Paper:

Software Deployment Infrastructure for Component Based RT-Systems

Noriaki Ando*, Shinji Kurihara*, Geoffrey Biggs*, Takeshi Sakamoto, Hiroyuki Nakamoto***, and Tetsuo Kotoku***

*Intelligent Systems Research Institute, National Institute of Advanced Industrial Science and Technology (AIST)
Tsukuba Central 2, 1-1-1 Umezono, Tsukuba-shi, Ibaraki 305-8568, Japan
E-mail: n-ando@aist.go.jp
**Technologic Arts Inc., Cosmos-Hongo 9F, 4-1-4 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan
***System Engineering Consultants Co., Ltd., Setagaya Business Square, 4-10-1 Yoga, Setagaya-ku, Tokyo 158-0097, Japan
[Received October 12, 2010; accepted February 7, 2011]

In component-based Robotic Technology (RT) systems, launching the system involves installing component binary files to the target computers, instantiation of components, and establishing connections between components. In order to operate RT systems with many CPU nodes effectively, the deployment features provided by the middleware are important. Deployment means system life-cycle management, including software installation, configuring components, and launching components. In this paper, we describe deployment tools for RT systems. The component deployment functionality is realized based on the OMG Robotic Technology Component (RTC) specification [1]. Description formats are defined, a service interface is designed and tools are implemented using the OpenRTM-aist that is the implementation of the OMG RTC specification. The implemented deployment infrastructure is evaluated and discussed, and issues and potential future work are considered.

Keywords: software deployment, software platform, distributed systems, RT-middleware, RT-component

1. Introduction

Next generation robots are still lacking in functionality, have too wide a variety of uses, suffer from a high cost of development and price, and face issues in safety. As a result, few of them have satisfied quality and price needs well enough to be marketed as commercially available products. Since the killer application for next generation robots is not yet clear, it is important to develop markets in the same way as information technologies such as personal computers did in the past: through radical innovation [2] in which open modular technologies allow various applications to be trialed so as to explore promising robot uses.

This environment encourages, in Japan and in other countries, the research and development of software platforms which promote modularized robots functionalities, also known as Robotic Technology (RT), and easier development of inexpensive, flexible robot systems [3–9].

In this paper, we will discuss the installation, configuration, and launch of components in robot systems consisting of multiple CPU nodes and multiple software components, (the deployment RT-systems) and propose the features necessary to achieve this. In Section 2, we describe the process of general software deployment. In Section 3, we discuss the required deployment framework for robot systems. In Section 4, on the basis of the discussion in Section 3, we define services for component deployment in OpenRTM-aist, the authors' component framework. In Section 5, we implement and evaluate the effectiveness of the services. In Section 6, we give conclusions and discuss future challenges.

2. Software Deployment

The installation, configuration and launch of software such as executable files and modules onto target computers are collectively referred to as deployment.

Well-known generic deployment standards include the Java-specific J2EE Deployment Specification [10], OSGi (Open Services Gateway initiative) [11], and the Deployment and Configuration Specification [12] issued by OMG (Object Management Group). **Fig. 1** shows a general outline of deployment features common to these systems.

The deployment process in **Fig. 1** will be explained based on the terms defined in [12, 13]. In the first step, installation, a developed piece of software is released in a package including binary files and meta data describing its components. This is placed in a preparation area called a repository. The second step is the planning process, in which a plan is created describing how to execute the software in the target environment, including where each piece of software will be deployed and how they will be connected. In this step, information on deployment, connections, and configuration is described using XML,

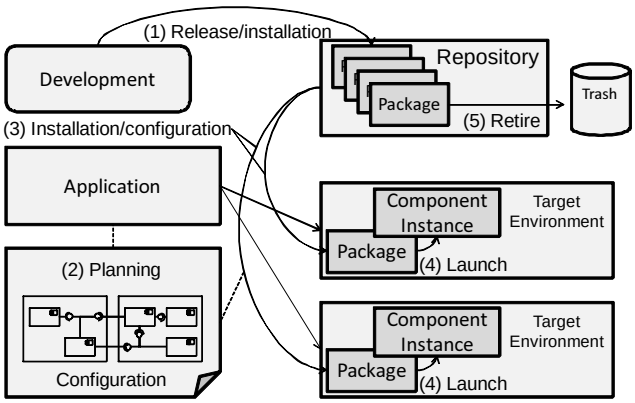


Fig. 1. General software deployment process.

script languages, or similar methods.

During the launch step, components are instantiated and connected based on the previously-created plan and the system is launched. When terminating an application, its components are de-activated. In contrast to standard software release processes, deleting a package that has became old and unnecessary from the repository is called un-installation, and it is at this point that the software is retired and its lifecycle ends.

3. Deployment in RT Systems

This section discusses the functionality required for the deployment of RT systems.

3.1. Characteristics of RT Systems

RT systems recently tend to be decentralized due to a reduction of CPUs in size and price, and the development of networks. In addition, in a system with many external sensors working together, such as networked robots, sensor nodes, robots, and other pieces of hardware are often incorporated into or removed from the system dynamically.

Unlike software for simple distributed systems, therefore, sensors and actuators connected to the computer, devices that combine them, and even environmental information from outside the devices can be a target for the software to manage. In short, a more dynamic resource management system is necessary.

Furthermore, in RT systems there are mixed structures with various densities and sizes, from real-time control requiring components to be tightly-coupled to a service-oriented, loosely-coupled structure used in the higher layers of the system.

On the basis of such characteristics, we will now discuss the necessary functionality for deployment of RT systems in terms of the lifecycle from system design, through component implementation, to system operation.

3.2. Component Lifecycle

We will now outline the lifecycle of components, which are the smallest indivisible unit in a system.

First, the detailed functionality of each component is determined based on splitting the requirements and design of the system into modules. The components are implemented according to their designs, tested, packaged, and then installed in suitable locations.

The installed components are set up and activated based on the pre-planned system when the application is launched. In addition, the configuration of the RT system must be dynamically changed during operation based on the occurrence of relevant events. After that, termination of the application causes all the activated components to stop. Finally, unnecessary components are un-installed.

When the application is launched, it is necessary to obtain information as to whether components to be used are available and resources have been secured in the target environment. A middleware is necessary to provide these functions.

3.3. System Lifecycle

The lifecycle of a component-based software system begins with designing the overall system, continues with dividing functions into modules according to the design, and then on into a design process for these components.

When building the system by combining the individual components, even if there are no components actually operable, most parts of the system can be designed in advance, as long as there are specifications (profile information) of the components available.

Based on the information in the pre-created plan, components are installed and instantiated and connections between components are established on relevant nodes at launch of the system. Components are dynamically generated and destroyed as necessary after system launch according to the operational scenario. After terminating the system, relevant components are stopped and terminated.

As described above, robot systems are required to achieve both loose coupling and tight coupling between components and dynamic configuration changes in the system. The ability to launch components on a node in real-time is necessary. Fine control is needed for deployment, for example, components required to be tightly coupled must be launched in a same process, while other components may be launched in separate processes in order to prevent interference between the components. In addition, these component deployment operations need to be executed on many nodes in the network.

3.4. Existing Middleware and Deployment

Many existing robot middlewares provide a function equivalent to deployment. For instance, ROS provides a command called roslaunch, which sets and launches local or remote nodes (the name for components in ROS) through SSH (secure shell) [4]. OPRoS uses a tool called TaskInterpreter to execute tasks, and provides the OPRoS

TaskExecutor RemoteController, a GUI client for Task-Interpreter [6]. An API (Application Programming Interface) related to deployment and configuration is defined, which is called from unique script languages to load and launch components. It also has an XML-based configuration system. A deployment system called Ice (Internet communication engine) Grid, part of the Ice distributed object middleware, is used to launch components from a command line.

4. Service and Data Model

On the basis of the above, we will now discuss what is necessary to install, set up, and launch RT-Components (RTC) and to launch the whole RT system.

For installing components from remote locations, obtaining the profile of the components, launching and terminating the components, and so on, a service operating in the target environment is necessary. In addition, a client application or a tool is necessary to send a command to this service to perform a predetermined operation.

For obtaining information about installed components, a specification description format of the components, a tool to fetch information from a module, and so on are necessary. In addition, a data structure for system description which describes deployments and connections of the components and can be used for planning must be defined.

Based on this, the following elements necessary for deploying components on the RT system are realized in this paper.

1) Component profile
2) System profile
3) RT-Component (RTC) manager service
4) Various tools

4.1. Component Profile

Prior to the launch of the system, it is necessary to obtain information about the components to be used in advance when planning. This information includes 1) information on the type of components, which pertains to the type before instantiation and 2) information on the component instances, which depends on each instance and its use.

In the case of RT-Components (RTC), an interface to obtain the information in 2) is defined in [1]. On the other hand, although the information in 2) is only used by the information in 1) currently, there should only be an amount of information necessary to provide the developer with sufficient information available for planning.

We define a component specification description called the RT-Component (RTC) Profile to describe the information in 1). The RTC Profile is a component specification description method for component design, template source code generation, as a template for document description, for offline design of the system, as meta-data for deployment, and as a template for re-use.

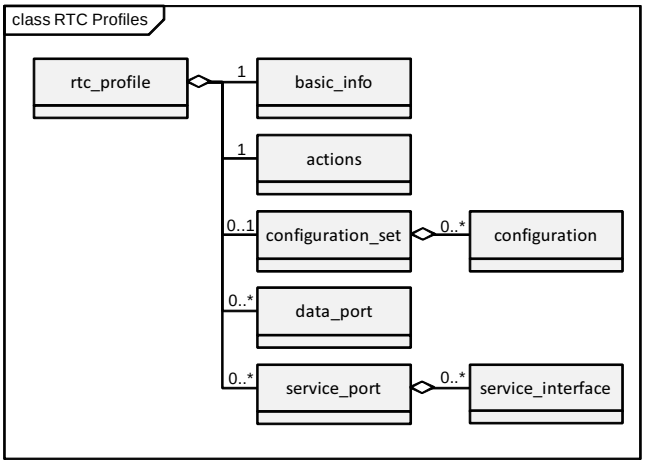


Fig. 2. RTC Profile class diagram. RTC Profile defines data structure for a component description.

Table 1. The RTC Basic Profile::basic_info definition.

Element name	Description
name	The name of the RTC
componentType	The type of the RTC
category	RTC categorisation
description	Brief description of the RTC
executionRate	Frequency of execution of the RTC's behaviour
executionType	Type of execution (periodic, externally-triggered, etc.)
vendor	RTC vendor
version	Version number

The RTC Profile has the following three parts: i) RT-Component (RTC) Basic Profile; ii) RT-Component (RTC) Document Profile; and iii) RT-Component (RTC) Extended Profile. **Fig. 2** shows the data model of the RTC Basic Profile.

As shown in **Fig. 2**, the Basic Profile includes the basic meta-information of the RTC, with elements such as `basic_info`, `actions`, `configuration_set`, `data_port`, `service_port`, with "`rtc_profile`" as the root node.

Table 1 shows some attributes and their meanings as an example. The `basic_info` holds basic profile information of the RTC. These attributes include values mainly defined by the `ComponentProfile` of the OMG RT-Component (RTC) Specification. Although other elements also have attributes, they are not shown due to space limitations.

The RT-Component (RTC) Document Profile, for describing specifications of components in detail, and the RT-Component (RTC) Extended Profile package, for extending tools and applications, are also defined.

4.2. System Profile

The system profile, RTS Profile, is a model used to describe a system made up of RTCs and their connections

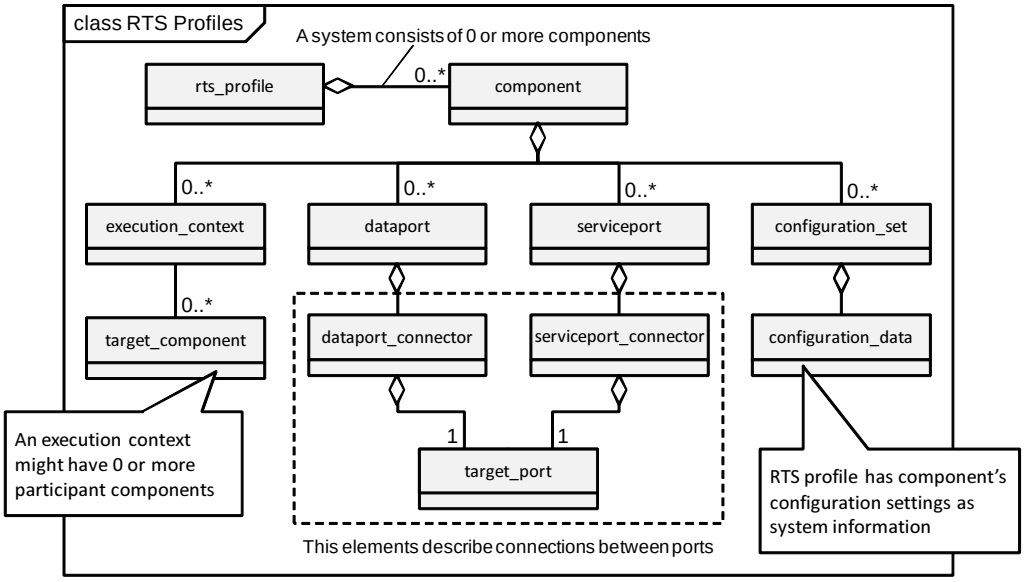


Fig. 3. RTS Profile class diagram. RTS Profile defines data structure for a system description.

based on the RT-Component (RTC) model [1] defined by the OMG RT-Component (RTC) standard.

The RTS Profile is made up of two packages, the RTS Basic Profile shown in **Fig. 3** and the RTS Extended Profile for describing additional information of the system.

The RTS Basic Profile includes basic meta information of the RT system. The RTS Basic Profile includes, as shown in **Fig. 3**, execution context information such as `component`, `execution_context`, `dataport_connector`, `serviceport_connector` and system configuration information such as deployment information, connection information, and configuration information of the components. It is all stored in an `rts_profile` node.

4.3. Manager Service

We propose a program called the "RT-Component (RTC) manager," which resides in each target environment so as to provide lifecycle management of RTCs and provide applications with information.

The RT-Component (RTC) manager is a service that provides functions of management of the lifecycle of the RT-Component (RTC) described above and information acquisition. Operations are accessible to any program in the network through the CORBA distributed object middleware interface. Operations are available to dynamically instantiate and delete RTCs from applications.

On the basis of the requirements derived from the discussion described above, we define the following interface (**Fig. 4**).

As main functions, 1) module load, unload and information acquisition; 2) component instantiation, deletion and information acquisition; 3) linking master and slave managers. The master-slave mechanism is a function for achieving a stable operation of the system and grouping the RTCs.

```
interface Manager
{
  ReturnCode_t load_module(in string pathname,
                           in string initfunc);
  ReturnCode_t unload_module(in string path);
   : (syncopated)
  RTObject create_component(in string mod_name);
  ReturnCode_t delete_component(in string name);
  RTCList get_components();
  ComponentProfileList get_component_profiles();
   : (syncopated)
  boolean is_master();
  ManagerList get_master_managers();
  ReturnCode_t add_master(in Manager mgr);
  ReturnCode_t remove_master(in Manager mgr);
  ManagerList get_slave_managers();
  ReturnCode_t add_slave(in Manager mgr);
  ReturnCode_t remove_slave(in Manager mgr);
};
```

Fig. 4. The RTC manager interface definition.

4.3.1. Master and Slave Managers

The simplest manager operation method is a single manager constantly activated to provide a service, receive commands from outside, launch and terminate the components, and so on. Multiple RTCs can be launched and terminated in one manager process to instantiate a system.

However, with this method, when just one unstable RTC causes a memory access violation (a so-called Segmentation Fault) or the like, the other RTCs, free from problems, are forced to quit together. On the other hand, since, when two or more RTCs are in a same process, communication between them becomes a simple function call, further improvements in performance can be expected by operating two or more closely cooperating RTCs in a same process. Unlike components in [4], any type and any number of OpenRTM-aist RTCs can be oper-

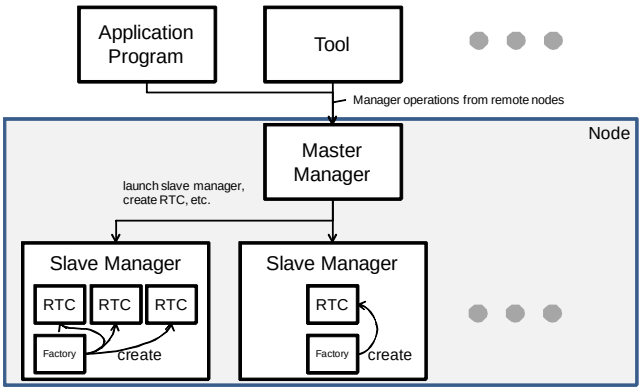


Fig. 5. The relation among master manager and slave managers.

ated in any process, and it is preferable for RTCs launched by the manager to take advantage of those merits so as to allow the operation of RTCs in any designated process.

For these reasons, the RT-Component (RTC) managers are divided into two types, masters and slaves, as shown in **Fig. 5**, which have the following functions.

The master manager accepts all the requirements from outside to the manager, and, according to the commands, launches slave managers and generates RTCs on any slave manager. The slave manager, according to commands from the master manager, launches RTCs and provides information to the master manager.

4.3.2. Configuration Function

Although in the RT-Component (RTC) specification [1] a configuration interface is defined for changing internal parameters, we defined a setting method using an argument given to the `create_component()` operation described above as a method for setting component parameters according to a system specification description when launching the components.

This method uses keys and values specified after a delimiter, "?," after the component ID in the following manner:

ID?<*key*>=<*value*>&<*key*>=<*value*>&...

Passing configuration parameters as the argument of the `create_component()` operation enables setting parameters before instantiation as components. In addition, we introduced an option, `manager=<host_name>:<port>`, in order to designate which slave manager an RTC is launched in. This option is interpreted in the master manager, and the `create_component()` operation is carried out on the designated slave manager. When the slave manager does not yet exist, the master manager launches it.

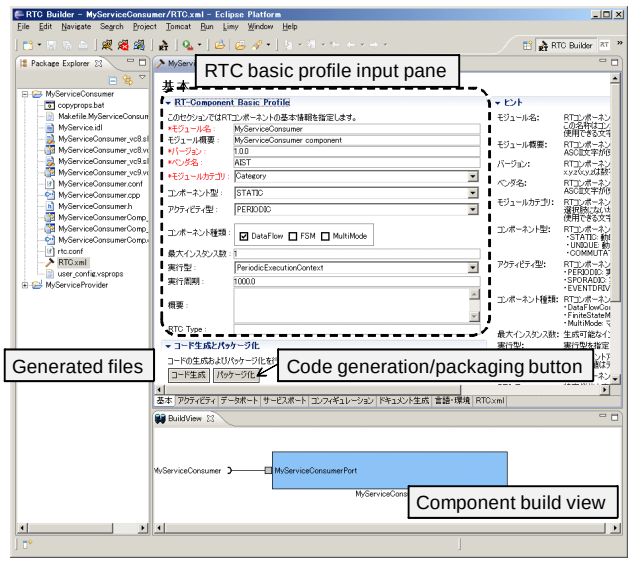


Fig. 6. The RTC Builder, an RT-Component development tool based on the component specification.

5. Implementation

As a tool to generate or use the RTC and RTS Profiles, described above, we have implemented the RT-Component (RTC) Builder and RT System Editor tools. They are implemented as Eclipse plug-ins to provide an integrated development environment.

5.1. RT-Component (RTC) Builder

The RT-Component (RTC) Builder (RTCB) is a tool in which a specification of an RTC is created so as to generate RTC Profile XML files and create RTC template code for various languages, such as C++, Java, and Python. **Fig. 6** shows a screenshot of RTCB.

The specification of an RTC is entered in the central editor fields. Entry items include basic meta-information, data ports, service ports, configuration parameters, language and environment settings, and information such as documentation. These items are entered so that the template code of the RTC can be generated.

After entering the specification, the "Generate" button is clicked to generate the template code in the selected language. If a development environment plug-in of the selected language has been installed in Eclipse, the implementation of the RTC logic can be continued on the same screen.

5.2. RT System Editor

RT System Editor (RTSE) is a tool for developing RT Systems. **Fig. 7** shows a screenshot of the RTSE online editor. RTSE has an offline editor, which reads RTC Profiles created in RTCB and designs the system offline, and an online editor, which connects and controls RTCs in operation and develops and verifies the system. The system

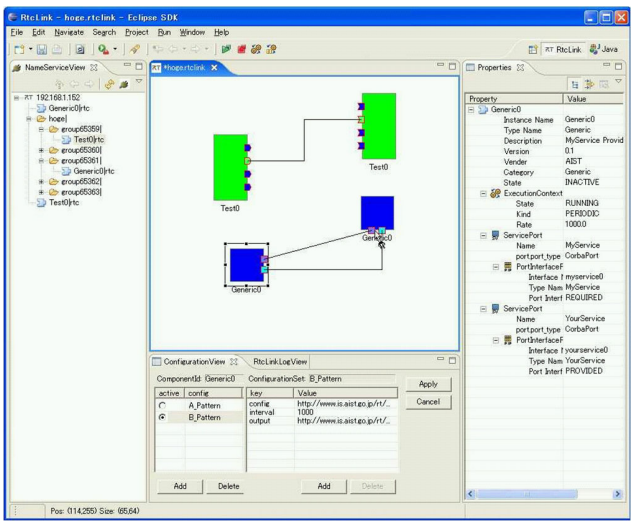


Fig. 7. The RT System Editor, an RT-Component based system development tool.

is built by arranging RTCs in the central editor screen, in which ports are connected, parameters are set, and so on. The tools can save, load, and reconstruct a created system as an XML file using the RTS Profile.

5.3. RT-Component (RTC) Manager

In this section, we will discuss specific implementation methods of the manager.

5.3.1. "Corbaloc" URL Scheme

In order to achieve the master-slave structure of managers described above, an interface is necessary between the master manager and the slave manager. In addition, the manager should provide applications in the network with a means of direct access without requiring a name server.

For this, we utilize the corbaloc scheme of the Interoperable Naming Service (INS), which is a method to designate an object reference by a URL string in CORBA [14].

A CORBA object is usually provided with a function to specify an object by an encoded character string called an Interoperable Object Reference (IOR). However, since an object is provided with a random "Object Key" every time it is generated and the object is encoded in the IOR together with the Object Key, it has a different IOR every time it is generated. On the other hand, the corbaloc scheme allows a CORBA object to have permanent access with a readable URL in the following format:

corbaloc:iiop:<*host_name*>:<*port_number*> /<*object id*>

Predetermining the port number and object ID using corbaloc enables direct access to an object on any host without going through an external service such as a name server. In this article, we set the port number of the master manager to "2810" and the object ID as "manager," and implemented the RTC Manager as a directly accessible CORBA object via the URL:

corbaloc:iiop:localhost:2810/manager.

5.3.2. Negotiation Between Master and Slave

With the corbaloc scheme, the slave manager can directly access the master manager without using an external service such as a name service. After launching, the slave manager searches the same host for the master manager and registers itself to the master manager. The master manager registers the slave manager in a slave manager list that is managed by the master manager itself, and the master and slave negotiation is complete.

From this point, the master manager can provide clients with information on RTCs on all the slave managers which are managed by the master manager itself through a call to all slaves.

5.3.3. GUI Operation Tools

In OpenRTM-aist, RTSE, described above, is a GUI tool for building and operating the RTCs and the RT System. In the Eclipse, a set of screens on which a series of operations are performed is referred to as a perspective. The perspective is made up of views and editors. As one perspective, RTSE is made up of a name service view, a system editor, and so on. We further added a manager view for operating the RTC manager.

Figure 8 shows the newly-implemented manager view, which allows operations for building the system, such as loading modules, instantiating components, and connecting the components, to be performed within one perspective.

5.3.4. CUI Operation Tool

We have implemented the "rtcshell" and "rtsshell" command toolkits for performing various operations on RTCs from the command line [15] (**Table 2**).

Connection, status change, modification of configuration parameters, and so on can be performed on RTCs and RTC managers as though they are files in a directory structures we call the "RTC tree." "rtmgr" is a command for operating managers, which uses load, unload, create, and delete of sub-commands to enable RTC modules to be loaded and unloaded and RTCs to be instantiated and deleted. In addition, the "rtsshell" commands use RTS Profiles generated by RTSE or other tools to enable the system to be re-built and the system to be launched and stopped.

5.4. Evaluation

We will now show usefulness of the manager and the tools by comparing the methods to launch remote components. In UNIX-based operating systems, it is easy to log in and launch commands through the secure shell (SSH)

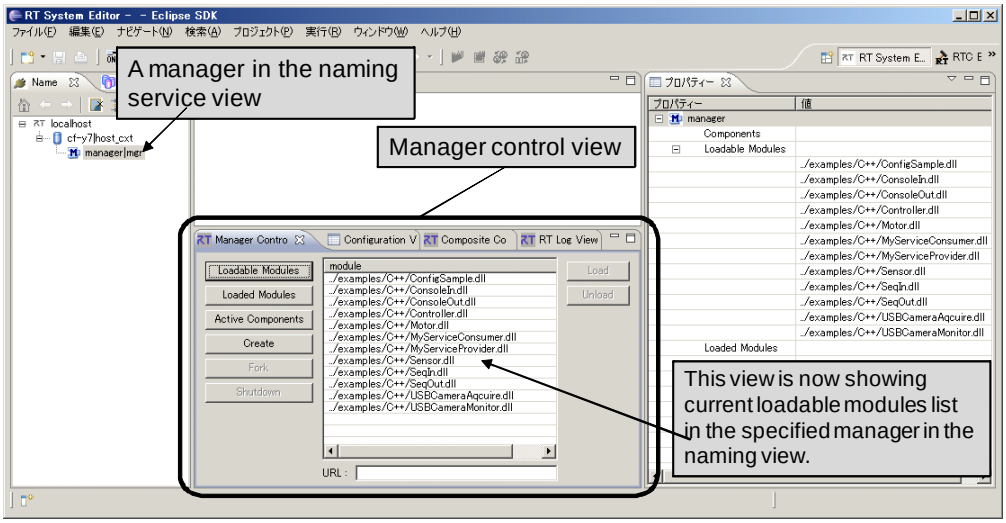


Fig. 8. The manager control view in the RT System Editor.

Table 2. The available commands list of the "rtcshell" and "rtshell."

Command	Description
rtcshell	
rtact	Activate a component.
rtcat	List component information.
rtcon	Connect two ports together.
rtconf	Manipulate configuration parameters.
rtcwd	Change the current RT tree directory.
rtdeact	Deactivate a component.
rtdis	Disconnect a connection.
rtfind	Search the RTC tree for components.
rtls	List RTCs of the RT tree.
rtmgr	Control managers.
rtpwd	Print the current RT tree directory.
rtreset	Reset a component.
rtsshell	
rtresurrect	Reconstruct system from RTS Profile
rtstart	Start a system
rtstop	Stop a system
rtteardown	Disconnect and shotdown a system
rtcryo	Save a system to a RTS profile file

tool, and this is one of the easiest methods to launch remote components. In fact in [4], the "roslaunch" command works as a wrapper for remote launch by SSH. We then compared remote launch times with RTSE and the rtmgr command, which were developed this time, with those by SSH (**Table 3**).

For SSH, we compared two methods: a method to launch components by entering a command after logging in without a password by a public key (manual SSH) with a method to give a launch command directly through SSH for automatic launch (automatic SSH). Each value indicates a time per one RTC when ten of them were launched.

Table 3. The component launch time comparison.

Method	time(per comp.) [ms]
Manual SSH (10 RTCs, diff. proc.)	1832
RTSE (10 RTCs, same proc.)	1760
rtmgr (one RTC diff. proc.)	792
rtmgr (one RTC, same proc.)	312
Auto SSH (10 RTCs, diff proc.)	178
rtmgr (10 RTCs, diff proc.)	84
rtmgr (10 RTC, same proc.)	37

On the other hand, when the "rtmgr" command is used, one command can launch multiple RTCs. We then compared a case in which one command was repeated ten times to launch RTCs (one RTC) with a case in which one command launched ten RTCs (10 RTCs). In addition, since "rtmgr" can designate each RTC to be launched in a different process (diff. proc.) or to be launched in a same process (same proc.), we also compared those cases.

Manual launch using SSH and RTSE takes 1.8 s per one RTC, or about 18 s for launching ten RTCs. These are originally designed for interactive use; we have presented them for comparison.

Table 3 indicates that "rtmgr" is faster when launching two or more RTCs and SSH is faster when launching only one RTC. It also indicates that the use of the simultaneous launch function of rtmgr can launch the RTCs significantly faster than other cases. In addition, it shows that, compared with the case in which they are launched as separate processes, overhead of the process launch is small (about 47 ms) when the RTCs are launched in the same process.

When "rtmgr" and RTSE are used, the degree of freedom is high. For example, components can be deployed in any process. However, in that case, each component can only be launched as an individual process when launching with SSH.

Table 4. A comparison among existing methods and our methods.

Methods	Separated processes	Same process	CUI operation	GUI operation	Embed to applications
SSH manual	YES	NO	YES	NO	–
SSH auto	YES	NO	YES	NO	difficult
RTSE	YES	YES	NO	YES	not difficult
rtmgr/rtctree	YES	YES	YES	NO	easy

In addition, internal functions of "rtcshell" are implemented in a Python module called "rtctree," and those functions are called from Python scripts to perform various operations to the components such as launching the components.

While each of the methods has advantages and disadvantages, the use of "rtmgr" and "rtctree" allows the RTCs to be easily built in an application and in general the RTCs can be launched and operated at a high speed (**Table 4**).

6. Conclusions

In this article, we have discussed deployment function necessary for an RT system with multiple nodes and multiple robot software components, and proposed and implemented services and tools for component profiles, system profiles, and deployment.

We have defined the RTC Profile to systematically describe component specifications, and shown that it can be utilized for source code generation and static system configuration. In addition, we have defined the RTS Profile to describe specifications of RT systems, and implemented the RT System Editor and "rtshell" tools, which use RTS Profiles to describe and reconstruct systems. We have shown that various tools can be implemented and data can be easily exchanged between the tools based on a common specification description.

We have also designed and implemented the RTC manager tool, which can manage the lifecycle of RTCs. By dividing the managers into master and slaves and making them cooperate, we have achieved flexibility of component deployment that allows both tight coupling and loose coupling of the components and stability in operations.

In this paper, the implemented functions of the RTC manager do not include cooperation with repositories and download and install of executable files. In the future, we will implement a more flexible, more useful deployment tool by designing and implementing a service which automatically performs planning and installation using RTC and RTS Profiles.

All the profile specifications defined in this article and the implemented services and tools are disclosed in [3].

Acknowledgements

This study is partially subsidized in 2009 by NEDO Intelligent Robot Technology Software Project (2007 to 2011).

References:

[1] Object Management Group, "Robotic Technology Component Specification Version 1.0," OMG specification, formal/2008-04-04, 2008.

[2] M. Aoki and H. Ando, "Modularity: the Nature of Emergent Organizational Architecture (Japanese)," Toyo-keizai Shinposha, 2002.

[3] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku, and W. K. Yoon, "RT-Middleware: Distributed Component Middleware for RT (Robot Technology)," 2005 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS2005), pp. 3555-3560, 2005.

[4] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," ICRA Workshop on Open Source Software, 2009.

[5] H. Bruyninckx, "Open robot control software: the OROCOS project. In: Robotics and Automation," 2001 IEEE Int. Conf. on Robotics and Automation (ICRA2001), Vol.3, pp. 2523-2528, 2001.

[6] E.-C. Shin, S.-K. Son, B.-W. Choi, B.-H. Hwang, and D.-H. Lee, "Development of OPRoS Software Components," The 6th Int. Conf. on Ubiquitous Robots and Ambient Intelligence (URAI 2009), pp. 810-812, 2009.

[7] A. Makarenko, A. Brooks, and T. Kaupp, "Orca: Components for Robotics," IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 2006), Workshop on Robotic Standardization, 2006.

[8] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust Monte Carlo Localization for Mobile Robots," The Journal Artificial Intelligence, Vol.128, No.1-2, 2000.

[9] G. Metta, P. Fitzpatrick, and L. Natale, "YARP: Yet Another Robot Platform," Int. J. of Advanced Robotics Systems, special issue on Software Development and Integration in Robotics, Vol.3, No.1, 2006.

[10] "JSR-000088 J2EE(TM) Application Deployment Specification 1.0 Final Release," 2002.

[11] OSGi Alliance, "OSGi Service Platform Release 4 Version 4.2 Core Specification," 2009.

[12] Object Management Group, "Deployment and Configuration of Component-based Distributed Applications," OMG specification, formal/2006-04-02, 2006.

[13] A. Carzaniga, A. Fuggetta, R. S. Hall, D. Heimbigner, A. v. d. Hoek, and A. L. Wolf, "A Characterization Framework for Software Deployment Technologies," Technical Report Department of Computer Science, University of Colorado, Boulder, Colorado, April 1998.

[14] Object Management Group, "Interoperable Naming Service," OMG specification, formal/00-11-01, 2000.

[15] G. Biggs, N. Ando, and T. Kotoku, "Run-time management of component-based robot software from a command line," 2010 Second Int. Conf. on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR2010), pp. 192-203, 2010.

Name:
Noriaki Ando

Affiliation:
Senior Research Scientist, Intelligent Systems Research Institute, National Institute of Advanced Industrial Science and Technology

Address:
Tsukuba Central 2, 1-1-1 Umezono, Tsukuba-shi, Ibaraki 305-8568, Japan

Brief Biographical History:
2002 Received Ph.D. degree from the Department of Information and Communication Engineering, The University of Tokyo
2003- Research Scientist, National Institute of Advance Industrial Science and Technology (AIST)
2009- Senior Research Scientist, National Institute of Advanced Industrial Science and Technology (AIST)
2010- Visiting Researcher, Technical University Munich

Main Works:
• "Networked Tele-micromanipulation Systems "Haptic Loupe"," IEEE Trans. on Industrial Electronics, Vol.51, No.6, pp. 1259-1271, Dec. 2004.
• "RTC-Lite: Lightweight RT-Component for Distributed Embedded Systems," SICE J. of Control, Measurement, and System Integration (SICE JCMSI), Vol.2, No.6, pp. 328-333, Nov. 2009.
• OMG Robotic Technology Component (RTC) standard (formal/08-04-04) and its implementation OpenRTM-aist

Membership in Academic Societies:
• The Japan Society of Mechanical Engineers (JSME)
• The Society of Instrument and Control Engineers (SICE)
• The Robotics Society of Japan (RSJ)

Name:
Shinji Kurihara

Affiliation:
System Engineer, Intelligent Systems Research Institute, National Institute of Advanced Industrial Science and Technology

Address:
Tsukuba Central 2, 1-1-1 Umezono, Tsukuba-shi, Ibaraki 305-8568, Japan

Brief Biographical History:
2003- Joined Systems Engineering Consultants Co., Ltd.
2009- System Engineer, National Institute of Advanced Industrial Science and Technology (AIST)

Main Works:
• "New data port implementation in OpenRTM-aist-1.0," JSME Conf. on Robotics and Mechatronics 2010 (ROBOMEC2010), 2A1-G03, Jun. 2010. (in Japanese)
• development of the OMG RTC standard compliant RT-Middleware OpenRTM-aist-Python

Name:
Geoffrey Biggs

Affiliation:
Research Scientist, National Institute of Advanced Industrial Science and Technology

Address:
Tsukuba Central 2, 1-1-1 Umezono, Tsukuba-shi, Ibaraki 305-8568, Japan

Brief Biographical History:
2007 Received Ph.D. in Electrical Engineering, University of Auckland, New Zealand
2007- Postdoctoral Fellow, National Institute of Advance Industrial Science and Technology (AIST)
2011- Research Scientist, National Institute of Advanced Industrial Science and Technology (AIST)

Main Works:
• G. Biggs and B. A. Macdonald, "A pragmatic approach to dimensional analysis for mobile robotic programming," Auton. Robots, Vol.25, No.4, pp. 405-419, 2008.
• robot development tools, robot software architectures, standardization in robotics

Membership in Academic Societies:
• The Institute of Electrical and Electronics Engineers (IEEE) Robotics and Automation Society

Name:
Takeshi Sakamoto

Affiliation:
Senior Systemconsultant, Robot Business Promotion Group, Technologic Arts Incorporated

Address:
Cosmos Hongo Bldg. 9F, 4-1-4 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan

Brief Biographical History:
2004- Joined Technologic Arts Inc.
2011- Part-time Lecturer, Department of Engineering and Design, Shibaura Institute of Technology

Main Works:
• "Model-based design of Intelligent Mobile Robot," Proc. of Int. workshop on Model Based Engineering for Robotics 2010 (RoSym'10), 2010.
• "Mathematical Framework for Localization Information Coordinate Reference System for Robotics," Proc. of Int. Workshop on Standards and Common Platform for Robotics 2010 (SCPR 2010) , 2010.
• "Robotic Localization Service Standard for Ubiquitous Network Robots," Robotics 2010 Current and Future Challenges, Houssem Abdellatif, 2010.
• modeling of a robot system, development of development tools

Membership in Academic Societies:
• The Robotics Society of Japan (RSJ)

Name:
Hiroyuki Nakamoto

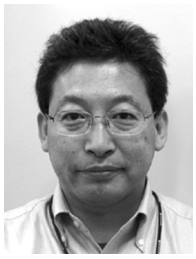
Affiliation:
Manager, Systems Engineering Consultants Co., Ltd.

Address:
4-10-1 Yoga, Setagaya-ku, Tokyo 158-0097, Japan

Brief Biographical History:
1996- Systems Engineering Consultants Co., Ltd.

Main Works:
• "Smart home for security and low power consumption with common network modules based on RT middleware," The 5th Int. Conf. on Advanced Mechatronics (ICAM2010), pp. 551-556, 2010.
• robot system, RT middleware

Membership in Academic Societies:
• The Robotics Society of Japan (RSJ)



Name:
Tetsuo Kotoku

Affiliation:
Group Leader, Intelligent Systems Research Institute (ISRI), National Institute of Advanced Industrial Science and Technology (AIST)

Address:
Tsukuba Central 2, 1-1-1 Umezono, Tsukuba-shi, Ibaraki 305-8568, Japan

Brief Biographical History:
1988- Research Scientist, Mechanical Engineering Laboratory (MEL), Ministry of International Trade and Industry (MITI)
1998-1999 Visiting Researcher, Northwestern University, USA
2001- Senior Research Scientist, Intelligent Systems Research Institute (ISRI), National Institute of Advanced Industrial Science and Technology (AIST)
2007- Research Group Leader, Intelligent Systems Research Institute (ISRI), National Institute of Advanced Industrial Science and Technology (AIST)

Main Works:
• virtual environments with force reflection, and their application to the tele-manipulation systems
• network robotics, reusing and sharing of robot technology (RT), and standardization of RT

Membership in Academic Societies:
• The Japan Society of Mechanical Engineers (JSME)
• The Society of Instrument and Control Engineers (SICE)
• The Robotics Society of Japan (RSJ)