Dynamic Deployment of Software Defined Radio Components for Mobile Wireless Internet Applications[*]

Saehwa Kim, Jamison Masse, and Seongsoo Hong

School of Electrical Engineering and Computer Science
Seoul National University, Seoul 151-742, Korea
{*ksaehwa, jamison, sshong*}*@redwood.snu.ac.kr*

Abstract. Software Defined Radio (SDR) is a key enabling technology for mobile wireless Internet. SDR represents unique opportunity to provide Internet connectivity to handheld devices over a limitless range of communication standards. For SDR systems to realize their full potential, they must be reconfigurable through the dynamic deployment of SDR components. However, the current SDR Forum standard, Software Communication Architecture (SCA), is insufficient in this respect since it fails to provide a complete component framework. 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, a boot-up process to restore deployment state, and a deployment process supporting lazy application instantiation and dynamic component replacement. Frameworks that incorporate these improvements will meet the dynamic software deployment needs of next-generation wireless Internet applications.

1 Introduction

Convergence of Internet and wireless communication technologies is creating huge demand for access to Internet services from wireless handheld devices. Software Defined Radio (SDR) is a key enabling technology for mobile wireless Internet [1]. SDR is a completely configurable radio that can be programmed in software. SDR offers mobile wireless Internet users the ability to use a single terminal to access a wide range of wireless services and features by ensuring that handheld devices are radio-agnostic. As new wireless standards emerge, SDR terminals represent an investment for consumers and carriers that are protected from the volatility of a still emerging set of technologies. Not only does SDR lower the barriers to entry for consumers and providers of wireless Internet services but opens the field to the improvement of standards with fewer burdens of legacy support. This means that wireless Internet facilitated by SDR offers the opportunity for not just the evolution of wireless standards but holds the potential to allow the introduction of completely new standards with lit-

[*] The work reported in this paper is supported in part by MOST (Ministry of Science and Technology) under the National Research Laboratory (NRL) grant 2000-N-NL-01-C-136.

tle expense or inconvenience. A wireless Internet built on SDR is a network constructed with a truly upgradable end-to-end infrastructure.

For SDR systems to realize their full potential, they must be reconfigurable through the dynamic deployment of SDR components. This can be achieved by component software technology that is aimed at creating new software systems through the combination of deployable software (components), as opposed to ground-up development. We adopt the OMG's (Object Management Group) [2] definition of a component: a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces [3]. 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 forum [1] has adopted Software Communication Architecture (SCA) [4] of the Joint Tactical Radio System (JTRS) [5] as the standard software structure of SDR systems. The SCA structure is composed of an application layer and an operation environment (OE) layer. The OE is also divided into RTOS, CORBA (Common Object Request Broker Architecture) [6], and core framework layers where 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.

However, the current SCA is insufficient, not properly specifying how the dynamic deployment of SDR components for mobile wireless Internet applications can be carried out, failing to provide a complete component framework. 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 is in a deployment package and how those contents are packaged, and (3) a deployment model that defines the deployment environment and deployment process. Although the SCA is based on CORBA technology, CORBA Component Model (CCM) [7] 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 [8] while SDR systems accommodate stand-alone wireless applications, as opposed to stand-by server applications.

In this paper, we propose a SCA-based component framework for dynamic deployment of SDR components for mobile wireless Internet applications. Specifically, we present (1) a component model that defines a common component CORBA interface that supports object management functionality 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, a boot-up process to restore deployment state, and deployment process that supports lazy application instantiation and dynamic replacement of application components.

The remainder of the paper is as follows. In Sections 2, 3, and 4 we present our component model, package model, and deployment model respectively. These together complement SCA and form a complete component framework. We conclude this paper in Section 5.

2 Component Model

In this section, we present our component model that describes how to define interfaces of components and its rationale. Our approach is composed of isolating object-management functionality and providing consistent notion of ports.

2.1 Isolating Object-Management Functionality

In the CORBA environment, a distributed object-computing system, a component in our framework has the ability to express itself as a run-time CORBA object with its own interface. Then, we need to provide a standardized common component interface so that a deployment tool can construct software by composing components. In our component model, a component at run-time can be viewed as an aggregation of objects. The contained objects' functionality is part of one or many applications. Our components also provide functionality unrelated to any application domain, object management functionality. Object management functionality includes operations for (1) internally managing object life cycles and (2) externally connecting objects. Although the object-management functionality does not contribute to the application domain itself, it is essential to enable scalable server object systems. In fact, CORBA programming is essentially the wrapping of proper scalable object management functionality around application objects.

This approach follows the basic principal of object-orientation that encapsulates behaviors according to divisible functions. Our approach is also justified when characteristics of software for handheld devices are considered where objects are statically connected in the deployment phase and maintain their connections for the lifetime of their installation.

2.2 Providing Consistent Notion 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 term port is used in the CCM and also in the SCA domain profile to stand for a named connection point through which components interoperate. 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.

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.

In our component model, component interfaces are declared via IDL (IDL2) and Software Package Descriptor (SPD). Specifically, the IDL declares only a component name and the SPD declares the port information of a component. We propose SCAComponentObject interface that is used as a common component. The declaration of a component interface in IDL is done as follows by only declaring the component name information.

```
interface component_name:SCAComponentObject {};
```

That is, interfaces inheriting only SCAComponentObject without a body are component interfaces. The SCAComponentObject interface provides getter operations for *provides* ports and setter and getter operations for *uses* ports. The SCAComponentObject also has an attribute *identifier* that is used as a unique identifier for instances of a component interface. Programmers should implement operations of SCAComponentObject for each component interface according to their set of ports.
In the current SCA, some objects contained in a component should implement Port interface to connect *uses* ports contained in the component. Moreover, contained objects that will be exported as *provides* ports should also implement connection operation of PortSupplier interface that is inherited by Resource interface. In our component model, the common component interface (SCAComponentObject) provides connection functionality, rather than using port objects to facilitate connection. The SCAComponentObject makes both the Port and PortSupplier interfaces of the current SCA for connecting objects obsolete.

3 Packaging 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. Since a component is described with Software Package Descriptor (SPD), a component package contains one top-level 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.

4 Deployment Model

In this section, we describe our deployment model that is composed of (1) a complete deployment environment, (2) a system boot-up process to restore deployment state, and (3) a deployment process supporting lazy application instantiation and dynamic replacement of application components.

4.1 Deployment Environment

A DomainManager object exists in one processor in a system as a singleton. Application and ApplicationFactory interfaces should be collocated with DomainManager since the latter directly uses the former. Additionally, a FileManager object and an

XML parser objects should exist as singletons. Note that these objects need not be collocated with the DomainManager object. Each processor also 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 [9]. Each processor also accommodates one DeviceManager object to manage logical devices and service objects instantiated in it, each of which require a FileSystem object. To set up this deployment environment, we need a pre-installed component package supporting DomainManager and component assembly packages described with DCDs.

4.2 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 whether they should be instantiated immediately after boot-up if they were instantiated at shutdown. 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.

4.3 Deployment Process

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 handheld devices, multiple stand-alone applications may be installed together but can be selectively instantiated.
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 up-

grade follows the same procedure as the lazy upgrade except that after updating the SAD, the run-time upgrade (1) stops all the resource objects within the target application, (2) disconnects the target component connections, (3) instantiates the replacement component, and (4) restores the previous connections. 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 SPD an XML element *upgradetype* that can be either *runtime* or *lazy*. Only when both values are *runtime*, the run-time upgrade is performed.

5 Conclusion

We have presented a SCA-based component framework supporting dynamic deployment of SDR components for mobile wireless Internet applications. The main contributions of the paper are three fold. First, we have proposed the component model specialized to handheld 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 handheld embedded systems addressing the characteristics of handheld devices applications with static connection management of components, the need for a boot up process that properly restores deployment state, and lazy application instantiation policies and methods. Frameworks that incorporate these improvements will meet the dynamic software deployment needs of next-generation wireless Internet applications. There are several future research directions. Designing a lightweight container for handheld embedded systems that supports such a component model 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. Software Defined Radio (SDR) Forum, http://www.sdrform.org.
2. Object Management Group (OMG), http://www.omg.org.
3. Unified Modeling Language Specification Version 1.4 Appendix B - Glossary, Object Management Group, September 2001.
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. 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.
9. 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.