KRS KOREA ROBOTICS SOCIETY

DAEJEON METROPOLITAN CITY

International Robot Week 2005

The 2nd International Conference on Ubiquitous Robots and Ambient Intelligence

http://www.robotweek.or.kr

Nov. 2-4, 2005

KAIST / DAEJEON, KOREA



MIC
Ministry of Information and Communication

KAIST
Korea Advanced Institute of Science and Technology

ETRI
Electronics and Telecommunications Research Institute

DIF
Daejeon Hightech Industry Promotion Foundation

KAIRA
Korea Advanced Intelligent Robot Association

RT (Robot Technology) -Middleware
– Towards Component Based Networked Robot Systems Development –

Noriaki Ando, Takashi Suehiro, Kosei Kitagaki and Tetsuo Kotoku
Intelligent Systems Research Institute
National Institute of Advanced Industrial Science and Technology (AIST)
AIST Tsukuba Central 2,Tsukuba,Ibaraki 305-8568, Japan
{*n-ando, t.suehiro, k.kitagaki, t.kotoku*}*@aist.go.jp*

***Abstract*— In this paper, we propose RT-Middleware for robot system integration. "RT" means "Robot Technology", which is applied not only to industrial field but also to nonindustrial field such as human daily life supporting systems.**

We have studied modularization of RT (Robot Technology) elements and have developed RT-Middleware, which promotes application of RT in various field[1], [2], [3]. RT-Middleware is a software platform for RT systems and provides RT specific features for the component based system development. The component which constructs RT-systems in RT-Middleware is called the "RT-Compoment".

The RT-Component, which is a basic software unit of RT-Middleware based system integration, is derived from this discussion. A methodology of system development by using RT-Components, and a framework for component development are proposed. Evaluations of some RT-Component based systems is performed. Finally conclusion and future work will be described.

I. Introduction

Robotics has had accumulated knowledge of robotic basic functions and elements enough to realize simple intelligent tasks to make human daily life more convenient. Therefore, research on system integration of a basic robot function is becoming important in robotics in recent years. Robotic research to apply RT (Robot Technology) to full-scale application, which is exemplified by ubiquitous robot[4], intelligent environment[5] and sensor network systems[6], is also active.

From such backgrounds, the necessity for the knowledge of systematic robot system integration and the development of the implementation platform has increased. As shown in Figure 1, the methodology for robot system integration independent from persons' experience and knowhow, and robot system platform to support it are also needed for systematic RT-system development.

We have studied integral elements for robot software platform, and the RT-Middleware has been proposed. The purpose of RT-Middleware is to establish basic technologies for integrating robot systems with a new function easily by using modularized software components.

If robot systems with new functions can be constructed more flexibly, it can satisfy every users' needs individually which cannot be satisfied now. Thus, it is expected that the conventional robot industry mainly restricted to the manufacturing field will be expanded to the nonmanufacturing field like support robots for daily life.

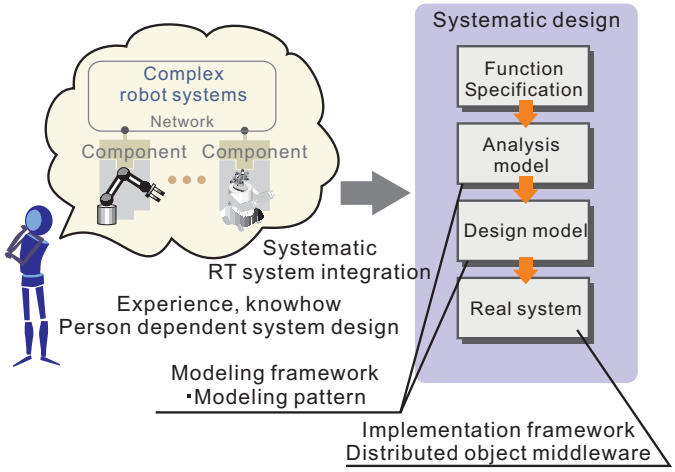


Fig. 1. A Robot Systems Modeling Flow: The RT system should be modeled and designed through systematic design flow independent from the persons' experience and knowhow.

A. Related research

The research on software modularisation of robot functions and development of software libraries for robotics are performed actively in recent years. Table I shows related researches on robot software platforms and its features.

ORiN (Open Resource interface for the Network/Open Robot interface for the Network) is a middleware which offers the standard communication interface over various FA apparatus including robots [7], [8]. **Orocos** (Open Robot Control Software / Open Realtime Control Services) is a free software project for robotics [9], [10]. **ORCA** is a project to develop an Open-Source Robotic Control System derived from OROCOS project [11]. Another **ORCA** (Open Robot Controller Architecture) developed at Toshiba is a HORB (Java ORB developed in AIST) based robot controller architecture [12], [13]. In Toshiba's home robot "ApriAlpha", functional components are integrated and controlled using ORCA. **OPEN-R** is the standard interface for the entertainment robot system that SONY is actively promoting. This interface greatly expands the capabilities of entertainment robots like AIBO, SDR-3X and QRIO[14]. **OSCAR** (Operational Software Components for Advanced Robotics) is an object-oriented framework for the development of control programs for robotic manipulators [15]. **Player/Stage** is a free software

TABLE I
ROBOT PLATFORMS AND MIDDLEWARE.

	Network	Language	Style	Application
RT-Middleware	Open specification.			
OpenRTM-aist	CORBA	C++,Python	CBSE	Any
ORCA	CORBA,CURD,Socket	C++	CBSE	Any
ERSP	Socket	C++,Python	CBSE	Any
Player/(Stage)	Socket	CLL,Java,Tcl,Python	PO	Mobile Robot
ORiN	DCOM	C++,VB	OO	FA, Robot
ORCA(Toshiba)	HORB	Java,Python,C++	OO	Service Robot
Open-R	Socket	C++	OO	AIBO,SDR
CARMEN	IPC	C	PO	Mobile Robot

CBSE (Component Based Software Development), OO (Object Oriented), PO (Procedure Oriented).

project for research in robot and sensor systems. The Player, which is a robot server with robot control interface, and its simulation backends, Stage and Gazebo, are very widely used [17].

B. Goals of RT-Middleware

The purpose of RT-Middleware project is research and development of the middleware for robots, which supports efficient development of robot systems. RT-Middleware aims at the spread of an open robot system architecture to contribute to robot market activation.

Buisiness model of robot market: Until now, only some makers with the synthetic technical capabilities of hardware and software have been able to participate in the robot market. If a standardized robot system architecture spreads, the makers with technical capabilities of software, can also get into the market as "a robot system integrator".

Wide application of RT: The software platform is an infrastructure to improve flexibility of robot system integration. Flexibility of system integration is needed to apply RT to a wide variety of market needs. Component based software development (CBSD) supported by RT-Middleware realizes flexible software and robot system integration.

Robotics research tool: A researcher has only to develop his/her logic or algorithm as a component, and he/she can build a new system by combining with other available components. If a researcher want to try some algorithms to specific part of the system, a researcher has only to replace a related module with new one implemented with his/her new algorithm. Therefore the efficiency of an experiment will improve.

From academic view, robotics research can be shifted to research of integration technology by robot technology componentization.

Since software and hardware are making rapid progress, implemented software platform and framework are eventually becoming obsolete. Therefore robot middleware and its component model should be discussed on metamodel. For the above-mentioned purpose, we defined RT-Middleware object model as an open specification, which is a set of interface and its component model of distributed object middleware for RT functional element. Since the "RT-Middleware" is an open specification, anyone can develop own implementation on any platform in any language.

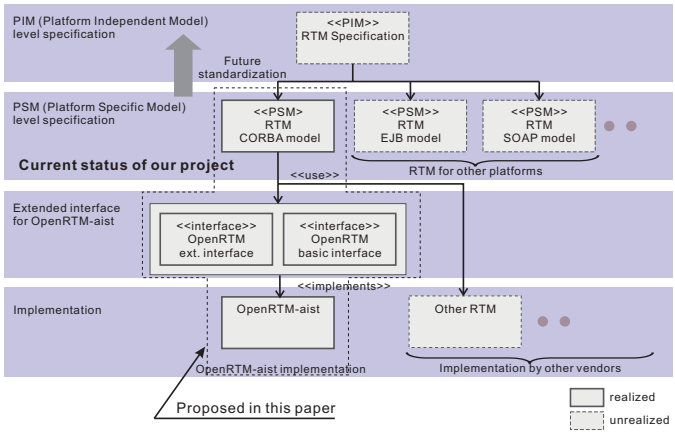


Fig. 2. The RT-Middleware (RTM) standardization process and the relation between RTM specification and our implementation of OpenRTM-aist.

Moreover, an open source implementation named "OpenRTM-aist" has been developed for the purpose of obtaining the feedback from many robot research fields. Figure 2 shows the relation between RTM (RT-Middleware) specification and our implementation of OpenRTM-aist.

II. MODULARIZATION OF RT ELEMENTS

In this section, structural differences in the modularization of the simple distributed object and the RT functional element are clarified, and the core architecture design of RT-module is discussed. We considered what kind of function required for a modularization of the RT element would be realized in the framework of distributed object middleware. As one of the views of a modularization of RT element, "RT-Component" was proposed. Required functions, a structure and a realization method based on distributed objects for the RT-Component were also reviewed.

A. RT specific functions

Granularity of modules: When modularizing an RT element, a module of various granularity size can be considered. Modules of fine granularity level, such as a motor, a sensor, and a controller. Modules of middle-fine granularity level, such as a vision system with some image processing algorithms, a several degrees of freedom manipulator arm and a mobile

robot with some sensors. Modules of rough granularity level, such as a humanoid robot with legs, arms and vision system, an intelligent room with distributed sensors and robots. It is necessary to provide a framework which can choose such various granularity freely in RT-Middleware.

Active module: Usually, a general distributed object works as a passive object that sends back return values to a method invocation. In this case, an object is modeled as interfaces that contain operations with input and output parameters and a return value. An internal activity model of an object is not considered.

On the other hand, an RT element has its own tasks like real-time feedback control. Furthermore, it is necessary to collect required data RT-element itself, or to notify event to other elements when it happened.

Rrealtime capabilities: Realtime capabilities of module activity is a indispensable function in RT systems. RT-Middleware should support realtime capabilities in its software module as a framework.

Realtime capabilities between modules: RT system requires the high speed communication and close cooperation with other modules, such as a servo control.

Time management: For example, a servo control has to be performed under stable periodic real-time task. In order to make two or more modules cooperate in the real-time schedule, the time synchronization between modules, which may be running on distributed host machines, is needed.

Software reuse: Users are unwilling to use the framework which needs to remake all programs. In order to reuse a lot of software library created until now, it is necessary to provide the framework for modularizing the existing software library easily.

Platform independent middleware: In order to improve the reusability of software, the middleware has to be modeled on the platform (in this context, "platform" means operating systems) independent abstraction level.

Network independent middleware: RT-Middleware has to support various communication media and its model should have independent structure from them. If a real-time communication media is available, modules that depend on realtime features should use it.

III. RT-Component Architechture

RT-Component is basic functional unit of RT-Middleware based systems. Figure 3 shows the architecture block diagram of the RT-Component.

We chose CORBA (Common Object Request Broaker Architecture) as distributed object middleware for its platform independency, and tried modeling of the RT-Component on CORBA.

An RT-Component consists of the following objects and interfaces.

- Component object.
- Activity.
- InPort as input port object.
- OutPort as output port object.
- RTComponent Service interface.

The general distributed object model can be described as some interfaces that contain operations with parameters and a return value. On the other hand, the RT-Component model has a component object as a main body, activity as a main process unit, input ports (InPort) and output ports (OutPort) as data stream ports.

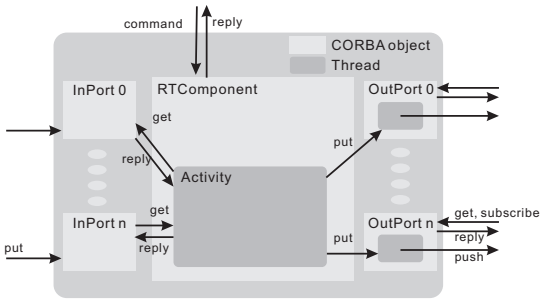


Fig. 3. A proposed architecture of RT-Component. RT-Component has a component object, command interfaces, an activity, InPorts and OutPorts.

A. Activity and state transition

An RT-Component itself has an activity which always continues processing its tasks, and activity serves as a subject of a device control, such as a robot.

The activity of RT-Component has eleven states: `BORN`, `INITIALIZE`, `READY`, `STARTING`, `ACTIVE`, `STOPPING`, `ABORTING`, `ERROR`, `EXITING`, `FATALERROR`, `UNKNOWN`. Figure 4 shows the state transition chart (UML state chart) of RT-Component's activity.

According to the UML notation, RT-Component's method names, which are invoked from clients, are described in each state block. The meaning of method prefixes is the following.

- entry: An atomic action performed on entry to the state.
- do: An action performed while being in the state.
- exit: An atomic action performed on exit from the state.

The states of having only a "entry" method are transient states which change to the next state immediately. The states of having "do" method are steady states which can stay at the state. The states and state transition of the component

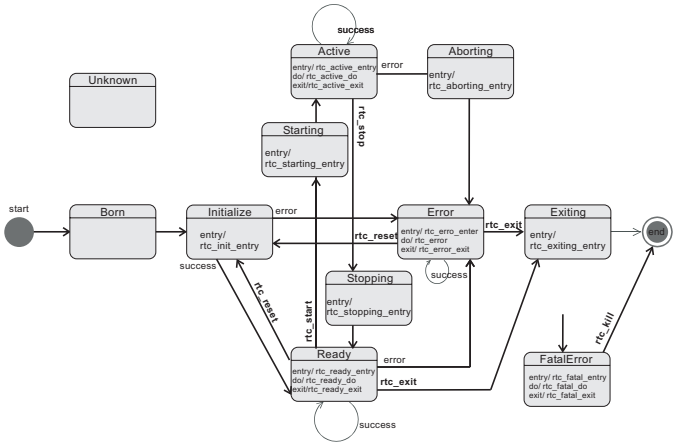


Fig. 4. RT-Component statechart diagram.

was defined so that various type of RT-Components could be

treated as common software parts. By giving a common state transition to RT-Components, and specifying the meaning of states, it is possible to control the action of many components similarly.

B. InPort/OutPort

In the low level real-time control layer, if a component is considered as the functional unit which consists of inputs, processing, and outputs like a control block diagram, it will be easy to perform a system configuration.

This input/output model is not so suitable for general usage of the distributed object model. Because an object which sends its data to other objects has to know all objects' complete interface definition. On the other hand, in such low level control layer, data type, number of data and unit of data are more important than interface definition. RT-Component adopted the publisher/subscriber model and defines it as InPort/OutPort.

An InPort receives data from an OutPort that calls method of "InPort::put()".

An OutPort sends data to InPorts that "subscribes" this OutPort, calling "InPort::put()" as "push" type data exchange. "pull" type data exchange calling "OutPort::get()" method is also provided.

OutPort supports some subscription types, "New", "Once", "Periodic", "Periodic New", "New Periodic", "Triggered", "Triggered Priodic", "Periodic Triggered".

For example, the "New" subscription type means that an OutPort sends data to InPorts which subscribe it when a new data come from the activity. Other subscription type can also be defined if user needs.

C. RTComponent Service

The software component should have enough interfaces to change internal attributes from outside without re-compileing. The object which provides the function attached to the RT-Component is called "RT-Component Service (RTCS)" in RT-Middleware. Component developer can provide his functional object interfaces to other RT-Components by using RT-Component Service feature.

Figure 5 shows whole UML Class Diagram of RT-Component, which depicts the relation among RT-Component, InPort/OutPort and RTComponent Service.

IV. System integration using RT-Components

In a RT-Middleware based system, it will be constructed with two or more RT-Components in some ways.

The following methods are available to construct RT-systems with RT-Components.

- GUI tool
- XML file
- Script language
- Application program

A. Assemble using GUI

Since the software component is based on some isolation principles such as cleanly specified interfaces, interoperability and glue mechanism, it is one of the great merits of component based software development that user can integrate application software using GUI tools.

"rtc-link" is a GUI tool that manages a connection of InPort/OutPort between RT-Components like a control-block diagram and performs activation/deactivation of an RT-Component (Figure 6). "rtc-link" is the powerful tool which can be used in case development of RT-Component and debugging are performed. Moreover, it can use also in verification and an experiment of a robot system, performing the low level integration of components.

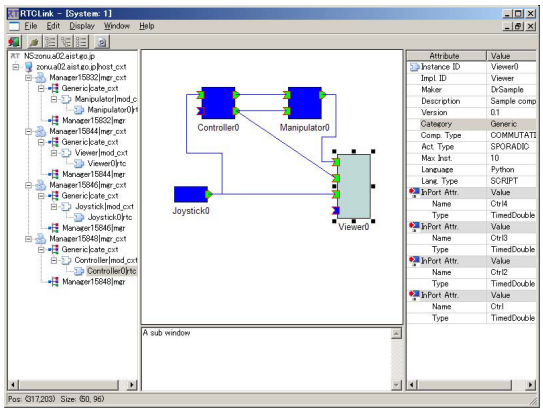


Fig. 6. GUI tool: "rtc-link". The rtc-link is a graphical user interface for RT-Components manipulation.

The Connection information among RT-Components can be saved as XML file. RT-Components connection can be also rebuilt from a saved XML file.

B. Script language

RT-Components are able to be controlled from script language. Now Python language is available in our RT-Middleware implementation. Script languages are useful for rapid prototyping. High level programming language is suitable for high level system integration programming.

C. Application Programs

An RT-Component is a distributed object (in our case it is CORBA object). Usual CORBA based system application program can control RT-Components as CORBA objects.

V. Our implementation and a experiment

A. OpenRTM-aist

"OpenRTM-aist" is prototype implementation based on RT-Middleware interface specification and RT-Component object model. "OpenRTM-aist" consists of an RT-Component development frame work, a manager and some set of tools.

1) RT-Component frame work: RT-Component frame work provides a managed state transition and InPort/OutPort management as frozen spots. A component developer inherits the base class of the RT-Component, and can create his new component class. The state transition logic is implemented in the base class, and a developer only needs to map process to be executed in each state by the specific method override.

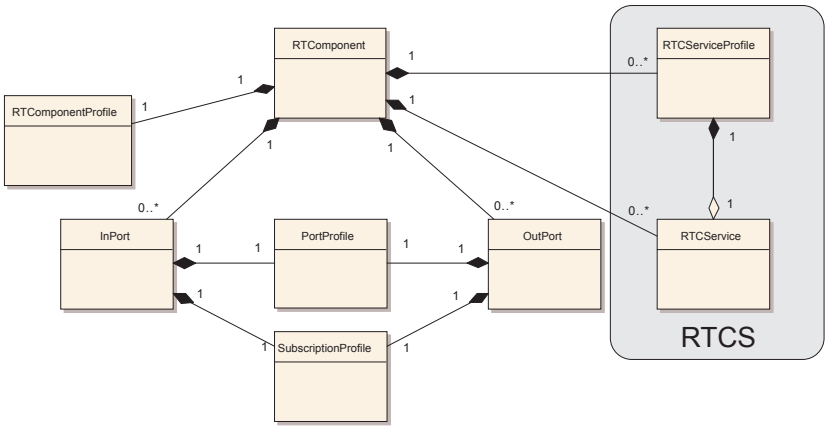


Fig. 5. UML Class Diagram of RT-Component, InPort and OutPort.

2) RT-Component manager: RT-Component manager manages a life cycle of an RT-Component and provide access to CORBA naming service. RT-Component can exist as a loadable module. The loadable module is loaded by a component manager and an instance of an RT-Component is created by the manager. A manager also activates a component as a CORBA object and binds it to a name server. Clients can obtain RT-Component object references from the name server.

3) RT-Component template generator: OpenRTM-aist provides a template source code generator for RT-Component development. The template generator creates C++ and Python source code from given component profiles and InPort/OutPort profiles. (Here, "template" is not C++ template metaprogramming.) C++ source code includes a header file, component source, executable component source code, Makefile and specification file of the RT-Component. A standalone RT-Component executable and a loadable RT-Component module are created from these files.

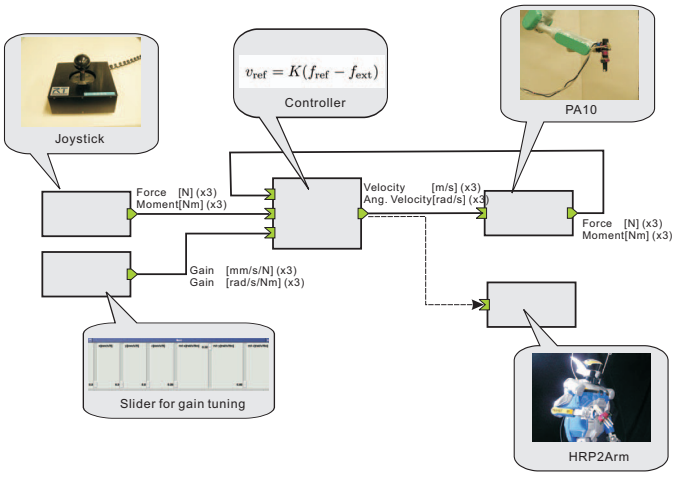


Fig. 7. Manipulator system equipment: Manipulators, an end-effector force/torque sensor, a joystick, a controller and a GUI slider.

B. Experiment

We applied RT-Component to a force controlled manipulator systems.

At first, the following RT-Components were developed.

- Force/torque sensor on endeffector (Nitta, IFS).
- PA10 manipulator component (Mitsubishi HI, PA10).
- HRP2 arm component (Kawada/AIST, Humanoid HRP2 Promet).
- Joystick using force/torque sensor (Nitta, IFS).
- GUI slider gain tuning tool.
- Dumper controller component.

Figure 7 shows experimental setup composed of a manipulator (PA10 and HRP2 arm), a force/torque sensor, a joystick, a controller and a GUI slider component.

1) System I: System I is a basic force controlled manipulator system. A manipulator (PA10) component, an end-effector force/torque sensor component, a joystick component, a controller component and a slider gain tuning component are used. All the components were executed in a same PC (Pentium4, 2.8GHz).

Velocity of the manipulator's end-effector is given by the controller. The controller output is calculated from the following control law.

$$v_{\mathrm{ref}} = K(f_{\mathrm{ref}} - f_{\mathrm{ext}}) \tag{1}$$

Here, f_{ref} is reference force/torque, which are generated on the manipulator's end-effector. f_{ext} is external force which is applied to the manipulator's end-effector. f_{ref} is given by the joystick component which is applied force/torque to the joystick. f_{ext} is given by the end-effector force/torque sensor. K is damper gain given by the slider gain tuning component.

In the experiment, when force was applied to the end-effector's force/torque sensor, it was confirmed that an end-effector moves in the same direction of force. When force was applied to the joystick, the end-effector moved. It stops when the end-effector touched something and the applied force to the joystick, and the contact force between the end-effector and the contact object, were balanced.

2) System II: Next, a HRP2's arm component was developed and replaced with PA10 arm component in the *System I* as shown in Figure 7.

Since the HRP2 arm component InPort was designed by the same specification as PA10 arm component, the PA10 arm

component can be replaced with the HRP2 arm component easily. Although the joystick and the HRP2 are separated physically, the HRP2 arm can be also controlled from joystick because of distributed middleware.

In the experiment, when force was applied to the joystick, it was confirmed that the HRP2 arm moves. Only developing a new arm component with same specification, a new force controled arm system can be developed. Here, other components, which are the joystick, the slider and the controller components, except arm component can be used without any changes in the new system.

3) System III: Next, the controller's output is connected to the PA10 arm component and the HRP2 arm component simultaneously. In this case, the PA10 is controlled same as *System I*. Since the controller output of end-effector velocity is connected to HRP2 arm component simultaneously, two arms' end-effectors move with same velocity. The PA10 manipulator acts as a master manipulator of the HRP2 arm. Only using component used in *System I* and *System II*, a simple teleoperation system was composed without developing any new components.

The point is that these four devices components and two software components are not a monolithic program but programs completely created separately. Since RT-Middleware is constructed on distributed object middleware, each component can be easily distributed on network. RT-Components which have same InPort/OutPort specification can be replaced easily. Moreover, a new system can be composed only developing some new components and using other components which are already developed.

VI. Conclusion

In this paper, we proposed RT-Middleware as a software platform for robot system integration. About the basic functions needed in case RT functional element is modularized were discussed. RT-Middleware interfaces for the distributed object middleware, a component model and OpenRTM-aist as an implementation were introduced.

Simple manipulator system was constructed using RT-Components and the RT-Component usability was evaluated. It was shown that the reusability of software and the flexibility of integration can be improved if RT-Component is used. Moreover, since it becomes possible to handle the existing component as a black box and to combine it, a complicated system can be constructed easily.

In this paper, we focused on general features of RT-Middleware and RT-Component. Therefore, other features and details were not discussed. These are discussed on other papers [3], [2].

Robot specific features, for example coordinate system handling, unit definition and conversion of data and so on are not clearly mentioned in RTM specification and OpenRTM-aist currently. However these features are indispensable for RT-Middleware. In the future work, Some of them will be realized as services, and others will be imported into the RTM specification.

Acknowledgement

This work, supported by NEDO (New Energy and Industrial Technology Development Organization), was performed in the Joint Research Projects: "Development of Core Technology needed for Creation of Robotic Fanctions" with Matsushita Electric Works Ltd. and JARA (Japan Robot Association).

References

[1] Noriaki ANDO, Takashi SUEHIRO, Kosei KITAGAKI, Tetsuo KOTOKU, Woo-Keun Yoon, "RT-Middleware: Distributed Component Middleware for RT (Robot Technology)", 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2005), pp.3555-3560, 2005.08

[2] Noriaki ANDO, Takashi SUEHIRO, Kosei KITAGAKI, Tetsuo KOTOKU, Woo-Keun YOON, "RT-Component Object Model in RT-Middleware - Distributed Component Middleware for RT (Robot Technology) -", 2005 IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA2005), p.We-B2-5, 2005.06

[3] Noriaki Ando, Takashi Suehiro, Kousei Kitagaki, Tetsuo Kotoku and Woo-Keun Yoon, "Composite Component Framework for RT-Middleware (Robot Technology Middleware)", IEEE/ASME International Conference on Advanced Intelligent Mechatronics(AIM), 2005

[4] Young-Guk Ha, Joo-Chan Sohn and Young-Jo Cho, "Service-Oriented Integration of Networked Robots with Ubiquitous Sensors and Devices Using the Semantic Web Services Technology", 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2005), pp.413-418, 2005.08

[5] Yasushi Nakauchi, Tsukasa Fukuda, Katsunori Noguchi and Takashi Matsubara, "Intelligent Kitchen: Cooking Support by LCD and Mobile Robot with IC-Labeled Objects", 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2005), pp.2464-2469, 2005.08

[6] Taketoshi Mori, Aritoki Takada, Hiroshi Noguchi, Tatsuya Harada and Tomomasa Sato, "Behavior Prediction Based on Daily-Life Record Database in Distributed Sensing Space", 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2005), pp.1833-1839, 2005.08

[7] M.Mizukawa, H.Matsuka, T.Koyama, T.Inukai, A.Noda, H.Tezuka, Y.Noguchi, N.Otera, "ORiN Open robot Interface for the Network – The Standard Network Interface for Industrial robots and its Applications –", ISR2002, No.45

[8] Makoto Mizukawa, Hideo Matsuka' Toshihiko Koyama, Toshihiro Inukai, Akio Noda, Hirohisa Tezuka, Yasuhiko Noguchi, Nobuyuki Otera, "ORiN: Open Robot Interface for the Network – The Standard and Unified Network Interface for Industrial Robot Applications –", SICE Annual Conference 2002, pp.1160-1163, Osaka

[9] Orocos: Open Robot Control Software. http://www.orocos.org

[10] C. Schlegel, R. Worz, "The Software Framework SmartSoft for Implementing Sensorimotor Systems", IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS '99, pp.1610-1616, Kyongju, Korea, October '99

[11] http://orca-robotics.sourceforge.net/about.html

[12] Fumio OZAKI, "Open Robot Controller Architecture (ORCA)", IROS2004 Workshop on Robot Middleware toward Standards,

[13] Fumio OZAKI, "Open robot controller archtecture (ORCA)", AIM2003 Workshop: Middleware Technology for Open Robot Architecture

[14] Kohtaro SABE, "Open-R : An Open Architecture for Robot Entertainment" , AIM2003 Workshop: Middleware Technology for Open Robot Architecture

[15] Kapoor, C. Tesar, D., "A Reusable Operational Software Architecture for Advanced Robotics", Proceedings of the Twelfth CSIM-IFToMM Symposium on theory and Practice of Robots and Manipulators, Paris, France, July 1998.

[16] Minsu Jang, Jaehong Kim, Meekyoung Lee and Joo-Chan Sohn, "Ubiquitous Robot Simulation Framework and Its Applications", 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2005), pp.3213-3218, 2005.08

[17] Brian P. Gerkey, Richard T. Vaughan, Kasper Stoy, Andrew Howard, Gaurav S. Sukhatme, and Maja J Mataric, "Most Valuable Player: A Robot Device Server for Distributed Control", Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2001), pp.1226-1231, 2001.11.