. The Common Object Request Broker: Architecture and Specification, Version 3.0, Object Management Group, June 2002.
7. CORBA Component Model Version 3.0, Object Management Group, June 2002.
8. Enterprise JavaBeans Technology, Sun Microsystems, Inc, http://java.sun.com/products/ejb/.
9. The Distributed Component Object Model (DCOM), Microsoft Corporation, http://www.microsoft.com/com/tech/DCOM.asp.
10. W. Emmerich and N. Kaveh, Component Technologies: Java Beans, COM, CORBA, RMI, EJB and the CORBA Component Model, In Proceedings of International Conference on Software Engineering, pp. 691-692, 2002.
11. Benjamin H. Wang, Pangan Ting, S. Charles Tsao, Hung-Lin Chou, and Nanson Huang, Integration of System Software and SDR Hardware Platforms, SDRF-01-I-0052-V0.00, Software Defined Radio Forum Contribution, August 2001.