IEEE

Program

CIRA2005

June 27–30, 2005 Espoo, Finland

RT-Component Object Model in RT-Middleware
– Distributed Component Middleware for RT (Robot Technology) –

Noriaki Ando, Takashi Suehiro, Kousei 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*— This paper proposes RT-Component object model in 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 is a software infrastructure for RT systems. We have studied modularization of RT elements at software level. For that reason, RT-Middleware, which promotes application of RT in various field, have been developed. Robotic system development methodology and our RT-Middleware concepts will be discussed. RT-Component, which is a basic software unit of RT-Middleware based systemintegration, is derived from that discussion. Next, the object model and the interface definition of RT-Component architecture will be discussed. Finally conclusion and future work will be described.**

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

I. Introduction

Robotics research is making the transition from analysis to synthesis and integration(Figure 1). We have already had a lot of robotic basic functions enough to realize simple intelligent tasks, which makes human daily life more convenient. The research on the system integration of a basic robot function is becoming important in robotics in recent years.

Robotic researches, that try to apply robotic functional elements to full-scale application, are also active. Intelligent environment and ubiquitous computing are typical example of full-scale application for robotic system integration.

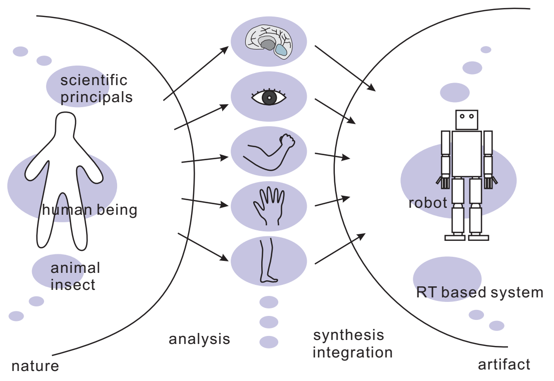


Fig. 1. The transition from analysis to synthesis. The research on the system integration of a basic robot function is becoming important in robotics.

From such backgrounds, the necessity for the systematic knowledge for robot system integration has increased. As shown in a Figure 2, the methodology for robot system integration independent from persons' experience and knowhow, and robot system platform to support it is also needed.

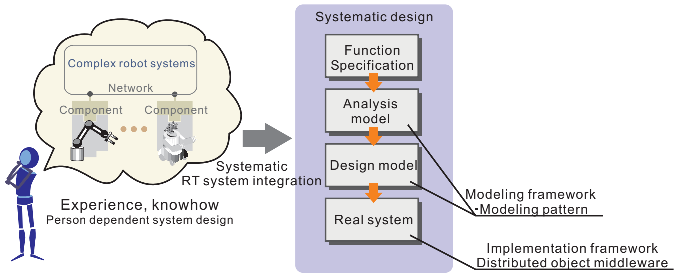


Fig. 2. A Robot Systems Modeling Flow: The RT system should be modeled and designed through systematic design flow independent from the persons' experience and knowhow.

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 easily integrating robot systems having new functions by modularized software components.

If robot systems with new functions can be constructed more flexibly, every users' needs, which cannot be satisfied now, will be satisfied individually. 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 of software modularisation of robot functions and development of software libraries for robot system integration 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 (factory automation) equipment including a robot [1], [2].

Orocos is the free software project that includes a set of class libraries and application framework, and a hard realtime kernel for all possible feedback control applications [3], [4].

ORCA (Open Robot Controller architecture) developed in Toshiba is HORB (Java ORB developed in AIST) based robot controller architecture [5], [6]. SONY is actively promoting the **OPEN-R** which is the standard interface and platform for the entertainment robot system [7].

On the other hand, one of our purpose is to define the standard software component interface that makes interconnection possible among robots. 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 those interface specifications is another important purpose. Getting feedback from actual research use and application use, it is expected that the improvement of the interface 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. 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.

B. *Goals of RT-Middleware*

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

Buisiness model of robot market: Until now, only some makers with the synthetic technical capabilities of hardware and software have participated in the robot market. When 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 maker with software technical capabilities can also get into the market as "a robot system integrator".

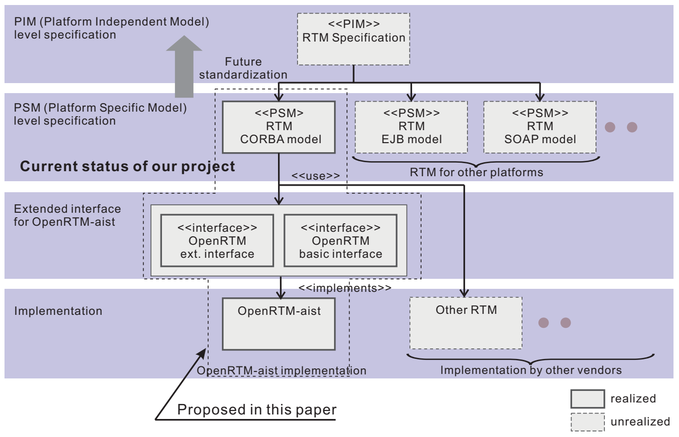
Wide application of RT: The RT-Middleware is a robot system platform. The software platform is an infrastructure to improve flexibility of robot system integration.

Robotics research tool: A researcher can be concentrated on its subject of his/her research.

A highly modularized component can be used as a black box. A researcher has only to develop his logic or algorithm as a component, and can build a system by combining with other available components. When a researcher want to try some algorithms to specific part of the system, since a researcher has only to replace a related module with new one implemented his new algorithm, the efficiency of experiments will be improved.

Research on robot system integration: From academic view, robotics research can be shifted to research of integration technology by robot technology componentization. The knowledge about the system integration, which have been missing in robot system construction until now, can be stored.

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 3 shows the relation between RTM specification and our "OpenRTM-aist" implementation.



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

In this paper, we mention about the RT-Middleware interfaces for the distributed object middleware, the RT-Component object 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. The RT-Component object model derived from this discussion will also be mentioned. To evaluate component based system development, a force control manipulator system which is implemented using OpenRTM-aist based on these ideas will be shown. The result of the experiment will be reviewed and future works will be described in the conclusion.

II. Modularization of RT elements

Only by simply implementing RT element as a distributed object, 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-Component 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 module: When modularizing a RT element, a module of various granularity size can be considered. Modules of fine granularity level, such as a motor, a sensor, a camera, and a controller. Modules of middle-fine granularity level, such as a vision system with some image processing, 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 a 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.

Rrealtimeness: Realtimeness of module activity is an indispensable function in RT systems. RT-Middleware should support realtimeness in its software module as a framework.

Realtimeness between modules: An 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 system 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 a 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: The 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 realtimeness should use it.

B. *RT-Component model*

For the above-mentioned reason, we chose CORBA as distributed object middleware, and tried modeling of RT module on CORBA. We propose the RT-Component, as a RT module unit model based on the distributed object model.

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.

III. RT-Component architecture

Figure 4 shows the architecture block diagram of the RT-Component.

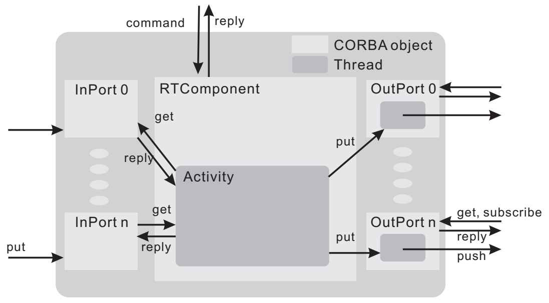


Fig. 4. The proposed architecture of the RT-Component. An RT-Component has component object, command interface, activity, InPorts and OutPorts.

A. *Activity and state transition*

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

B. *RT-Component object model*

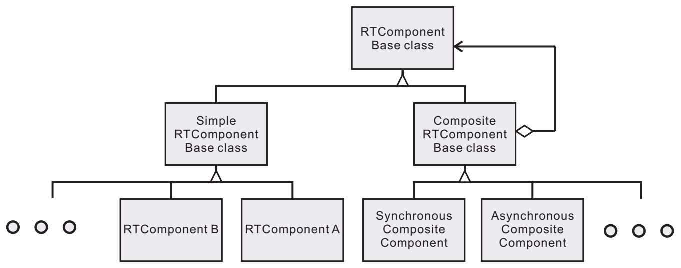

RTComponent Base class
Simple RTComponent Base class
Composite RTComponent Base class
RTComponent B
RTComponent A
Synchronous Composite Component
Asynchronous Composite Component

Fig. 5. The RT-Component class hierarchy. User defined component class inherits simple RT-Component base class. The composite component has RT-Component itself by the composite pattern.

Figure 5 shows the hierarchy of RT-Component classes.

Simple RT-Component base class: In figure 5, the simple RT-Component base class is the base class of each new RT-Component class which is created by the component developer. A component developer can develop his/her component class by inheriting from the RT-Component base class.

Composite RT-Component base class: 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 component states and state transition of the component 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. Various granularity components can be treated similarly, by developing a component according to this state definition. It becomes possible to realize the composite component, which is nested components and grouped components, by defining the common component state transition.

1) RT-Component state: The activity of RT-Component has ten states: `BORN, INITIALIZE, READY, STARTING, ACTIVE, STOPPING, ABORTING, ERROR, EXITING, FATALERROR, UNKNOWN`. Figure 6 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 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 which have only a "entry" method are transient states, which changes to the next state immediately. The states which have "do" method are steady states, which can stay at the state.

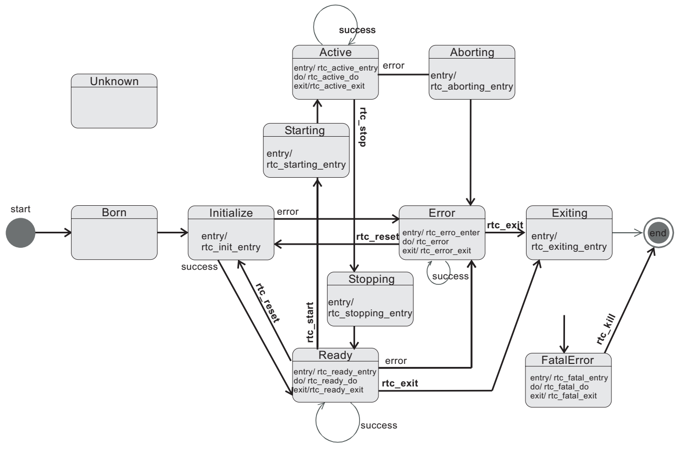


Fig. 6. RT-Component statechart diagram.

A component developer has only to map his/her algorithm or library into each RT-Components state, and should 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 method invocation. Because the 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. Therefore RT-Component adopted the publisher/subscriber model [8], [9] and defines it as InPort/OutPort.

The publisher/subscriber model supports an asynchronous communication among many-to-many objects, in contrast to the synchronous style of object method invocation. Published information is forwarded eventually to all subscribers, either immediately when being published (push) or on demand when a subscriber asks for updates (pull).

```
interface InPort
{
  void put(in any data) raises(Disconnected);
  readonly attribute PortProfile profile;
};
```

Fig. 7. The InPort interface definition.

```
interface OutPort
{
  any get();
  RtmRes subscribe(in InPort in_port,
                   out SubscriptionID id,
                   in SubscriberProfile profile);
  RtmRes unsubscribe(in SubscriptionID id);
  readonly attribute InPortList inports;
  readonly attribute PortProfile profile;
};
```

Fig. 8. The OutPort interface definition.

1) InPort object: The InPort object is a input port of RT-Component. Figure 7 shows the interface definition (CORBA IDL) of the InPort. An InPort receives data from OutPorts 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: The OutPort object is a output port of RT-Component. The output port named OutPort interface definition (CORBA IDL) is shown in Figrue 8. Data connection channel is created by the "OutPort::subscribe()" operation with an InPort object's reference, a subscription id (UUID) and a profile of the subscriber. The 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.

An unique ID, which is UUID (Universally Unique IDentifier), is given to each subscription channel. Currently the OutPort supports some subscription type, "Once", "New", "Periodic", "Periodic New", "New Periodic", "Triggered", "Triggered Priodic", "Periodic Triggered" in OpenRTM-aist. When subscription between an InPort and an OutPort is created, one of these subscription types has to be assigned.

Figure 9 shows a sequence diagram of "New", "Periodic", "Periodic New" and "New Periodic" subscription type.

For example, the "New" subscription type means that an OutPort send data to an InPort which subscribes it when new data come from the activity. In the "Periodic" subscription type, an OutPort pushes data from the activity to subscriber InPorts in constant period Δt. In the "Periodic New" subscription type, an OutPort pushes data to InPorts in constant period Δt except when data were not updated. In the "New Periodic" subscription type, an OutPort pushes data to InPorts in constant period Δt and its time base is arrival time of the

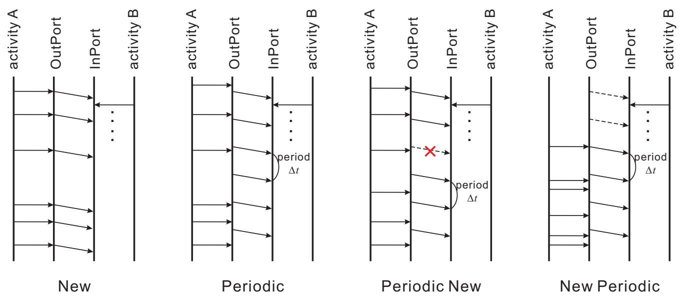


Fig. 9. Subscription Types of OutPort.

first new data.

In these subscription types, the trigger event of sending data to InPorts is the new data arrival. "Triggered", "Triggered Priodic", "Periodic Triggered" are another subscription types. In these subscription types, user can define the trigger event sending data to InPorts.

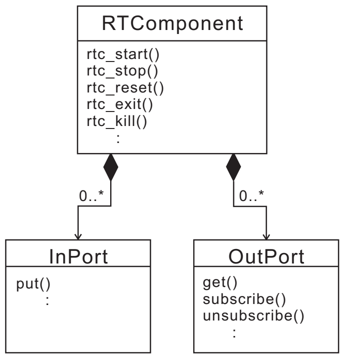


Fig. 10. UML Object Diagram of RT-Component, InPort and OutPort.

As shown in UML object diagram of Figure 10, the relation between an RT-Component object and Inport/OutPort objects is a composition. An RT-Component object manages object creation and destruction of InPorts and OutPorts. Other object or software can ask the RT-Component what kind of InPort/OutPort it has.

D. *Composite component*

The composite components are roughly divided into "an asynchronous composite component" and "a 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 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 components are possible by the state transition handling type between internal components and the composite component.

IV. Our implementation and a experiment

A. *OpenRTM-aist*

"OpenRTM-aist" is prototype implementation based on RT-Component interface definition and RT-Component object model. "OpenRTM-aist" consists of a 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, InPort/OutPort management and simplified development. A component developer inherits the base class of RT-Component, and can create his new component class. The state transition logic, which is frozen spot, is implemented in the base class. A developer can map process, which is hot spot, to be executed in each state by the specific method override.

2) RT-Component manager: RT-Component manager manages a life cycle of a 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 a 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 can generate C++ and Python source code from given component profiles and InPort/OutPort profiles. (Here, "template" does not mean 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 system for evaluation of RT-Component based system development and its performance.

At first, the following RT-Components were developed.

- Force/torque sensor on endeffector (Nitta)
- Manipulator (Mitsubishi HI, PA10)

- Joystick using force/torque sensor (Nitta)
- Dumper controller

Figure 12 shows force/torque sensor on endeffector, manipulator, joystick.

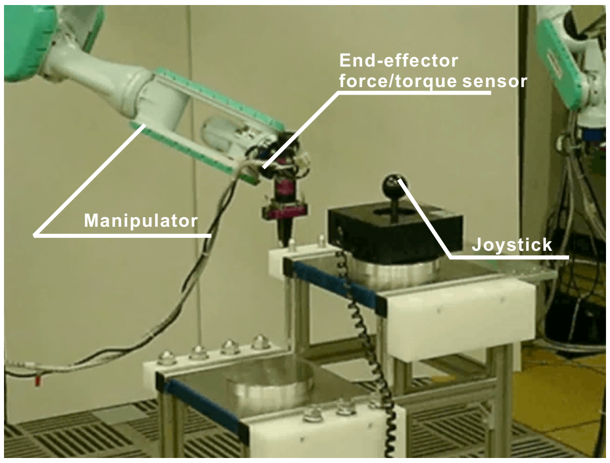


Fig. 11. Manipulator system equipment: End-effector force/torque sensor, manipulator, joystick.

Each RT-Component is build as a standalone executable component, and connected each other as shown in figure 12.

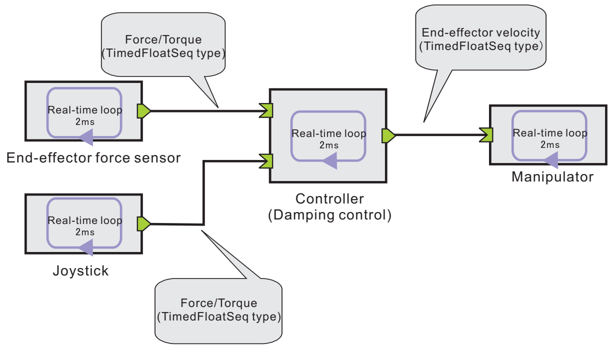


Fig. 12. A force controlled manipulator system using RT-Components.

Each component activity was executed as a 2 ms periodic task in real-time. Component main process is independent from each other, and is executed in parallel. All the components were executed in a same PC (Pentium4, 2.8GHz) on ARTLINUX. ARTLINUX is one of real-time Linux developed at AIST (former ETL) in Japan.

In the experiment, when force was applied to end-effector force/torque sensor, it was confirmed that an end-effector position moves in the direction of force.

When force was applied to the joystick, it was confirmed that the end-effector position of the manipulator moves similarly. It was confirmed that force control is performed stably in these case.

The point is that these three device components and one control component are not a monolithic program but programs completely created separately.

V. Conclusion

In this paper, we proposed RT-Middleware 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 intdroduced.

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 the experiment, it was shown that each component can be executed in real-time. Since each component was running asynchronously, however, the force controled manipulator system is not real-time system in a narrow sense. This problem will be solved by using real-time synchronous composite component framework.

Robot specific features, for example coordinate system handling, unit definition and conversion of data, etc.. are not clearly mentioned in RTM specification and OpenRTM-aist currently. However these features are indispensabe for RT-Middleware. 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, 2002
[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, 1999
[5] Fumio OZAKI, "Open Robot Controller Architecture (ORCA)", IROS2004 Workshop on Robot Middleware toward Standards, 2004
[6] Fumio OZAKI, "Open robot controller archtecture (ORCA)", AIM2003 Workshop: Middleware Technology for Open Robot Architecture, 2003
[7] Kohtaro SABE, "Open-R : An Open Architecture for Robot Entertainment" , AIM2003 Workshop: Middleware Technology for Open Robot Architecture, 2003
[8] Ragunathan Rajkumar, Mike Gagliardi and Lui Sha, "The Real-Time Publisher/Subscriber Inter-Process Communication Model for Distributed Real-Time Systems: Design and Implementation", Proceedings of the Real-Time Technology and Applications Symposium (RTAS'95), pp.66-75, 1995
[9] J. Kaiser and M. Mock, "Implementing the Real-Time Publisher/Subscriber Model on the Controller Area Network (CAN)", Second IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, pp.172-181, 1999