Extending Software Communications Architecture for QoS Support in SDR Signal Processing

Jaesoo Lee, Jiyong Park, Seunghyun Han, and Seongsoo Hong
School of Electrical Engineering and Computer Science, Seoul National University, Seoul 151-742, South Korea {jslee, parkjy, shhan, sshong}@redwood.snu.ac.kr

Abstract

The Software Communications Architecture (SCA) is middleware for providing component interfaces and dynamic reconfigurability to software defined radio (SDR) modems and it is widely adopted by the SDR forum. Although its main application domains include real-time signal processing, SCA lacks QoS support. We thus propose a QoS-enabled extension of SCA called Q-SCA. Our extended architecture incorporates a new waveform model, extended domain profiles that describe QoS requirements, and a modified application instantiation process that supports both admission control and resource allocation. We show the viability of Q-SCA through full implementation and experiments.

1. Introduction

Software Defined Radio (SDR) is characterized by a modem that can be dynamically reconfigured by dynamically loadable software. Thus, it can easily conform to a broad range of wireless protocols in order to access various mobile services and devices. To accomplish this, the SDR modem itself is often composed of heterogeneous hardware components which interact via common communication buses. The Joint Tactical Radio System (JTRS) has created a middleware specification called the Software Communications Architecture (SCA) [1] to support communication between these devices.

The SCA supports SDR developers by providing them with two types of interfaces. The first type enables communication and interchangeability between various signal processing components. The second type, known as the Operating Environment (OE), supports the dynamic reconfiguration of SDR applications. As illustrated in Figure 1, the OE incorporates a common-of-the-shelf (COTS) software layer comprised of CORBA [2] middleware and a POSIX [5] compliant operating system. Utilizing both of these is a set of API interfaces and system profiles known as the core framework (CF).

The core framework is composed of a domain profile and standard interfaces. The domain profile describes the hardware and software configuration of a SCA system domain and is composed of XML descriptor files. The CF interfaces are composed of interfaces for (1) application components, (2) domain management, and (3) services such as logging. The SCA middleware is therefore the implementation of the domain management and service part of the SCA core framework. Wireless application developers can dynamically deploy and manage the life cycle of their software using the SCA middleware.

While SCA enables SDR components to communicate with each other and provides for dynamic reconfigurability, it does not provide developers with any means of specifying or enforcing the QoS requirements of their applications. Because SCA is often used in real-time signal processing applications, we have created Q-SCA, a QoS-enabled extension of SCA. Q-SCA is supported by a new model for

The work reported in this paper was supported in part by MOST (Ministry of Science and Technology) under the National Research Laboratory (NRL) grant 2000-N-01-C-136, by “System IC 2010” project of Korea Ministry of Science and Technology and Ministry of Commerce, Industry and Energy, and by “Embedded Component Technology and Standardization for URC” project of Ministry of Information and Communication.
describing SDR waveforms that describes both QoS requirements and resource requirements.

The remainder of this paper is organized as follows. Section 2 presents the new waveform model. Section 3 introduces the QoS descriptors in the model, the admission control and resource allocation mechanisms, and the modified application instantiation process which uses those mechanisms to guarantee QoS requirements. In Section 4 we report on the run-time performance of our Q-SCA implementation and in Section 5 the paper is concluded.

2. Waveform Model

In this section, we focus on our model which is capable of specifying a wide spectrum of digital signal processing applications. The underlying mechanisms that are responsible for managing and allocating resources will be explained subsequently in Section 4.

In our application model, as in many other embedded system models, we use a graphical language with hierarchical abstraction. More specifically, we adopt a synchronous dataflow (SDF) model [6] that has been widely used in signal processing applications. In the SDF model, as with Kahn process networks [7][8] and the U.S. Navy’s Processing Graph Method (PGM) [9], a program is specified by a directed graph $G(P,E)$. In this graph, vertices represent computational processes, and edges represent flows of data from one process to the next. Specifically, information flows along unidirectional FIFO channels where writes to the channel are non-blocking and reads are blocking. Unlike the PGM, however, our model is able to capture the dependencies of a process on deployable hardware. We formally define our graph model.

Given a process graph $G(P,E)$, $P=\{\sigma_1,\ldots,\sigma_n\}$ is a set of processes and $E \subseteq P \times P$ is a set of directed edges such that $\sigma_i \rightarrow \sigma_j$ denotes a data dependency of $\sigma_j$ on $\sigma_i$. For edge $e_i$, $\text{source}(e_i)$ and $\text{sink}(e_i)$ denote its source and sink processes. A process has a set of typed ports with which edges are connected. A port of $\sigma_i$ with which edge $e_i$ is connected is denoted by $p(\sigma_i, e_i)$. Each port is annotated with the number of tokens $n(p)$ either produced or consumed depending on whether the port is an output port (typically a uses port in SCA terms) or an input port (provides port). A process represents a computational unit that maps one or more input streams (from input ports) to one or more output streams (to output ports), and therefore is associated with a certain amount of time $C(\sigma_i)$ required for the computation. It can be the maximum propagation delay per unit frequency (e.g., 1MHz) if it is a piece of FPGA code or the worst-case instruction cycles if it is binary executable on a general purpose processor or DSP.

Our waveform model allows programmers to specify three types of QoS requirements and two types of resource requirements. The QoS requirements include the relative execution rate of a process, the absolute sampling rate of an input process, and the maximum latency of an output process. The resource requirements include matching properties and allocatable properties. We define these requirements.

For edge $e_i$, the number of tokens produced by each invocation of the source process is represented as $n(p(\text{source}(e_i), e_i))$, or shortly $r_{\text{source}}(e_i)$. The number of tokens consumed by the sink process is similarly defined as $n(p(\text{sink}(e_i), e_i))$, or shortly $r_{\text{sink}}(e_i)$. If edge $e_i$ exists between any two processes $\sigma_i$ and $\sigma_j$, then the relative invocation rate of $\sigma_i$ to $\sigma_j$ is determined by (1).

$$r_{\text{max}}(e_i) \leq r_{\text{sink}}(e_i)$$

Input process set $S \subseteq P$ denotes source processes in $P$ that have no incoming edges. Typically, input processes are sampling processes or modern processes in SDR systems. The execution rate $R(p)$ of input process $\sigma_i$ is explicitly specified, and thus determines the execution rates of subsequent processes in the data dependency chain. The execution rate of process $\sigma_j$ with multiple incoming edges is determined by taking the maximum possible rate as given by (2) where $\text{Inedges($\sigma_i$)}$ denotes the incoming edge set of $\sigma_i$ and $\text{Outedges($\sigma_i$)}$ denotes the outgoing edge set of $\sigma_i$ accordingly.

$$\max_{e_i \in \text{Inedges($\sigma_i$)}} \left\{ \frac{r_{\text{source}}(e_i)}{r_{\text{sink}}(e_i)} \right\} \leq R(\text{source}(e_i))$$

Output process set $Q \subseteq P$ denotes sink processes in $P$ that have no outgoing edges. Each sink process is associated with maximum latency requirements $D(\sigma_i)$ as described in the following:

$$\sum_{\sigma_i \in Q} D(\sigma_i) \leq D(\text{output}(\xi))$$

where a critical path represents an acyclic path from an input process to an output process, $D(\sigma_i)$ represents the worst case response time of $\sigma_i$ enforced by underlying resource allocation mechanisms, and $D(\text{output}(\xi))$ is an output node in path $\xi$.

Waveform applications will receive a reserved set of resources to meet imposed QoS requirements. It is important for them to secure a sufficient amount of resources with appropriate types. We consider resources hosted by a loadable and executable device such as CPU cycles and memory. The loadable and executable devices in a system are specified with a set of processing elements $PE=\{PE_j, \ldots, PE_n\}$. Each $PE_j$ is
3. Components of Q-SCA

We extend SCA to incorporate the waveform model described in the previous section. More specifically, (1) we extend domain profiles to allow for resource and QoS requirement specification; (2) we add services providing admission control and resource allocation to the SCA core framework; and (3) we extend the software communication bus based on the real-time ORB following the RT-CORBA v.2.0 specification [3] for the enforcement of the resource allocation result. These extensions are transparently integrated into the application instantiation process. Since RT-CORBA provides static/dynamic priority scheduling disciplines and prioritized communications in addition to the features provided by CORBA, we exploit these features for the admission control and resource allocation of core framework components to meet QoS requirements as described in the application’s domain profile.

3.1. QoS Descriptors for Extended Domain Profiles

Our extended SCA (Q-SCA) allows application developers to achieve desired QoS guarantees by simply specifying their requirements in extended domain profiles. In doing so, application developers are responsible for describing their application structure and participating components in a dedicated XML descriptor called the software assembly descriptor (SAD). Since a legacy SCA SAD describes only application components, the Q-SCA core framework should provide a mechanism to derive the scheduling parameters from an application’s QoS requirements. This problem is well studied in most QoS middleware systems such as 2K⁴S [10], Agilos [11], and QuO [12], and thus any result from the literature [13] [14] [15] [16] [17] [18] can be used. After scheduling parameters are derived, Q-SCA should deliver them to each application component. For this purpose, the Q-SCA core framework uses one of the existing base application interfaces, PropertySet, to configure resources.

For admission control and resource allocation, we add the ResourceAllocator component. Its interface definition is given in Figure 2. It keeps track of the availability of resources in the system and stores it in the deviceCapacities attribute. Upon a request for the
creation of an application, it checks the schedulability of the system for the application and assigns a loadable/executable device to each component of the application. It performs resource allocation via the createAssignments operation. At run-time, the RT-CORBA scheduling service is responsible for the enforcement of resource allocations.

3.3. Modified Application Initiation Process

In designing Q-SCA, it is necessary to modify the application initiation process because it involves reading QoS parameters in the extended domain profiles, performing admission control, and reserving resources. Figure 3 depicts the modified application initiation process. It shows that such extensions are transparently integrated into the original SCA. We elaborate on the components shown in the figure. An application in the SCA domain is created by the ApplicationFactory component, which belongs to the SCA domain management part and is in charge of instantiating a specified type of application. When ApplicationFactory instantiates an application in Q-SCA, it ascertains its QoS requirements from the domain profile and then passes the information to the ResourceAllocator. This action corresponds to step 3 in Figure 3. If the application is admissible, the ResourceAllocator generates the resource allocation for the application based on current resource availability.

The ApplicationFactory component performs the resource allocation generated by ResourceAllocator in the following steps: it deploys all components onto the loadable/executable devices as designated in the plan (in step 4 in Figure 3), and then it delivers scheduling parameters to each component (in step 5). To accept the scheduling parameters from the ApplicationFactory, application components should implement the PropertySet interface (in step 11).

4. Performance Evaluation

In evaluating Q-SCA, it is important to quantify its run-time performance since it is built upon the COTS software layer containing the RT-CORBA ORB. To do so, we have completely implemented Q-SCA and constructed an Ethernet-based telephony application which exchanges voice data over an Ethernet connection using Q-SCA components and interfaces. We have conducted experiments to measure message propagation delay between application components. In this section, we report on our run-time performance evaluation of Q-SCA.

4.1. Experimental Setup

Our Q-SCA was implemented using Linux v. 2.4.20 and the TAO [4] real-time ORB 1.3.1 on a hardware platform consisting of two embedded CPU boards. Each of them was equipped with a 1 GHz Intel-compatible VIA processor and 256 Mbytes DDR SDRAM. They were connected via 10 Mbps Ethernet interfaces. One of the boards was also connected to the Internet via another 10 Mbps Ethernet interface. The other has a soundcard with a microphone and a speaker connected. Although this configuration is not as complicated as actual wireless handsets, it has all the components required to measure the performance of Q-SCA without incurring the various secondary effects that would otherwise be seen in a wireless environment.

The Ethernet-based telephony application was constructed with two Q-SCA devices NetDevice and PCMAudio as shown in Figure 4. These devices abstract the Internet and the soundcard, respectively. The application maintains two dataflows simultaneously: an upstream dataflow from PCMAudio to NetDevice and a downstream dataflow in the opposite direction. For the upstream dataflow, audio samples captured by the PCMAudio component from the microphone are compressed and encrypted by
Compressor and Cipher, respectively, and finally transmitted by NetDevice to the peer SDR device. Similarly, for the downstream dataflow, voice data received by NetDevice is decrypted and decompressed by Decipher and Decompressor, respectively, and finally played by PCMAudio. As already explained in Section 3.1 and as can be seen in Figure 4, the number of produced and consumed tokens, the worst case number of floating point operations, as well as the matching and allocation properties are described in the software package descriptor whereas the maximum execution rate (only for the input process, PCMAudio) and the maximum latency requirement (only for the output process, NetDevice) are described in the software assembly descriptor.

Each of the components comprising the application must be allocated with a specified amount of CPU resources for processing. For example, one invocation of Decipher takes 4 mega floating point operations in the worst case and consumes 16 tokens produced by the predecessor component, NetDevice, as seen in Figure 4. Both dataflows are expected to be initiated every 33.3 ms. Note that the Decipher and Cipher components need to be executed 4 times more often, that is (4, 33.3 ms), than the other components as can be derived from Equation (1). Note that they consume only 16 tokens per every invocation while their predecessors, NetDevice and Cipher respectively produce 64 tokens per invocation. There are also QoS requirements to be met: both dataflows have a 300 ms maximum latency requirement, meaning that the time taken for the total processing of any voice data should not exceed 300 ms.

### 4.2. Performance Evaluation Results

In order to quantify the run-time performance overhead incurred by Q-SCA, we measured the delay incurred by transferring voice data between two application components deployed on two different nodes, and compared them with the case where Q-SCA is not used. If Q-SCA is not present, we assume that TCP/IP is directly used instead of the RT-CORBA ORB. When Q-SCA is used, our ResourceAllocator allocates NetDevice, Decipher, and Decompressor to one node and PCMAudio, Compressor, and Cipher to the other node. We constructed this same configuration manually in the TCP/IP version of the application.

Figure 4 shows the distribution of latencies incurred to transfer voice data from PCMAudio to Compressor using TCP/IP and the RT-CORBA ORB. The PCMAudio and Compressor components are selected among the others since they are the components that our Q-SCA implementation deploys separately on different nodes. The average latency is about 70.5 ms when we use Q-SCA, while it is 65.0 ms when TCP/IP is used instead. Thus, the overhead incurred by Q-SCA is less than 10%. These results show that Q-SCA can provides applications with QoS guarantees with a relatively small overhead.
5. Conclusions

In this paper, we have presented Q-SCA, a fully implemented, QoS-enabled extension of SCA. Specifically, we have presented the Q-SCA waveform model that allows programmers to specify both QoS requirements and resource requirements. These QoS requirements include the relative execution rate of a process, the absolute sampling rate of an input process, and the maximum latency of an output process. The resource requirements include matching properties and allocatable properties. Furthermore, we have extended the domain profiles used by the SCA core framework so that developers may describe the requirements of their applications in XML form. Finally we have shown the efficacy and performance of Q-SCA by implementing and testing it in an internet telephony application.

We are looking to extend our research on Q-SCA by exploring the monitoring and adaptation capabilities found in the COTS layer to allow Q-SCA to dynamically adapt to changing requirements. Also, we are interested in applying Q-SCA to robotic applications. Because robots are often composed of a complex array of distributed hardware that is subject to replacement while running resource intensive audio and video applications, they represent the type of system which would benefit greatly from Q-SCA.

References


