IROS 2005

IEEE / RSJ International Conference on Intelligent Robots & Systems

CONFERENCE DIGEST

Co-sponsors: IEEE Robotics and Automation Society, IEEE Industrial Electronics Society, Robotics Society of Japan, Society of Instruments and Control Engineers, and New Technology Foundation
Technical Co-sponsor: The Institute of Control, Automation and Systems Engineers

Shaw Conference Centre • Edmonton, Alberta, Canada • August 2–6, 2005

2005 IEEE/RSJ International Conference on Intelligent Robots and Systems

RT-Middleware: Distributed Component Middleware for RT (Robot Technology)

Noriaki Ando, Takashi Suehiro, Kosei Kitagaki, Tetsuo Kotoku and Woo-Keun Yoon
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, wk.yoon*}*@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 support systems. RT-Middleware which is proposed in this paper is a software platform for RT systems. We have studied modularization of RT elements and have developed RT-Middleware, which promotes application of RT in various field. Robotic system development methodology and our RT-Middleware concepts is discussed. 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.**

***Index Terms*— RT (Robot Technology), software component, middleware, robot system, system integration**

I. Introduction

The researches on the system integration of a basic robot function are becoming important in robotics in recent years. Robotics has already had the accumulated knowledge of robotic basic functions enough to realize simple intelligent tasks to make human daily life more convenient. Robotic researches that try to apply robotic functional elements to full-scale application are also active. Intelligent environment and ubiquitous computing are potential full-scale application for robotic system integration.

From such backgrounds, the necessity for the knowledge for systematic robot system integration 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.

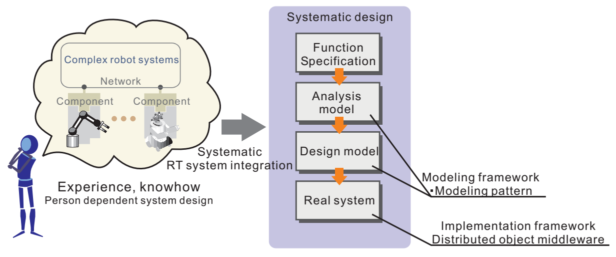


Fig. 1. A Robot Systems Modeling Flow: The RT system should be modeled and designed through systematic design flow independent from the persons' experience and knowhow.

We started RT-Middleware project from 2002 under NEDO's (New Energy and Industrial Technology Development Organization) "Robot challenge program". Basic functions for robot software platform which supports complex robot system integration have been studied. The purpose of this project is to establish basic technologies for integrating robot systems with a new function easily by modularized software components which constitutes robot systems.

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.

A. Related research

Research about software modularisation of robot functions and development of software libraries for robotics are performed actively in recent years.

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 [1], [2]. The specification of ORiN is released from the ORiN consortium, in which almost all the Japanese industrial-robots maker has affiliated. **Orocos** (Open Robot Control Software / Open Realtime Control Services) is a free software project for robotics [3], [4]. However it has outgrown its robotics-dependent roots. "Open Robot Control Software" is a set of class libraries and an application framework offering generic functionality for machine tools and robots. "Open Realtime Control Services" is a hard realtime kernel for all possible feedback control applications, fully independent of the project's original robotics focus. **ORCA** (Open Robot Controller architecture) developed in Toshiba is a HORB (Java ORB developed in AIST) based robot controller architecture [5], [6]. In Toshiba's home robot "ApriAlpha", functional components are integrated and controlled using ORCA. They expects ORCA to play a key role in advancing the use of robots in a variety of fields. **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. AIBO's reconfigurability is realized because software and hardware are completly modularized by Open-R.

On the other hand, one of our purpose is to define a set of standard interface of the software component for robots, which makes interconnection possible. Since the standard interface specification is free and open, any vendors can implement middleware based on this interface specification. To provide open-source middleware based on the interface is another purpose. Getting feedback from actual application use, improvement of the specification will be advanced. Final target is to establish a systematic robot system design theory derived from the knowledge of component-oriented robot system integration.

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, a maker with hardware technical capabilities can get into the robot market as a robot device component vendor. The makers with technical capabilities of software, can also get into the market as "a robot system integrator".

Wide application of RT: RT-Middleware is a robot system platform. 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.

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.

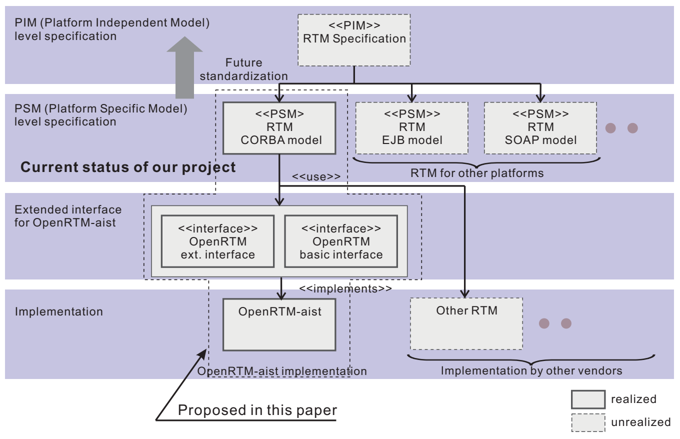

PIM (Platform Independent Model) level specification
<<PIM>> RTM Specification
Future standardization
PSM (Platform Specific Model) level specification
<<PSM>> RTM CORBA model
<<PSM>> RTM EJB model
<<PSM>> RTM SOAP model
Current status of our project
<<use>>
RTM for other platforms
Extended interface for OpenRTM-aist
<<interface>> OpenRTM ext. interface
<<interface>> OpenRTM basic interface
<<implements>>
Implementation
OpenRTM-aist
Other RTM
OpenRTM-aist implementation
Implementation by other vendors
Proposed in this paper
realized
unrealized

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

For the above-mentioned purpose, we defined a set of interface and its component model of distributed object middleware for RT functional element. 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.

In this paper, we mention about RT-Middleware interfaces for the distributed object middleware, a component model and OpenRTM-aist as an implementation. First, we will make a discussion about the basic function which is needed in case an RT functional element is modularized. An outline about the basic structure of the RT-Component derived from this discussion will also be mentioned. Next, the architecture of the core part of the RT-Component will be shown. Moreover, how to make two or more components cooperate and constitute one system will be explained. A force control manipulator system which was implemented using OpenRTM-aist based on these ideas would be shown, and finally a conclusion is described.

II. Modularization of RT elements

Only by implementing RT element as a distributed object simply, a modularization of RT element is unrealizable.

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, which 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 architecture

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

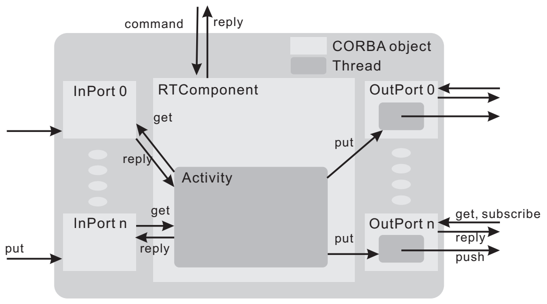


Fig. 3. A proposed architecture of RT-Component. RT-Component has a component object, command interfaces, an activity, InPorts and OutPorts.

A. RT-Component object model

For the above-mentioned reason of platform independency, we chose CORBA (Common Object Request Broaker Architecture) as distributed object middleware, 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.
- Command interfaces.

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.

B. 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

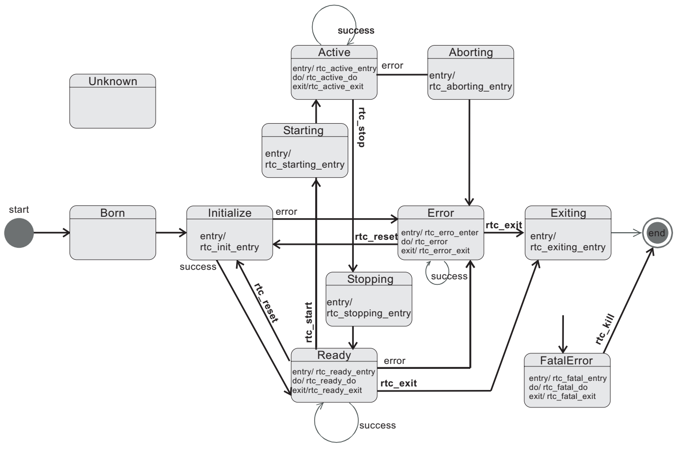


Fig. 4. RT-Component statechart diagram.

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. A component developer has only to map his/her algorithm's process or library's process into each RT-Components state, and just insert his/her code to the RT-Component framework.

C. 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 so that it may be exactly expressed with 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.

1) InPort object: An input port of RT-Component. An InPort receives data from an OutPort that calls method of "InPort::put()". This is basic function of the InPort.

Other functional InPorts, that raise a signal or invoke a callback method etc., can be implemented as subclasses of the InPort.

2) OutPort object: An output port of RT-Component. 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 supported.

OutPort supports some subscription type, "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. Due to insufficient space, the details of all subscription type cannot be described. Other subscription type can also be defined if user needs.

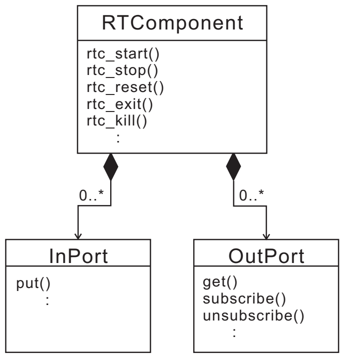


Fig. 5. UML Object Diagram of RT-Component, InPort and OutPort.

As shown in UML Object Diagram of Figure 5, the relation between RT-Component object and InPort/OutPort objects is the composition. An RT-Component object manages object creation and destruction of InPort and OutPort. Other object or software can ask the RT-Component what kind of InPort/OutPort it has.

D. Composite component

On the RT-Middleware, various granularity RT-Components will be provided by component developer. In this case, such a composite structure or a nested structure are useful for hierarchical robot system integration. To realize the composite structure, the composite pattern is applied to the RT-Component object structure.

The composite components are roughly divided into "the asynchronous composite component" and "the synchronous composite component". The composite components have the following features,

- A composite component can include components to manage them.
- Internal components' InPorts/OutPorts are delegated to the composite component.
- A composite component manages activity states of intertnal components.

Moreover, the synchronous composite component has the following features,

- Activity states of internal components are completely synchronized.
- Activities of internal components are performed serially in preconfigured order.
- If a thread that invokes each internal component's activity is running in real-time mode, and the response time boundary of method invocation is given and is finite, internal components can be a real-time control task.

The synchronous composite component architecture is utilized to realize real-time composite components [9]. Due to insufficient space, the details of real-time composite component cannot be described.

The basic asynchronous composite component has the following features,

- States of an internal components do not necessarily have a synchronization.
- Activities of internal components are performed in parallel.

Some types of the asynchronous composite component are possible by the state transition handling type between internal components and the composite component.

IV. System Integration Using RT-Components

The composite components are one of the low level component integration method. For higher level such as application layer, there are some integration methods provided in RT-Middleware shown in Figure 6.

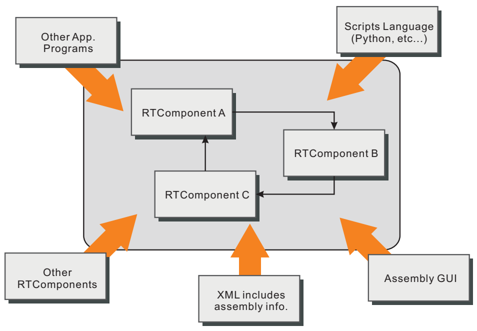


Fig. 6. RT-Component system integration methods.

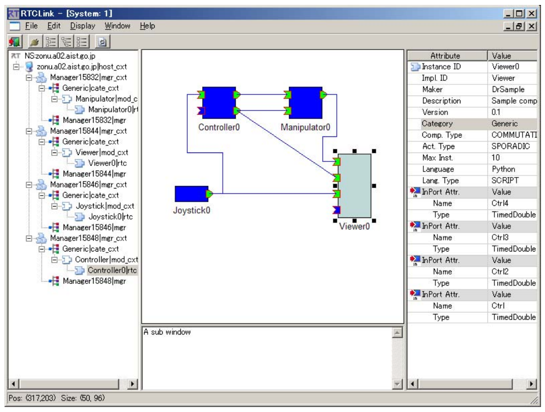
The following methods are available.

- Assembly GUI tool
- Script language
- XML file

- Other RT-Components
- Other application program

A. *Assemble using GUI*

"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 7). "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.



```
NS:zonu.a02.aist.go.jp
  zonu.a02.aist.go.jp|host_cxt
    Manager15832|mgr_cxt
      Generic|cate_cxt
        Manipulator|mod_c
          Manipulator0|r
      Manager15832|mgr
    Manager15844|mgr_cxt
      Generic|cate_cxt
        Viewer|mod_cxt
          Viewer0|rtc
      Manager15844|mgr
    Manager15846|mgr_cxt
      Generic|cate_cxt
        Joystick|mod_cxt
          Joystick0|rtc
      Manager15846|mgr
    Manager15848|mgr_cxt
      Generic|cate_cxt
        Controller|mod_cxt
          Controller0|rtc
      Manager15848|mgr
```

Attribute	Value
Instance ID	Viewer0
Impl. ID	Viewer
Maker	DrSample
Description	Sample comp
Version	0.1
Category	Generic
Comp. Type	COMMUTATI
Act. Type	SPORADIC
Max Inst.	10
Language	Python
Lang. Type	SCRIPT
InPort Attr.	Value
Name	Ctrl4
Type	TimedDouble
InPort Attr.	Value
Name	Ctrl3
Type	TimedDouble
InPort Attr.	Value
Name	Ctrl2
Type	TimedDouble
InPort Attr.	Value
Name	Ctrl
Type	TimedDouble

```
Pos: (317,203)  Size: (50, 96)
```

Fig. 7. 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. *Others*

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.

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.

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

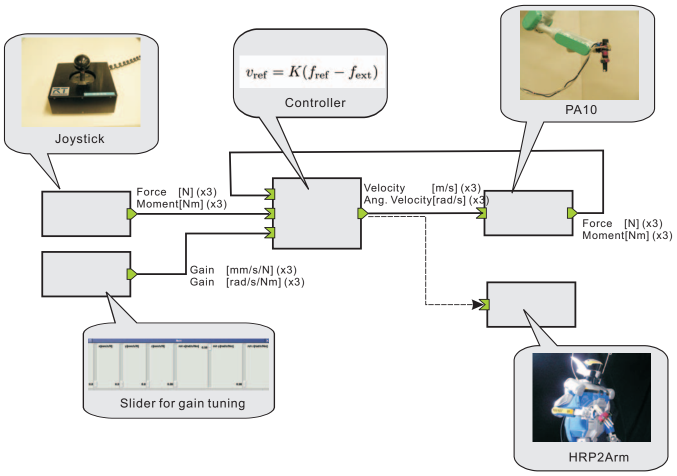


Fig. 8. Manipulator system equipment: Manipulators, an end-effector force/torque sensor, a joystick, a controller and a GUI slider.

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 low.

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

Here, f_{ref} is reference force/torque, which should be 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 dumper 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 8.

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 tele-operation 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 [9], [10].

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] 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

[2] 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

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

[4] 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

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

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

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

[8] Masahiko NARITA, "RoboLink Protocol - A Robot Collaboration Protocol based on Web Services - " IROS2004 Workshop on Robot Middleware toward Standards

[9] 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

[10] Noriaki Ando, Takashi Suehiro, Kousei Kitagaki, Tetsuo Kotoku and Woo-Keun Yoon, "RT-Middleware: Distributed Component Middleware for RT (Robot Technology)", IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005