SICE-ICASE International Joint Conference 2006
Oct. 18-21, 2006 in Bexco, Busan, Korea

The Robot Software Communications Architecture (RSCA): QoS-Aware Middleware for Networked Service Robots

Jonghun Yoo[1], Saehwa Kim[2], and Seongsoo Hong[3]

School of Electrical Engineering and Computer Science, Seoul National University, Seoul, Korea
(Tel: +82-2-880-8370; E-mail: {[1]jhyoo, [2]ksaehwa, [3]sshong}@redwood.snu.ac.kr)

Abstract: The ubiquitous robot companion (URC) project has been recently launched in Korea with an aim of putting networked service robots into practical use in residential environments by overcoming technical challenges of home service robots. Embedded middleware is surely one of such challenges since it has to deal with many critical and difficult problems such as real-time guarantees and software reconfigurability on a heterogeneous, distributed mechatronics system. In this paper, we adopt middleware called SCA from the software defined radio domain and extend it for use in URC robots. We call the end result Robot Software Communications Architecture (RSCA). The RSCA provides a standard operating environment for robot applications together with a framework that expedites the development of such applications. The operating environment is comprised of a real-time operating system, communication middleware, and deployment middleware, which collectively form a hierarchical structure. Specifically, the RSCA deployment middleware supports the reconfiguration of component-based robot applications including installation, creation, start, stop, tear-down, and un-installation. Since the original SCA lacks real-time guarantees and QoS support, we have significantly extended it while maintaining backward compatibility so that URC robot developers can use existing SCA tools. We have fully implemented RSCA and performed measurements to quantify its run-time performance. Our implementation clearly shows the viability of RSCA.

Keywords: Middleware, QoS, Ubiquitous Robotic Companion (URC)

1. INTRODUCTION

The convergence of disparate technologies has been significantly intensified in recent years. Robot industry is surely a representative example of this trend where electronics, communication, and software technologies are unified with automatic control and mechatronics technologies. Recently, the ubiquitous robot companion (URC) project has been launched in Korea with an aim of putting networked service robots into practical use in residential environments by overcoming technical challenges of home service robots. While the usefulness of an intelligent service robot has been evident for a long time, its emergence as a common household device has been painfully slow. This is due in part to the variety of technologies involved in creating a cost-effective robot. A modern service robot often makes a self-contained distributed system, typically composed of a number of embedded processors, hardware devices, communication buses, and computers. The logistics behind integrating these devices are dauntingly complex, especially if the robot is to interface with other household devices. The ever-falling prices of high performance CPUs and the evolution of communication technologies has made the realization of robots' potential closer than ever. What is left is to address the complexity of robotic technology convergence.

At the head of this effort is the URC robot project. Under development since last year, the URC has been conceived as "a robot friend to help people anywhere, anytime." The project's aim is to improve robot technology and facilitate the spread of robot use by making them more cost-effective and practical. Specifically to that end, it has been proposed that a robot's most complex calculations be handled by a high-performance remote server which is connected via a broadband communication network. For example, the vision or navigation systems that need a high performance MPU or DSP would be implemented on a remote server, and the robot itself would act as a thin client, making it cheaper and more lightweight.

Clearly, this type of system demands a very sophisticated software platform to operate it. In order to make the logistics of such a system manageable, the software should provide (1) a framework in which programs can be executed in a distributed environment, (2) a dynamic deployment mechanism by which a program can be loaded, reconfigured, and run, (3) real-time capabilities that allow robot software to meet hard deadlines, (4) QoS capabilities which can support robotic vision and voice processing, and (5) a management capability for limited resources and heterogeneous hardware inherent in the URC robot. In addition, the robot software should be flexible enough to use the very limited resources and heterogeneous hardware inherent in most modern robot systems.

Beyond simply creating such a platform, the desired goal is to create a standard that could serve the robotics community at large. Recently, there has been a great deal of research activity in this area, and yet there is still no current standard that has garnered international approval. In this paper, we thus propose a new middleware architecture for networked service robots by adopting an existing middleware technology from the software radio (SDR) domain. It is called Software Communications Architecture (SCA) [1]. We extend it for use in URC robots. The SCA was defined by Joint Tactical Radio Systems (JTRS) and has become a de

facto standard middleware currently adopted by the SDR forum. It is now widely accepted as a viable solution to reconfigurable component-based distributed computing for adaptive wireless radio terminals and base stations. In spite of its numerous strengths as embedded middleware, it cannot be directly applied to our URC robots since it lacks QoS capabilities in terms of both QoS specification and enforcement. Thus, we significantly extend it and we name the end result Robot Software Communications Architecture (RSCA). The RSCA provides a standard operating environment for robot applications together with a framework that expedites the development of such applications. We also allow robot developers to use their familiar robot software frameworks while deriving benefits from RSCA. This is possible by our mechanism that integrates conventional robot software frameworks into RSCA. We have fully implemented RSCA and performed measurements to quantify its run-time performance. Our implementation clearly shows the viability of RSCA.

The remainder of the paper is organized as follows. In Section 2, we give an overview of the system software configuration specified by RSCA. In Sections 3 and 4, we explain the RSCA core framework and introduce the QoS capabilities of RSCA as a key extension to the original SCA. In Section 5, we show how conventional robot software frameworks can be integrated into RSCA. Finally, we conclude this paper in Section 6.

2. Overall Structure of RSCA

The RSCA is specified in terms of a set of common interfaces for robot applications as the SCA is. These interfaces are grouped into two classes: (1) the standard operating environment (OE) interfaces and (2) the standard application component interfaces. The former defines APIs that developers use to dynamically deploy and control applications and to exploit services from underlying platforms. The latter defines interfaces that an application component should implement in order to exploit the component-based software model supported by the underlying platforms.

As shown in Fig. 1, the RSCA's operating environment consists of a real-time operating system (RTOS), communication middleware, and deployment middleware called core framework (CF). Since RSCA exploits COTS software for the RTOS and communication middleware layers, most of the RSCA specification is devoted to the CF. More specifically, RSCA defines the RTOS to be compliant to the PSE52 class of the IEEE POSIX.13 Real-Time Controller System profile [2], and the communication middleware to be compliant to minimum CORBA [3] and RT-CORBA v.1.1 [4]. The CF is defined in terms of a set of standard interfaces called CF interfaces and a set of XML descriptors called domain profiles as will be explained subsequently in Section 3.

The RTOS provides a basic abstraction layer that makes robot applications both portable and reusable on

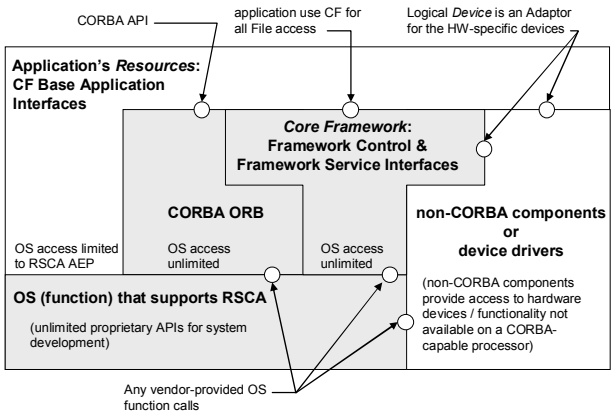

Fig. 1 Overview of operating environment of RSCA.

diverse hardware platforms. Specifically, a POSIX compliant RTOS in RSCA defines standard interfaces for multi-tasking, file system, clock, timer, scheduling, task synchronization, message passing, and I/O to name a few.

The communication middleware is an essential layer that makes it possible to construct distributed and component-based software. Specifically, the RT-CORBA compliant middleware provides (1) a standard way of message communication, (2) a standard way of using various services, and (3) real-time capabilities. First, the (minimum) CORBA ORB in RSCA provides a standard way of message communication between components in a manner transparent to heterogeneities existing in hardware, operating systems, network media, communication protocols, and programming languages. Second, the RSCA communication middleware provides a standard way of using various services. Among others, naming, logging, and event services are the key services that RSCA specify as mandatory services. Finally, the RT-CORBA in RSCA provides real-time capabilities including static and dynamic priority scheduling disciplines and prioritized communications in addition to the features provided by CORBA. Robot application developers are free to exploit these real-time capabilities to meet their applications' hard deadlines. Note that the

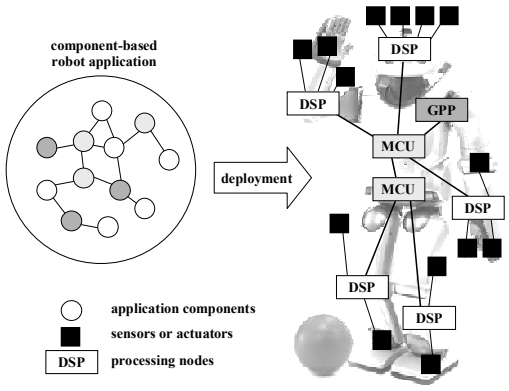

Fig. 2 Deployment of component-based robot applications.

original SCA's communication middleware does not support real-time capabilities since it recommends using minimum CORBA instead of RT-CORBA.

The deployment middleware layer provides a dynamic deployment mechanism by which robot applications can be loaded, reconfigured, and run. A URC robot application consists of application components that are connected to and cooperate with each other as illustrated in Fig. 2. Consequently, the deployment entails a series of tasks that include determining a particular processing node to load each component, connecting the loaded components, enabling them to communicate with each other, and starting or stopping the whole URC robot software.

3. RSCA Core Framework

As mentioned above, the RSCA CF is defined in terms of a set of standard interfaces called CF interfaces and a set of XML descriptors called domain profiles. The RSCA core framework interfaces are composed of interfaces for application components, domain management, and services. The deployment middleware is therefore the implementation of the domain management and service part of the RSCA core framework.

In this section, we explain the structure of the RSCA core framework in detail and the functionalities it provides.

3.1 Structure of RSCA Core Framework

The CF interfaces consist of three groups of APIs as shown in Fig. 3. (1) The base application interfaces are the interfaces that the deployment middleware uses to control each of the components comprising an application. Thus, every application component should implement these interfaces. These interfaces include the functionalities of starting/stopping a resource and configuring the resource. (2) The CF control interfaces are the interfaces provided to control the robot system. Controlling the robot system includes activities such as installing/uninstalling a robot application, starting/stopping it, registering/unregistering a logical device, tearing up/down a node, etc. (3) The service interfaces are the common interfaces that are used both by the deployment middleware and applications. Currently, three services are provided: distributed file system, event, and QoS.

The domain profiles are a set of XML descriptors describing configurations and properties of hardware and software in a domain. They consist of seven types of XML descriptors as shown in Fig. 4. (1) The *Device Configuration Descriptor* (DCD) describes a hardware configuration and (2) the *Software Assembly Descriptor* (SAD) describes a software configuration and the connections among components. (3) These descriptors consist of one or more *Software Package Descriptors* (SPD) each of which describes a software component (*Resource*) or a hardware device (*Device*). (4) The *Properties Descriptor* (PRF) describes optional reconfigurable properties, initial values, and executable

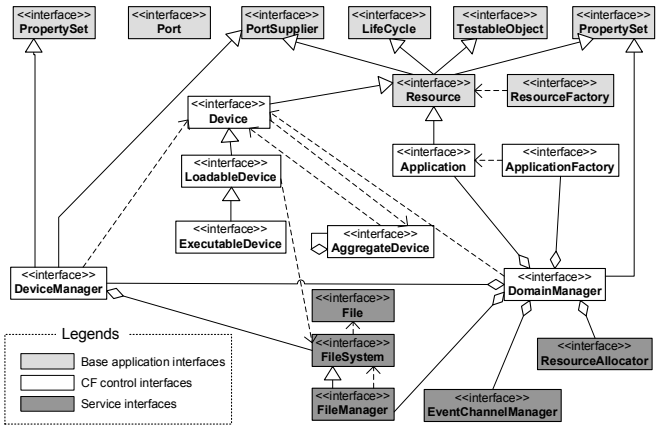

Fig. 3 Relationships among CF interfaces.

parameters that are referenced by other domain profiles. (5) The *Domainmanager Configuration Descriptor* (DMD) describes the *DomainManager* component and services used. (6) The *Software Component Descriptor* (SCD) describes the interfaces that a component provides or uses. Finally, (7) the *Device Package Descriptor* (DPD) describes a hardware device and identifies the class of the device.

3.2 Functionalities of RSCA Core Framework

Primarily, the RSCA core framework provides (1) dynamic system reconfiguration, (2) QoS and real-time guarantees, (3) heterogeneous distributed computing, and (4) heterogeneous resource management.

Dynamic system reconfiguration: In RSCA, the system reconfiguration is supported at three different levels: component level, application level, and deployment level. For reconfiguration at the individual component level, the RSCA deployment middleware provides a way to specify and dynamically configure reconfigurable parameters of components. Among the RSCA CF interfaces, *PropertySet* provides interfaces that allow applications to query and configure the reconfigurable parameters at runtime. For the application-level reconfiguration, the RSCA deployment middleware provides a way to describe an application in various possible configurations (structures and parameters), each for different application requirements

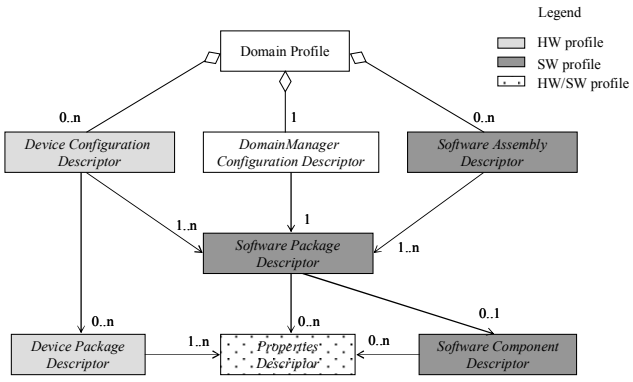

Fig. 4 Relationships among domain profiles.

and constraints. The deployment-level reconfiguration indicates that the RSCA deployment middleware should choose one of possible assemblies as is appropriate to the current resource availabilities.

QoS and real-time guarantees: As will be explained subsequently in Section 4, the deployment middleware supports application-level QoS guarantees while the RTOS and communication middleware support real-time guarantees for individual components.

Heterogeneous distributed computing: The RSCA deployment middleware hides the distributed nature of hardware platforms from applications by having distributed nodes be seen as a single virtual system or a domain. Robot applications need not consider how many processing nodes the domain consists of or which communication medium they use.

Heterogeneous resource management: RSCA deployment middleware supports heterogeneous resource management via the *Device* interface. The *Device* interface provides interfaces to allocate and de-allocate a certain amount of resource such as memory, CPU, and network bandwidth. The *Device* interface also supports the synchronization of accesses to a resource by providing the resource usage and management status. A developer should, of course, choose and implement how resources are allocated and synchronized based on the efficiency of resource usage.

4. QoS and Event Support in RSCA Core Framework

QoS and event services are the newly added services that mostly delineate RSCA from the original SCA. In this section, we explain how these services are provided by the RSCA core framework in detail.

4.1 QoS Support in RSCA Core Framework

While home service robots are heavily involved in real-time signal processing such as vision and voice processing, the original SCA lacks QoS capabilities in terms of both QoS specification and enforcement. Thus, we have significantly extended the SCA for the QoS support in defining RSCA. Specifically, we have (1) extended domain profiles to allow for resource and QoS requirements specification, (2) added services providing admission control and resource allocation to the RSCA core framework, and (3) extended the software communication bus based on the real-time ORB following the RT-CORBA v.1.1 specification. All of these extensions are made while maintaining backward compatibility so that URC robot developers can use existing SCA tools.

Using our RSCA core framework, robot application developers can achieve their desired QoS by simply specifying their requirements in the 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) described in Fig. 4. Since a legacy SCA SAD describes only connections or flows of messages between components, we extend various fields in the SAD to specify QoS-related information such as the sampling periods and the maximum latencies.

Robot application component developers should specify in the extended fields of XML descriptors resource demands in terms of dependencies on the hardware and the expected computational resource requirements for data processing. Such XML files are the *Software Package Descriptor* (SPD) and *Software Component Descriptor* (SCD) explained in Fig. 4. Along with this, application component developers should implement a predefined set of configurable property operations that the RSCA core framework invokes to deliver the results of resource allocation. For the implementation of configurable property operations, RSCA provides a skeleton component implementation from which QoS-aware components will be derived.

In order to guarantee the desired QoS described in the domain profiles, a certain amount of resources needs to be allocated to each application based on the current resource availability, and this must be enforced throughout the lifetime of the application. This involves admission control, resource allocation, and resource enforcement. For admission control and resource allocation, we added the *ResourceAllocator* component as shown in Fig. 3. On the other hand, for the resource enforcement, we rely on the COTS layer of the RSCA operating environment following the design philosophy of SCA. To aid in understanding how the desired QoS is guaranteed, we explain the application creation process in RSCA.

An application in a RSCA domain is created by the *ApplicationFactory* component which belongs to the RSCA domain management part and is in charge of instantiating a specified type of application. When *ApplicationFactory* instantiates an application in RSCA, it ascertains its QoS requirements from the domain profile and then passes the information to the *ResourceAllocator*. If the application is admissible, the *ResourceAllocator* generates the resource allocation plan for the application based on current resource availability. The *ApplicationFactory* component performs the resource allocation plan generated by *ResourceAllocator* in the following steps: it deploys all components onto the loadable/executable devices as designated in the plan, and then it delivers scheduling parameters to each component. On receiving the scheduling parameters, each component should set the RT-CORBA scheduling policy with the given scheduling parameters, and ascertain that those are enforced throughout its lifetime.

4.2 Event Support in RSCA Core Framework

In RSCA, a distributed event service is mandatory. Although the CORBA event service provides a standardized way of producing or subscribing to an event to and from a certain event channel, there are two problems in using it for robotic applications. First, it is a bit difficult and complex to make a connection to a CORBA event channel. Second, such a connection established using a CORBA event channel is hidden in

RSCA, making it hard to provide further services such as the lifecycle management of an event channel and QoS guarantees. Thus, instead of using the CORBA event channel service as in the SCA, we define our own. To do so, we have (1) extended domain profiles of the original SCA to allow for describing connections using CORBA event channels and (2) added to the RSCA core framework a service providing the life-cycle management of event channels.

When using our RSCA core framework, robot application developers can describe a connection between components via an event channel in our extended software assembly descriptor. The associated event channel is identified with its unique name. When *ApplicationFactory* creates an application, it should locate the CORBA event channel associated with the designated name and pass the channel to the application component. In doing so, the *EventChannelManager* shown in Fig. 3 provides interfaces to locate the event channel for the *ApplicationFactory*. The *EventChannelManager* also manages the lifecycle of event channels: creates or destroys event channels dynamically on needs.

5. Integration of Conventional Robot Software Frameworks into RSCA

Despite the various merits of RSCA, robot application developers often find it difficult to apply RSCA to their program development, particularly in the beginning phase of robot software development. One of the reasons for this is that robot application developers are very much accustomed to conventional robot software frameworks that have been used for decades in the robot industry. Unfortunately, these legacy frameworks are very different from RSCA.

To expedite the adoption of RSCA among robot application developers, it is thus desirable to integrate existing robot software frameworks into RSCA. Otherwise, robot application developers would have to use a conventional robot framework from software design to the early stage of implementation. Then they would have to restructure the partially implemented robot software in order to componentize and integrate it into the RSCA framework. If this is conducted manually, this will lead to the loss of productivity. Thus, the integration of legacy robot software frameworks into RSCA should be made transparent to robot application developers. In this section, we show how we integrate a conventional robot software framework into RSCA. For demonstration, we have chosen the ERSA (Evolution Robotics Software Architecture) of Evolution Robotics [5] as a representative conventional robot software framework. It is very popular and widely used for networked service robot applications. We first briefly overview ERSA and describe how to integrate it into RSCA.

5.1 ERSA Framework

ERSA is a commercial robot software framework that enables the component-based development of robot application. Among three layers of ERSA, Behavior Execution Layer (BEL) provides an environment that defines robot behaviors and executes them. For modularity, each behavior component's configuration and in/out ports are described by an XML file. More than one behavior components can be aggregated to make up a behavior network, which runs as a coarse grained behavior. Tasks in Task Execution Layer (TEL) implements planning and sequencing routines, and is able to communicate with each other in an event-driven manner. Basically, an application in ERSA consists of a set of tasks. TEL provides tasks with a mechanism to load a behavior network and execute it. A task that executes a behavior network is called a primitive task. By running a primitive task, it is possible to execute a set of behaviors in a sequential or parallel way. In both layers, the execution engines that are hidden from users execute behavior or task components and arbitrate the interaction between the two layers. Finally, Hardware Abstraction Layer (HAL) supports the abstraction of the hardware devices of a robot. Configuration and in/out ports of a device component is also described by an XML file.

5.2 Integration of ERSA into RSCA

Our basic idea of integration is shown in Fig. 5. Our strategy is to make each ERSA component into one RSCA component. To begin with, RSCA componentization is accomplished by adding wrapper code and making domain profiles that include deployment information. Wrapper code is made from CORBA stubs generated by an IDL compiler, and some user-supplied code that connects the original component to CORBA stubs.

As the first step, we map a task in ERSA to a RSCA component. This is needed because an application in ERSA consists of a set of tasks and each task can be considered as a unit of deployment. Specifically, a task is mapped to a resource which is a basic software component. In the second step, we map one behavior network, rather than individual behavior, to an RSCA component. This is needed because ERSA behaviors do

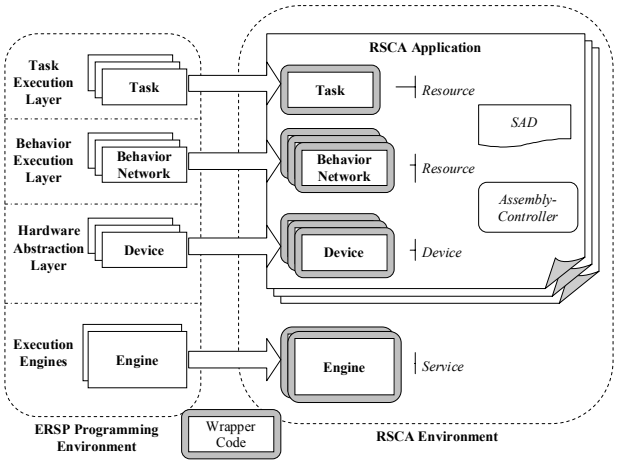

Fig. 5 Mapping of ERSA and its applications into RSCA environment

not have their own execution context, and they are too fine-grained to be a unit of deployment. Hence one behavior network or equivalently one primitive task is mapped to a resource. In the third step, we map each ERSA device component to an RSCA device component: that is *LoadableDevice* or *ExecutableDevice*. Finally, the ERSA execution engines are componentized as RSCA *Service* to manage robot applications in their original manner. They support the parallel and sequential execution of components, and also arbitration of components when their outputs are conflicting. As such, we make the execution engines an independent *Service* component instead of being a part of RSCA core framework. Since different robot domains often require different policies and requirements for behavior coordination, it is desirable to make them easily configurable and interchangeable.

On the other hand, each behavior and device component as well as a behavior network in ERSA is accompanied by an XML descriptor that has full information about them. We can easily create RSCA domain profiles from those descriptors. Specifically, SPD, SCD, and PRF profiles are needed for each behavior network or task component, and DPD, DCD, and PRF profiles are needed for each device component. Finally, to complete the integration of an application, SAD profile and *AssemblyController* should be added. They describe and control the connections of components that constitute the robot application.

6. Conclusions

In this paper, we have presented the Robot Software Communication Architecture (RSCA) we have developed to address the complexity inherent in networked home service robots. The RSCA provides a standard operating environment for robot applications together with a framework that expedites the development of such applications. The operating environment is comprised of a real-time operating system, communication middleware, and deployment middleware, which collectively form a hierarchical structure. The RTOS compliant to the PSE52 in IEEE POSIX.13 provides a basic abstraction layer that makes robot applications both portable and reusable on diverse hardware platforms. The communication middleware compliant to minimum CORBA and RT-CORBA v.1.1 provides an abstraction layer that hides the heterogeneity of distributed nodes in a URC robot thereby making the distributed application components able to flexibly communicate with each other. In addition to this, RT-CORBA provides robot applications with the real-time capabilities that applications can exploit to meet their hard deadlines. Finally, the deployment middleware called the RSCA core framework provides (1) a framework in which programs can be executed in a distributed environment, (2) a dynamic deployment mechanism by which a program can be loaded, reconfigured, and run, (3) real-time capabilities that allow robot software to meet hard deadlines, (4) QoS capabilities which can support robotic vision and voice processing, and (5) a management capability for limited resources and heterogeneous hardware inherent in the URC robot. As a result, the RSCA deals with many of important problems arising in creating an application performing complex tasks in the URC robot composed of the heterogeneous and distributed hardware. We have also shown that how conventional robot software frameworks can be integrated into RSCA. This allows robot software developers to develop their applications using conventional robot software frameworks without worrying about RSCA and to integrate their implemented applications into RSCA later.

The proposed RSCA is currently in an adoption process as a Korean domestic standard and is waiting for the industry approval.

REFERENCES

[1] Joint Tactical Radio Systems. “Software Communications Architecture Specification V.3.0,” August, 2004.

[2] ISO/IEC ISP 15287-2. “IEEE Std 1003.13, Information Technology - Standardized Application Environment Profile - POSIX Realtime Application Support (AEP).” Feb. 2000.

[3] Object Management Group. “The Common Object Request Broker Architecture: Core Specification Revision 3.0.” Dec. 2002.

[4] Object Management Group. “Real-Time CORBA Specification Revision 1.1.” OMG document formal/02-08-02 (August 2002).

[5] Evolution Robotics. “ERSP 3.0 Users Guide.” http://www.evolution.com, 2004.

[6] Object Management Group. “Unified Modeling Language (UML) Superstructure Specification v.2.0.” OMG document formal/05-07-04, http://www.uml.org.