A Software Platform for Component Based RT-System Development: OpenRTM-Aist

Noriaki Ando, Takashi Suehiro, 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,t.kotoku}@aist.go.jp,
http://www.openrtm.org/

Abstract. This paper proposes the 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. We have studied modularization of RT elements and have developed software platform 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 madular unit of RT-Middleware based system integration is derived from this discussion. A methodology of system development with RT-Components, and a framework to make component are shown.

1 Introduction

The progress of robotics research has accumulated vast amounts of knowledge and technology. Those technologies called "Robotic Technology (RT) [1]" have begun to be applied to various field including ubiquitous computing, intelligent room and service robot applications. However the applications of those technologies are not developed enough, and the system integration issues for those technologies are receiving increasing attention both by academia and industrial circles. Especially software takes the lead in robotic system integration methodology. As the supportive evidence of it, many software platforms for robots have been developed in the world in recent years.

We have studied software building block architecture for robot development, and the RT-Middleware (RTM) and RT-Component (RTC) has been proposed as the one of solution about it [2]. The purpose of the RT-Middleware is to establish basic technologies for integrating robot systems with a new function easily by using modularized software components named RT-Component. 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.

The research on software platforms and libraries for robotic systems are performed actively in recent years. "Player/Stage" is a free software project for

S. Carpin et al. (Eds.): SIMPAR 2008, LNAI 5325, pp. 83–94, 2008.

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 especially by mobile robotics researchers [3]. "ORCA" is an open-source framework for developing component-based robotic systems. It provides the means for defining and developing the software components as the building-blocks [4]. "CLARAty" (Coupled-Layer Architecture for Robotic Autonomy) is an integrated framework for reusable robotic software developed by JPL, University of Minesota and Carnegine Mellon University [5]. It defines interfaces for common robotic functionality and integrates multiple implementations of any given functionality. "MSRDS" (Microsoft Robotics Developer Studio) is software platform for robotics that is distributed by Microsoft. This platform is based on DSSP (Decentralized Software Services Protocol) that is SOAP based application protocol for lightweight services. This platform also provides Visual Programming Language (VPL) for robot developers.

The main differences between those software platforms and our RT-Middleware are characterized by the open specification and interoperability. The RT-Component, which is managed its lifecycle by the RT-Middleware, is a software component based on the open specification. Since the specification is opened, any software vendors can implement based on it, and because of the common specification, different implementation can be interoperable. We also have implemented RT-Middleware and RT-Component framework based on the specification, and the implementation named "OpenRTM-aist" is provided as an open source reference implementation.

In the following, first, the requirement of the software platform for component based RT-system development is discussed, and the basic concept of RT-Middleware is shown. Then, on the basis of the discussion, a component model of RT-Component is shown. Based on the proposed component model, the RT-Middleware and RT-Component framework is implemented. Finally, some RT-Middleware based systems are shown and the discussion and conclusion are given.

2 What is Needed for RT Software Platform

In this section, the core architecture of the RT-Componet is discussed. In consideration of RT-specific features for software, the requirement for the component model for RT systems is clarified.

2.1 Code Reusability

The reusability has two meanings. One is reusability of user's code, the other is reusability of components. Users are unwilling to use the component framework which needs to remake all codes. In order to reuse a lot of software library developed until now, it is necessary to provide the framework for modularizing the existing software library easily. Therefore, framework needs to support various operating systems and various programming languages. After modularized

as an component, the component should be used without any modification and re-compile of codes.

OpenRTM-aist provides a component framework and template code generator. User can easily embed their code in it and can make it reusable component. Since component framework provides various functionality such as lifecycle management, network communication including data-oriented and service-oriented interaction and runtime configuration, user can focus on his/her own main logic.

2.2 Various Granularity Support

Various granularity size of modules could be considered, when modularizing an RT element. A motor, a sensor, and a controller can be a fine granularity component respectivly. A vision system with some image processing algorithms, a several degrees of freedom manipulator arm and a mobile robot with some sensors are example of middle-fine granularity components. An application program might want to handle a humanoid robot, an intelligent room, etc. as a coarse grained module respectivly. The software platform needs to support various component grain size in the framework.

OpenRTM-aist's component model provides data-centric interaction method, which is mainly used for fine grained components, and service-oriented interaction method, which is mainly used for coarse grained components.

2.3 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, some of modularized RT element has its own tasks like real-time feedback control in it, and it is necessary to collect required data RT-element itself, or to notify event to other elements when it happened.

In the RT-Component model, a component's business logic is associated with at least one execution entity named Execution Contexts. The Execution Context, which is a logical thread, executes user's logic implemented in the RT-Component framework.

2.4 Realtime Capabilities

Realtime capabilities of module activity are an indispensable function in RT systems especially in low level control layer. It is necessary not only in one component but also in composite component that is composed of some fine-grained components. For example, in order to make two or more modules cooperate in the real-time schedule, the time synchronization between modules is needed. Software platform for RT systems should satisfy these requirements.

Above mentioned Execution Context, which is a logical thread, is associated with RT-Component in run-time. Replacing by a execution context driven by a real-time thread, real-time execution of the RT-Component's can be possible.

2.5 Platform Independency

Here the platform contains some meanings such as operating systems, programming language, network middleware and communication media. As mentioned above, it is significant that platform supports multiple operating systems and languages for code reusability. Generally as for the code of a low level which controls hardware, C and C++ language are used, and a code of a high level, such as behavior and judgment of a robot will often be described by Java or script languages. In many cases, device drivers for robotic devices support a few operating systems and needs special communication media. Since a device and its device driver often depend on operating systems and communication media, the framework for modularizing it should not be dependent on them.

Currently OpenRTM-aist supports C++, Java, Python and C# languages on various UNIX, Mac OS X and Windows platforms. OpenRTM-aist's interoperability among these languages and OSs is realized by CORBA (Common Object Request Broker Architecture).

2.6 Social Requirement

The software platform has to be stable in the meaning of the quality of software code itself and the social continuity of the software. Needless to say that the code quality of the software platform is important. Additionally since many software components that are developed on the platform depend on the platform, continual existence of the platform is also important issue. The open source and copyleft strategy can be one of solution for it, and the open specification is the other solution.

OpenRTM-aist is an open source project, and it is released under LGPL license. We also opened its specification including component model and interface definition. Currently we released C++, Java and Python version of OpenRTM, and one private company released OpenRTM for C#, which is compatible with our OpenRTM-aist for C++, Java and Python. Multi-vendor environment gives the software platform diversity, optionality and continuousness. Additionally the specification itself has to be stable, so we have standardized the RT-Component specification in OMG (Object Management Group) [6].

3 Component Model

From the requirements mentioned above, we had studied about appropriate component model for RT-systems, and had defined the functionality in the component model. We call it RT-Component (RTC). Figure 1 shows the architecture block diagram of the RT-Component. The functionality of the RT-Component is as follows:

- Component metadata for dynamic component assembly.
- Component action and execution context for business logic execution.
- Data ports for data exchange between RTCs.

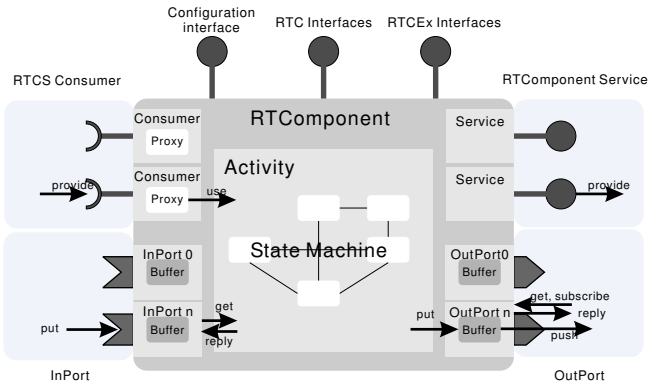
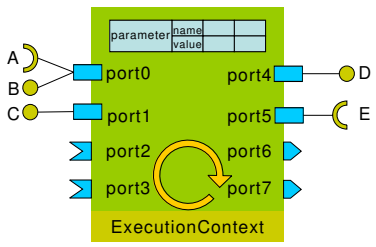


Fig. 1. The proposed architecture of RT-Component model. The RT-Component has a component body, common interface for metadata acquisition, component action, data ports, service ports and configuration interface.

– Service ports for service-oriented communication between RTCs.
– Configuration interface for runtime parameter setting.

3.1 Metadata Acquisition

The metadata acquisition capability, which realize querying and administering RTCs at runtime, is also known as "Introspection" (Figure 2). RTC has some interfaces to get metadata including profile, properties about ports. These capabilities can be used by other RTCs, tools or other application programs that support dynamic RTC composition. By using these metadata, application programs can obtain these metadata from RTC in runtime, and can make dynamic composition of RTCs in runtime. These metadata is also useful for components debugging tools and components composition tools. This functionality has two features, one is resource data model, the other is stereotype and interfaces. Resource data, which is a kind of data-only class, describes component profiles. Interfaces defines some methods to get or set profiles and properties.



Name	MyManipulator0
Type	Periodic execution type
port0	Provide: A, Required: B
port1	Provide: C
Port2	DataPort: InPort, velocity, float x6
Port3	DataPort: InPort, position, float x6
Port4	Provide: D
Port5	Required: E
Port6	DataPort: OutPort, status int x1
Port7	DataPort: OutPort, velocity, float x6
ExecutionContext	Period: 10ms
Parameter	gain0(float x6), flag(int x1), dev_file(string)

Fig. 2. The RTC provides introspection interfaces to obtain metadata of the RTC. Other RTC or application can utilize the metadata to make dynamic RTC composition.

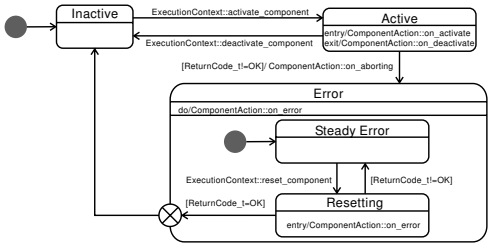


Fig. 3. The state machine of the ExecutionContext. Each callback named "on_xxx" is invoked on related transition events and actions.

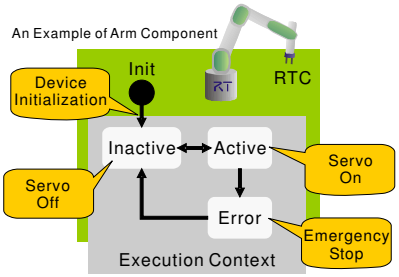


Fig. 4. The Component Action callbacks in which the component specific logic is implemented are invoked by the logical thread ExecutionContext

3.2 Component Action

The "Component Action" interface defines callbacks corresponding to the execution of the lifecycle operations of RTC. These callbacks would be invoked by the execution entity named "Execution Contexts" that is a logical thread object.

An RTC developer would implement Component Action's callback operations that would be invoked in each state of "Execution Context", in order to execute RT-component-specific logic. An RTC can participate in Execution Contexts, and an Execution Context can accept multiple RTC participants. As shown in Figure 3 and 4, an Execution Context performs a state transition between "Active" "Inactive" and "Error" state, and Component Action callbacks is invoked in appropriate timing in the state transition.

As mentioned above, the logic of an RTC and the logical thread is decoupled in the RTC model. This model is useful to implement tightly coupled RTCs in a single (real-time) thread. It is called the synchronous composite RT-Component.

3.3 Data Ports

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 (Figure 5). This input/output model is not 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. As shown in Figure 6, RT-Component adopted the publisher/subscriber model and defines it as InPort/OutPort.

OutPort supports some subscription types, "New", "Periodic" and "Flush." For example, the "New" subscription type means that an OutPort sends data to InPorts which subscribe it when a new data come from the Component Action.

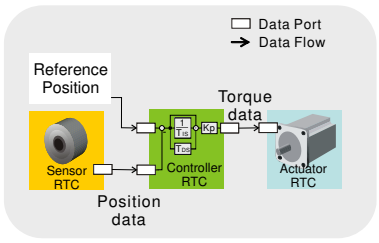


Fig. 5. A DataPort usage example. The DataPort is used for data-centric communication between components.

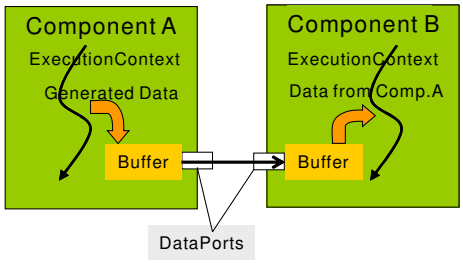


Fig. 6. The DataPort provides data-centric port for RTCs. The InPort receives data from the OutPort. The OutPort has some subscription types that control data pushing timing.

3.4 Service Ports

The software component should have enough interfaces to access to detailed functionality of the component from outside (Figure 7). The "Service Ports" provide endpoint to attach provided interfaces and required interfaces on it. Component developer can provide his/her own defined interface through the Service Port. The developer also can use provided interfaces by the other components through the Service Ports, as shown in Figure 8.

3.5 Configuration

The Configuration interface provides interfaces to administrate user defined RTC's parameters. As mentioned above, a component should not have the hard-coded configuration parameters which prevent reuse of the component.

The configuration consists of some configuration parameters as list of values with name, as shown in Figure 9. RTC is able to have some configurations as sets. This is called the Configuration Set. The Configuration Sets can be replaced in

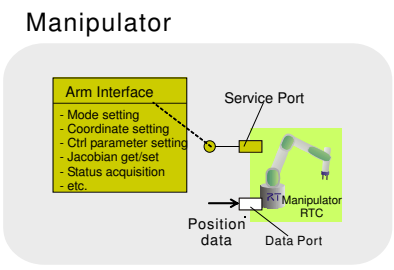


Fig. 7. A ServicePort usage example. The ServicePort is used for service-oriented communication between components.

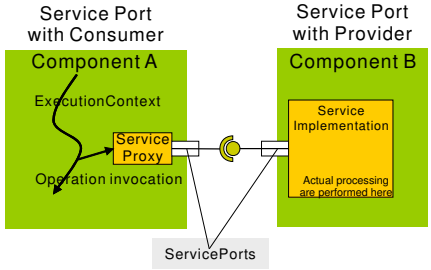


Fig. 8. The ServicePort provides service-oriented communication between RTCs. User defined service objects can be exported through the ServicePort.

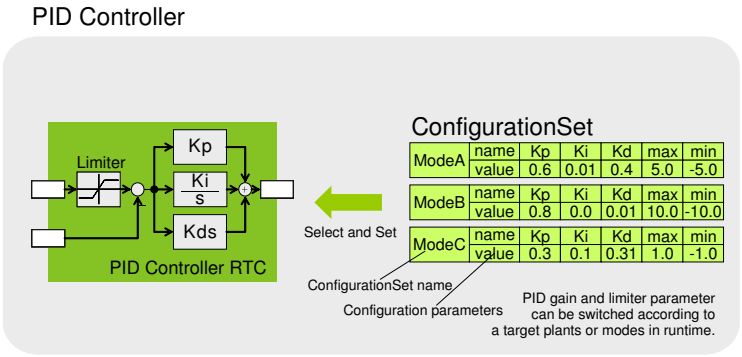


Fig. 9. The Configuration interface allows manipulation of configuration parameters in runtime. User can define some sets of the configurations.

runtime to adapt the RTC into the applications. For example, if an RTC realises PID controller with P-I-D gain as configuration parameters, the configuration set can be replaced or changed to adapt to the plant.

4 Implementation

4.1 OpenRTM-aist

According to the proposed RTC model, the "OpenRTM-aist (Open RT-Middleware distributed by AIST)" that is a component framework and middleware environment for RTCs have been implemented [7]. "OpenRTM-aist" consists of an RT-Component development frame work, a middleware including RTC manager and some tools. OpenRTM-aist is implemented on CORBA (Common Object Request Broker Architecture), because of its network transparency, OS/language independency and interoperability. Currently OpenRTM-aist supports C++, Java and Python languages on Windows, Linux and other UNIX-like operating systems. An RTC developer can choose appropriate language according to granularity, logic abstraction level and preference of language, and RTCs implemented on different languages are interoperable each other. OpenRTM-aist is also CORBA independent implementation, so it supports some CORBA implementation like omniORB, TAO, MICO and ORBexpress.

"OpenRTM-aist" provides a GUI tool to manage and administrate RTC on the network. The Figure 10 is the tool named "RtcLink."

The left side pane is "Name Service View" that show component list on the specific name server. The center pane is "System Editor" that is editor area to compose RTCs connection and to activate/deactivate RTCs. The right side pane is "Property View" to show the selected RTC's profile.

This GUI tool is implemented as an Eclipse plug-in. The Eclipse is a open-source project, and a lot of third party plug-ins is available. Since the Eclipse is one of the most widely used integrated development environment now, we have chosen the Eclipse as a platform of our tools.

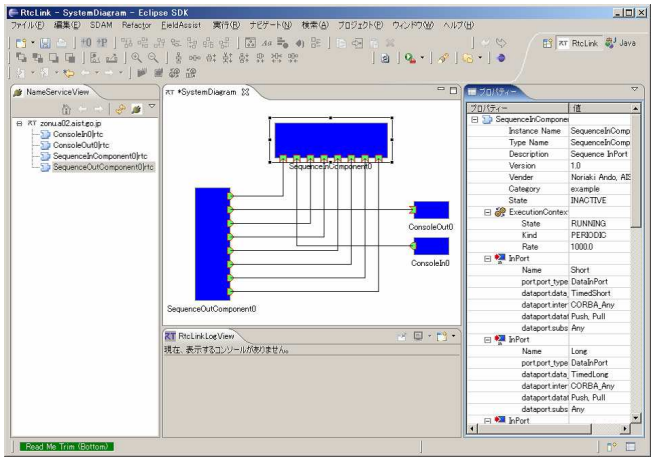


Fig. 10. RtcLink on Eclipse. RT-System online design tool running on Eclipse IDE.

4.2 RTM Based Systems

OpenRTM-aist already has more than 100 users, and some of national robotic projects in Japan adopts it as official platform. Here some of RTM based systems are shown.

Force Controled Manipulator. This is an example system, which consists of a force sensor RTC, a manipulator RTC, a joystick RTC and a dumper controller RTC, to show real-time capability of OpenRTM-aist.

As shown in Figure 12, these components are associated with same real-time thread, and each component's logic are executed synchronously in a 2 ms periodic task. Table 1 shows task execution time statistics in this experiment. The point is that these three devices components and one control component are not a

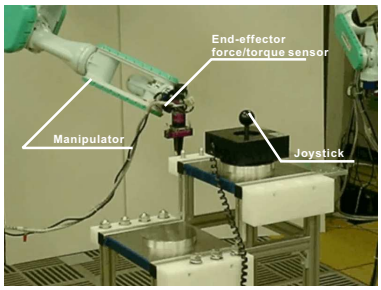


Fig. 11. Manipulator system equipment: End-effector force/torque sensor, manipulator, joystick

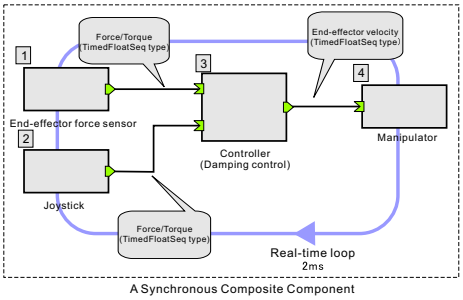


Fig. 12. A force controlled manipulator system using a synchronous composite component. Number in upper left of each block means execution order.

Table 1. Execution time of force controlled manipulator system

Task period time	2.00 ms
Maximum execution time	2.01 ms
Minimum execution time	1.99 ms
Mean execution time	2.00 ms
Standard deviation	4.41 μs

monolithic program but programs completely created separately. Furthermore, the point that these components were executed synchronizing in real-time is important.

Automatic LRF Calibration System. Sasaki et al. implemented their distributed LRF (laser range finder) automatic calibration algorithm on OpenRTM-aist [8](Figure 14).

This system consists of four type of RTC: LRF RTC, Tracker RTC, LRF Calibration RTC, Coordination Transform RTC. LRF sensors distributed on network are integrated by the network transparency capability of OpenRTM-aist.

Other RTCs. The following is an example of the components developed on OpenRTM-aist by OpenRTM-aist community.

- 3D recoginition and tracking RTCs by AIST and Applied Vision Co. Ltd.(Figure 15)
- Learning/inference RTCs baed on β-RNA by AdIn Research, Inc.
- LRF based human tracking RTCs by System Engineering Consultants Co., Ltd.
- Manipulator and bilateral tele-operation RTCs by AIST
- Input device RTCs (Game-pad, PHANToM, GUI, etc.) by AIST
- Dynamics simulator: OpenHRP3 by AIST(Figure 16)

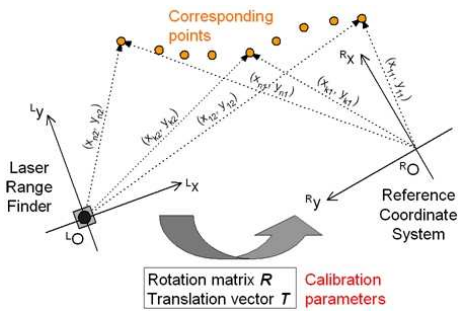


Fig. 13. The LRF automatic calibration algorithm. Relative.

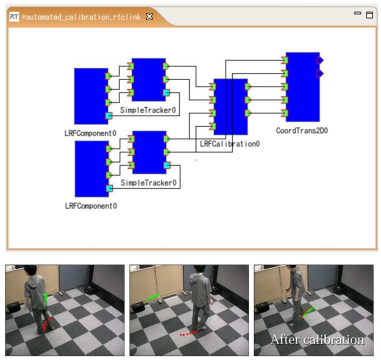


Fig. 14. LRF automatic calibration and tracking system based on OpenRTM-aist

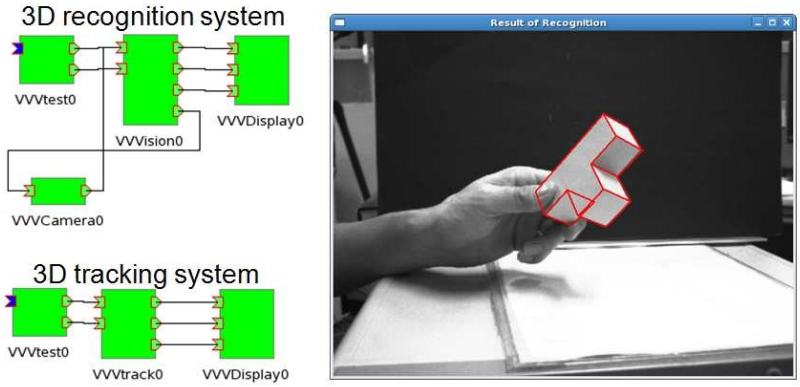


Fig. 15. 3D recognition and tracking RTCs. This system is based on VVV (Versatile Volumetric Vision) developed in AIST.

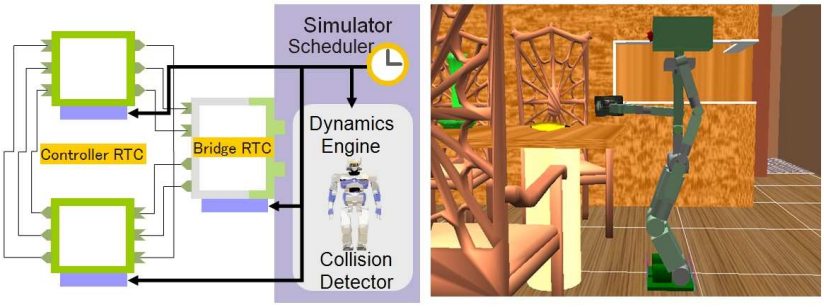


Fig. 16. The OpenHRP3 provides an environment for dynamics simulation for various types of robots including humanoid robots, manipulators and mobile robots. The controller RT-Component that is tested in the OpenHRP3 can be exported to the real robot without recompiling.

Currently a lot of RT-Components are being developed and circulation in a user community is also starting.

5 Conclusion

In this paper, we proposed component based robotic system integration scheme RT-Middleware and RT-Component. The functions required for the RT-Component which supports robot specific features were discussed and clarified. To realize component based robotic system development efficiently, RT-Component and its architecture was proposed. The "OpenRTM-aist", which includes RTC development framework, middleware and tools, have been implemented.

References

1. Japan Robot Association, Summary Report on Technology Strategy for Creating a Robot Society in the 21st Century (2001), http://www.jara.jp/e/dl/report0105.pdf
2. Ando, N., Suehiro, T., Kitagaki, K., Kotoku, T., Yoon, W.-K.: RT-Middleware: Distributed Component Middleware for RT (Robot Technology). In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005), pp. 3555–3560 (2005)
3. Gerkey, B.P., Vaughan, R.T., Stoy, K., Howard, A., Sukhatme, G.S., Mataric, M.J.: Most Valuable Player: A Robot Device Server for Distributed Control. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2001), pp. 1226–1231 (November 2001)
4. Makarenko, A., Brooks, A., Kaupp, T.: Orca: Components for Robotics. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2006), Workshop on Robotic Standardization (2006)
5. Volpe, R., Nesnas, I.A.D., Estlin, T., Mutz, D., Petras, R., Das, H.: The CLARAty Architecture for Robotic Autonomy. In: Proceedings of the 2001 IEEE Aerospace Conference, Big Sky Montana, March 10-17 (2001)
6. Object Management Group, Robotic Technology Component Specification Version 1.0, formal/2008-04-04 (2008)
7. OpenRTM-aist official web site, http://www.openrtm.org
8. Sasaki, T., Hashimoto, H.: Hierarchical Framework for Implementation of Intelligent Space. In: Proceedings of the 33rd Annual Conference of the IEEE Industrial Electronics Society (IECON 2007), vol. 11, pp. 28–33 (2007)