

Development of Component Based Middleware for Intelligent Robot Software

Makoto SEKIYA*

Norio NEKI*

Zenta MIYASHITA*

Nobuyuki ONO*

ABSTRACT

As robots become more intelligent, the software systems for realizing functions are becoming larger and more complex. This makes the use of component based middleware an effective means of design by dividing the overall system into function units (components) and then joining these components with the appropriate interfaces.

Intelligent robot software systems require composite components and finite-state machine components. The issues when developing middleware are the ability to synchronize these components to specific timings, and guaranteeing hard real time performance of minimum and predictable processing times including communication so that processing can finish in a certain sequence within a certain time.

This paper introduces Honda Robotic Technology Middleware, which is original component based middleware developed to address the above issues.

In consideration of use by outside organizations, the middleware specifications comply with the Robotic Technology Component specification formulated by the Object Management Group, which is an international standards organization. In addition, work is underway within the Object Management Group to standardize the original expanded specification.

1. Introduction

The ASIMO software system (hereafter, “system”) started from bipedal walking control⁽¹⁾ and has evolved to include abilities for working and human-robot interaction^{(2), (3)}. Recently, the system scale and complexity are expanding at a rapid pace, and the use of component based middleware is an effective means of design by dividing the overall system into function units (components) and then joining these components with the appropriate interfaces. This is because it allows understanding and verification of the system in smaller, simpler function units.

When developing component based middleware that can be applied to intelligent robot systems such as ASIMO, functions for designing and executing composite and finite-state machine (FSM) components are needed in order to componentize existing systems. The issues in this case are the ability to synchronize these components to specific timings, and guaranteeing the hard real time performance of minimum and predictable processing times including communication so that processing can finish in a certain sequence within a certain time.

This paper introduces Honda Robotic Technology Middleware (HRTM), which is component based middleware developed to address the above issues.

2. Component Based Middleware

Component based middleware (hereafter, “middleware”) is a development environment for realizing component based development, and has a hierarchical architecture consisting of the three elements of a design tool, component specifications, and a framework as shown in Fig. 1.

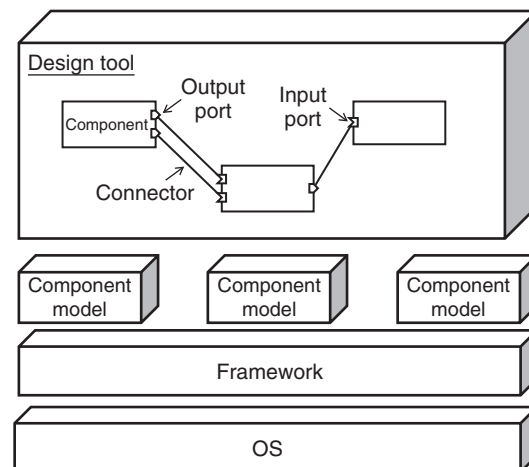


Fig. 1 Architecture of a component based middleware

* Fundamental Technology Research Center

The design tool is used to define components and design connection relationships using visual methods, and to generate the component source code. In some cases the design tool also has features such as debugging support functions.

The component specifications define the common interfaces used to invoke common function units (components) on the middleware. The defined elements typically include ports for sending and receiving data, and interfaces for getting and setting parameters.

The framework is a special program that executes the actual processing via the common interface defined in the component specifications, such as component startup and communication processing.

The framework should provide the functions required by the components in the Software In the Loop Simulation (SILS), Hardware In the Loop Simulation (HILS) and target environments in order to realize the reuse of components in each environment.

3. Requirements of Middleware for Intelligent Robots

The roles of middleware for intelligent robot systems are as follows.

1. Componentization of intelligent robot systems
2. Realization of component reuse between the SILS, HILS and target environments

This section describes the specific requirements for each element of component based middleware derived from the above roles. The distinct requirements in this development were to support composite components, FSM components and multiple communication protocols.

3.1. Design Tool Requirements

3.1.1. Ability to design systems using a standard modeling language

The use of an original modeling language for system design increases the time and effort required to master the tool operation methods and understand the design contents, and also means that existing design tools cannot be reused. Furthermore, a widely known standard modeling language should be used for the design tool in consideration of use by outside organizations.

3.2. Middleware Requirements related to Components

3.2.1. Ability to design hierarchically nested components

Large-scale and increasingly complex intelligent robot systems realize single functions using multiple hierarchically nested sub functions. Therefore, in order to componentize these systems, component design should be possible using hierarchically nested sub components.

3.2.2. Ability to design FSM components

Complex state transitions are used to implement the abilities for working and human-robot interaction of

intelligent robot systems, and enhancing the maintainability of these state transitions is an issue. To address this issue, it should be possible to design FSM components by modeling the internal state transitions.

3.2.3. Compliance with standard component specifications

Widely adopted standard component specifications should be used to enable easy creation of systems by reusing the many components developed by outside organizations.

3.3. Framework Requirements

3.3.1. Ability to execute components on various OS

The hardware systems of intelligent robots consist of multiple sensors with different roles. These sensors operate on various OS due to restrictions such as the provided drivers, and the programs used to process the data from the sensors at high speed are executed on the same OS as the corresponding sensors.

In order to componentize these types of systems, the framework should support multiple platforms to provide an environment for executing components on various OS.

3.3.2. Ability to reuse source code between different OS

The framework should enable the component source code to be compatible with different OS in order to realize component reuse between the SILS, HILS and target environments.

3.3.3. Ability to use multiple protocols for communication between components

Communication between programs within an intelligent robot system normally uses TCP/IP, UDP, shared memory, bus communication or other protocols in accordance with the application.

The reason for using different communication protocols according to the application is that using the fastest communication method that can be used in the program-running environment leads to shorter processing times.

In addition, components may also use different communication protocols between the SILS, HILS and target environments. In these cases, the framework should support multiple protocols for communication between components and it should be possible to select the protocol when compiling and executing components in order to realize both component reuse and high-speed processing.

3.3.4. Minimal framework processing time

If componentization notably increases the processing time, it may be a challenge to transfer from the SILS environment to the target environment in which the system runs on a real-time OS with limited processing capability. The framework should enable execution of processing including communication in a minimum and stable time in order to increase the ability to reuse components.

Table 1 Comparison of robotic middleware

Name	HRTM	OpenRTM-aist	ROS	OROCOS
Composite component	Yes	Up to sub-component	No	No
FSM component	Yes	No	Yes (Python)	No
Supported OS	Linux Windows Mac OSX	Linux Windows Mac OSX	Linux	Linux Windows Mac OSX
Supported protocol	CORBA shmem bus	CORBA	TCP/IP UDP	TCP/IP UDP
Supported RTOS	Proprietary RTOS	ART-Linux	No	No
System modeling language	UML	Original block diagram	No	No
Component model	RTC	RTC	Original	Original

4. Development of HRTM

When developing HRTM, the major implementations of robotic middleware were investigated. The investigated middleware are described briefly below.

OpenRTM-aist⁽⁴⁾ is robotic middleware developed by the National Institute of Advanced Industrial Science and Technology (AIST) in Japan. It complies with the Robotic Technology Component (RTC) specification⁽⁶⁾ issued by the international standards organization Object Management Group (OMG)⁽⁵⁾, supports a wide range of OS, and uses the international standard OMG CORBA⁽⁷⁾ as the communication protocol. OpenRTM-aist also has proven results supporting hard real time performance on ART-Linux⁽⁸⁾.

ROS⁽⁹⁾ is robotic middleware developed by Willow Garage Inc. (U.S.), and development is currently being continued by the Open Source Robotics Foundation. ROS is an open system and there is a wide range of products by numerous developers, but the officially supported OS is limited to Ubuntu⁽¹⁰⁾.

OROCOS⁽¹¹⁾ is a robotic software development project started under the sponsorship of the European Union. It supports the major OS and has proven operating results in real-time environments, but it does not have a component model, and has thus far been positioned as a software library for developing robotic control functions. Development is currently continuing with the support of PAL ROBOTICS SL (Spain), and steady advances are being made including the addition of original component models.

As a result of the investigation, it was found that there is currently no middleware that satisfies all the requirements described in Chapter 3, and that OMG RTC can be used as an international standard component specification. Therefore, development of the original middleware HRTM was started with the goal of realizing component specifications that satisfy the requirements while also complying with the international standard OMG RTC.

Table 1 compares the functions of HRTM and major middleware.

5. Achieved Functions

The features and functions achieved by HRTM are described below.

5.1. System Design Tool

HRTM enabled system design by a visual design tool that uses Unified Modeling Language (UML)⁽¹²⁾, which is an international standard modeling language.

The design tool enables defining of parameters such as component input/output port types, connection relationships and execution sequences using UML composite structure diagrams expanded in accordance with the RTC specification. The design tool also issues warnings when detectable errors such as input and output data type mismatch occur.

In addition, state transition diagrams are designed using UML statecharts, enabling the internal design of FSM components described hereafter. A statechart is a state transition diagram with a hierarchical structure, and enables creation of an easy to understand design when designing complex state transition rules for working and human-robot interaction.

Skeleton code generated automatically from the completed UML diagram is used when creating a new component. Common functions such as data communication based on the component specifications are processed by the framework, so component developers need only implement the internal processing to the skeleton code.

5.2. Component Specifications

5.2.1. Compliance with international standards and specifications

HRTM complies with the RTC specification that is defined as a component specification for robotic applications. However, the RTC specification is highly expandable and there are also examples of application to fields other than robotics⁽¹³⁾. RTC-compliant middleware has multiple implementations such as the previously described OpenRTM-aist^{(14), (15)}, and is used by research centers and universities both domestically and overseas.

HRTM and OpenRTM-aist components can be interconnected based on the RTC specification, so the ASIMO software system and existing components can be easily connected when performing joint research with an outside research organization.

5.2.2. Composite components

As described above, when componentizing large-scale systems, compositing is a key function for enhancing system visibility and maintainability. Therefore, HRTM implements a multi-tiered composite component function that expands the RTC specification.

The composite component based on the RTC specification consists of the following elements (Fig. 2).

1. Super component
2. Sub component

3. Public port
4. Execution thread

A super component includes one or more internal sub components. A public port is a super component port used by a sub component to communicate with external components. An execution thread executes one or more assigned sub components.

The RTC specification does not clearly state the means for specifying the execution sequence of the sub components contained in a composite component. Therefore, a sequential execution thread was developed as an original HRTM function that can specify the sub component execution sequence to the execution thread using parameters.

5.2.3. FSM components

The abilities for working and human-robot interaction of intelligent robot systems are implemented using complex state transitions, and enhancing maintainability has been an issue. HRTM realizes visual design of FSM models using UML statecharts, making it possible to express complex state transitions in a concise manner.

Figure 3 shows an example of a simple FSM model. The black circle in the figure represents the initial state, the rectangles with rounded corners represent states, and the arrows represent transitions. The labels indicate the names of the corresponding elements. The FSM model shown in Fig. 3 first automatically transitions from the initial state to State A. Thereafter, the state transitions according to

the transition rules. For example, when Event A, Event B and then Event C are input from the initial state, the state transitions to State B, State C and then State A.

Figure 4 shows a model with a transition rule added to the model in Fig. 3. Here, State A and State B both transition to State C when Event B occurs. This type of shared transition rule between states is often used for reset and other processing even in actual models. However, as the number of states and transition rules increase, it becomes easier to forget to assign transition rules.

Shared transition rules for sub states can be integrated by designing the states in a UML statechart in a hierarchical manner. Figure 5 shows the same model as that in Fig. 4 converted to a hierarchically nested model. In this figure, State A and State B are designed as a Super State sub component. In the UML statechart specification, sub components follow the state transition rules of the parent Super State, so the transitions to State C due to Event B are integrated into a single Super State transition rule. Comparison of Fig. 4 and Fig. 5 shows that composite design enables concise description using a model.

Implementation of the designed FSM model is included in the FSM component skeleton code, so there is no need for developers to implement the FSM. In addition, errors in the FSM model are detected on the design tool, so illegal FSM models cannot be output as skeleton code.

These functions facilitate maintenance of complex state transitions and enable elimination of bugs and other issues during implementation.

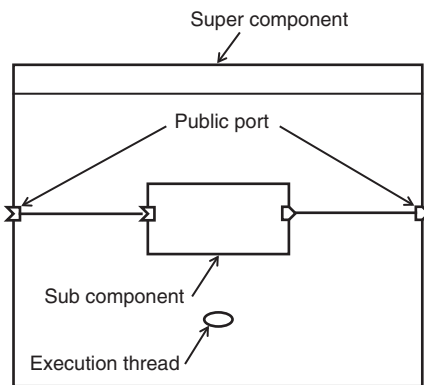


Fig. 2 Overview of a composite component

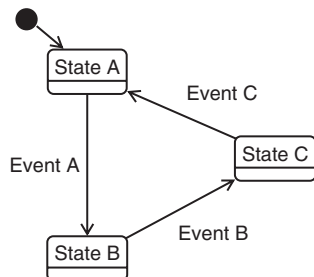


Fig. 3 UML statechart of simple finite state machine

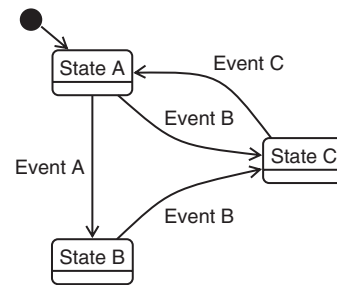


Fig. 4 UML statechart of finite-state machine in which different states have share transition rule

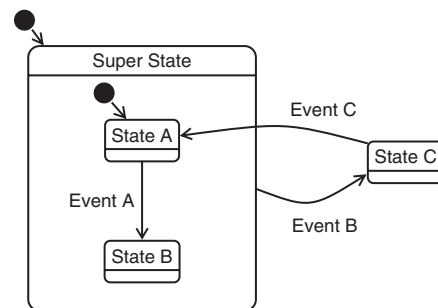


Fig. 5 UML statechart of the finite state machine in Fig. 4 with hierarchically nested states

5.3. Framework

5.3.1. Support of multiple platforms

Execution of components on various OS is a requirement of the framework, so an OS abstraction layer that hides multiple OS including real-time OS was created between the OS and the framework (Fig. 6).

Programs created using this abstraction layer can be compiled and executed on multiple OS without changing the source code. This made it possible to realize a highly portable framework that supports multiple platforms.

5.3.2. Support of multiple communication protocols

A function was implemented in HRTM that stores the parameters for specifying the communication protocol between components in the database of each component and selects the appropriate protocol when executing the components.

Communication processing using the selected protocol is executed by the framework, and component implementation does not depend on the communication protocol. Therefore, the unmodified component source code can be reused and executed even when the communication protocols that can be used differ between the SILS, HILS and target environments.

5.3.3. Hard real time performance

The control system of a bipedal walking robot needs to guarantee hard real time performance that completes calculation of command values in a certain sequence within a certain time while synchronizing with specific timings such as hardware interrupt signals. In order to guarantee the hard real time performance of components, efforts were made with HRTM to develop a synchronized sequential execution thread that executes components based on priorities and to reduce the framework execution processing time.

In addition to the sequential execution thread functions, the synchronized sequential execution thread can specify the synchronization source, such as hardware interrupts or an internal clock, using a parameter. This enables specifying of the execution sequence with the minimum component processing time within a specific cycle, thereby allowing guarantee of hard real-time performance.

Reduction of the framework processing time was the

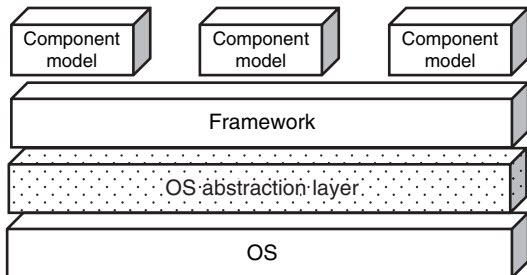


Fig. 6 OS abstraction layer of HRTM

Table 2 Processing time to send/receive 1000 bytes data [us]

	Data conversion	Data copy	Data receive	Data send	Total
Send	100	13	-	75	188
Receive	100	13	10	-	123

Table 3 Required processes and types of communication

	Send or receive	Data conversion	Data copy	Data receive	Data send
Network	Send	Yes	Yes	-	Yes
	Receive	Yes	Yes	Yes	-
Inter-process	Send	No	Yes	-	Yes
	Receive	No	Yes	Yes	-
Inner-process	Send	No	Yes (address)	-	Yes
	Receive	No	Yes (address)	Yes	-

most challenging technical issue in HRTM development, so this is described in detail as follows. The framework processing time when executing components was analyzed in the initial development stage. The results showed that the communication time accounted for a large percentage of the processing time, so the main issue was to reduce the communication time. Table 2 shows the results of measuring the processing time required to send and receive 1000 bytes of data via TCP/IP before measures to reduce the communication time.

The measurement results show that the receive processing time is 123 μ s and the send processing time is 188 μ s. These values account for a large percentage of the average processing time of programs running in hard real time in the ASIMO system. Therefore, the componentization of existing programs that involve data input and output leads to a notable increase in the processing time.

The main factors increasing the processing time are the processing time of 100 μ s for converting the data to the network format and back again, which is included in the communication time based on the RTC specification, and the data send time of 75 μ s required to send data via TCP/IP. The data send time can be shortened by replacing the communication protocol with a protocol faster than TCP/IP, but eliminating the data conversion processing would result in non-compliance with the RTC specification.

Therefore, HRTM expanded the RTC specification to enable selection of three types of connection formats (network, inter-process, intra-process) in addition to the communication protocol. When connecting with RTC specification components, this expansion enabled use of RTC-compliant communication methods for inter-process communication that omits the data conversion processing and for intra-process communication that replaces the data copy processing with address reference processing. This realized high-speed communication between HRTM components, which shortened the communication time (Table 3).

Table 4 Processing time of intra-process communication [us]

	Data conversion	Data copy	Data receive	Data send	Total
Send	-	7	-	-	7
Receive	-	13	3	-	16

When using the fastest available intra-process communication between components, the above measures reduced the communication processing time from 123 μ s to 16 μ s when receiving data and from 188 μ s to 7 μ s when sending data (Table 4). Intra-process communication is performed by referencing the address in the process memory space, so the communication time is definite, making it possible to guarantee hard real time performance.

6. Application to ASIMO

The results of application to the path tracking walk module of the ASIMO system are described here as verification of practical use of HRTM functions. The path tracking walk module is a program that performs walking along a tracking line by repeatedly calculating the next footstep landing position (hereafter, “step”) along a given path relative to the current position as shown in Fig. 7.

6.1. Overview of Path Tracking Walk Module

Figure 8 shows an overview of the path tracking walk module. The path tracking walk module repeatedly receives a timer interrupt, checks the walk state, and calculates the next step at a 5 ms cycle, so it is a suitable subject for verifying hard real time performance including communication performance.

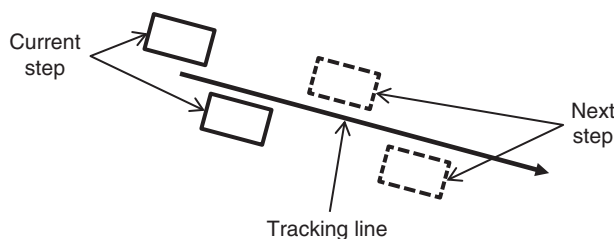


Fig. 7 Overview of path tracking walk

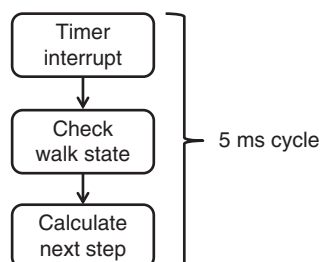


Fig. 8 Processing steps of path tracking walk

6.2. Application Procedure

The procedure for applying middleware to the path tracking walk module was as follows.

1. Component design
2. Confirmation of execution in SILS
3. Confirmation of execution in the target environment

6.3. Component Design

Figure 9 shows the architecture of the path tracking walk module before componentization. The foot step planner processing for path tracking receives an interrupt signal at a 5 ms cycle, executes the series of steps shown in Fig. 8, and then transfers the calculation results to the leg controller block via the shared memory.

The leg controller is also executed at the same 5 ms cycle, and realizes path tracking walk by calculating the track for placing the footstep in the given step position and then moving the leg accordingly.

The approach for componentization was to model the state transitions in the footstep planner as an FSM component. Therefore, the footstep planner processing was stratified and divided internally into an FSM and a robot status notifier (blue line part in Fig. 10). The divided FSM and status notifier are both executed at a 5 ms cycle, but execution needs to be performed in order of status notification and then state transition, so the sub component execution sequence was set

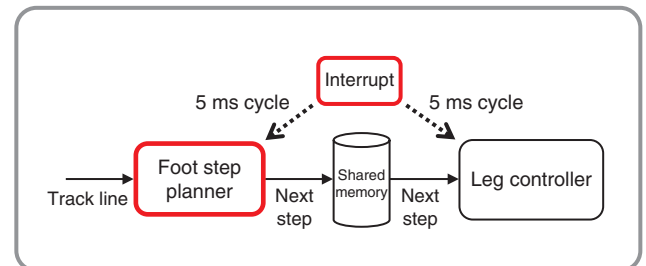


Fig. 9 Original architecture of path tracking walk (red line part)

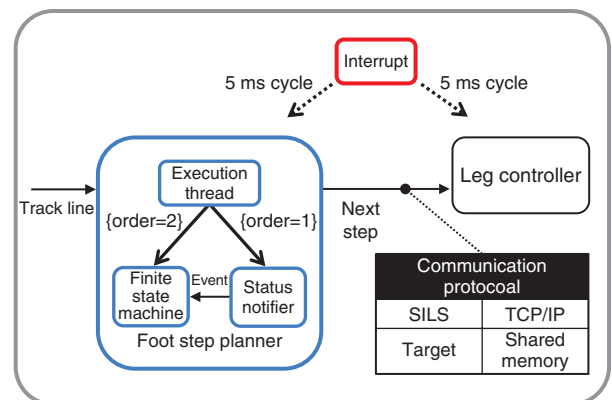


Fig. 10 Componentization of path tracking walk (blue line part)

using parameters. Figure 11 shows the model of the footstep planner actually created by the design tool, and Fig. 12 shows the model of the FSM component.

6.4. Confirmation of Execution in SILS

The component skeleton was automatically generated from the designed model, and the functions were ported from the program before componentization to the SILS environment.

The completed component was used to compare execution in the SILS environment before and after componentization, and it was confirmed that the results matched completely.

6.5. Confirmation of Execution in the Target Environment

When confirming execution in ASIMO, the component source code used for SILS could be executed without modification simply by recompiling it in the target environment.

All the parameters that change according to the environment were stored in a file or the component internal database, making it possible to check the component reusability.

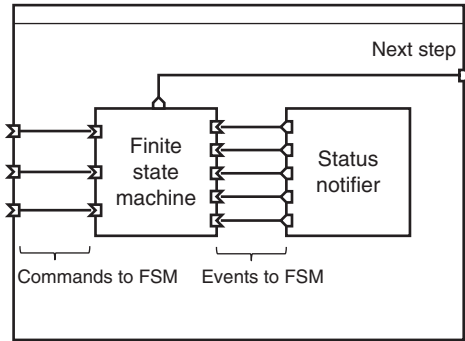


Fig. 11 UML diagram of foot step planner

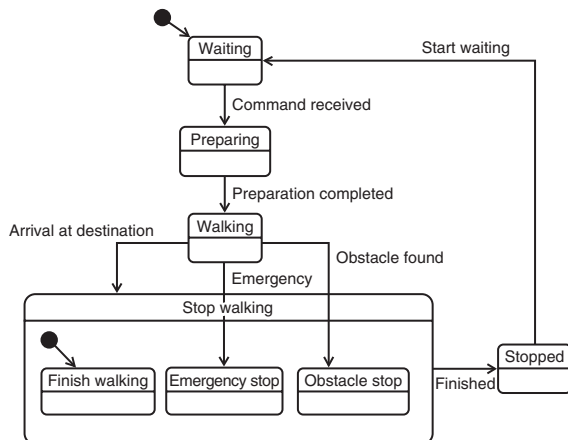


Fig. 12 UML statechart diagram of FSM component

Table 5 Comparison of processing time [us]

	Interruption processing	Input port processing	Walk status checking	Next position calculation	Total
Original	42	-		309	351
Component	22	80	50	280	432

6.5.1. Comparison of execution results

The results of walking tests using ASIMO confirmed that the componentized path tracking walk module executed at a 5 ms cycle and that the FSM component performs the designed state transitions.

In addition, comparison of the leg trajectory before and after componentization confirmed that both matched completely.

6.5.2. Processing time measurement results

The path tracking walk processing time was measured before and after componentization to verify whether hard real time performance is satisfied (Table 5). When componentizing path tracking walk, the implementation of the timer interrupt process was changed from the original program and the status notification and next position calculation were separated. This made it difficult to directly compare the processing time of each component, but the overall processing time after componentization increased by 81 μ s (23%).

6.6. Observations

Componentization of ASIMO path tracking walk module made it possible to confirm that the source code of components generated automatically from the designed model is available in both the SILS and target environments without modification.

In addition, componentization increased the processing time by 23%. This is within the allowable range for the ASIMO operation in this study, but methods of reducing the increase in the processing time should be examined in consideration of use under stricter conditions.

One item that can be reduced by modifying the component design is the number of input ports. There are a total of eight input ports to the FSM component of the footstep planner as shown in Fig. 11, and the receive data size ranges from 10 to 20 bytes. Each input port performs receive processing in sequence, so the total processing time is the product of the number of ports and the receive time. The measurement results show that the data receive time for intra-process communication is 9.4 μ s when receiving 20 bytes, so the estimated processing time when receiving data with all eight ports is $9.4 \times 8 = 75.2 \mu$ s. Compared to this value, the input port processing time of 80 μ s obtained from Table 5 is thought to be a valid value.

In the component design for this study, a port was created for each command and status notification type as shown in Fig. 11. However, design can also combine commands and status notifications into a single port. In this

case, Table 4 shows that the time required to receive 1000 bytes is 16 μ s, so the increase in the processing time could be kept to $16 \times 2 = 32 \mu$ s (10%) or less.

7. Standardization Activities

Lastly, the standardization activities related to HRTM are described below. HRTM also implements the following extended specifications with respect to the RTC specification.

- A data porting specification that can handle multiple protocols
- An interface specification for notifying and obtaining the current FSM component status

Efforts to standardize these extended specifications are being promoted by the Infrastructure Working Group under the Robotics Domain Task Force, which is the committee in charge of formulating OMG specifications in the robotics sector, with the goal of release as publicly available FSM4RTC specifications in 2015.

Standardization of these extended specifications will enable the reuse of compatible high-speed inter-component communication and FSM components between RTC-compliant middleware.

8. Conclusion

The component based middleware HRTM was developed that includes original extensions and complies with the international standard RTC specification.

In addition, ASIMO path tracking walk module was modeled and existing programs were ported to the generated skeleton code to verify the performance of the HRTM framework. The verification results proved that the components can be reused between the SILS environment and the ASIMO target environment, and that hard real time performance is also satisfied.

Componentization can be realized by adding part of the source code omitting the communication and other processing from existing programs to the skeleton code generated from the system model defined by the design tool. This can be expected to reduce the amount of source code handled by developers and to enhance both maintainability and quality.

In the future, efforts will be made to expand the effects of component based development by using HRTM in ASIMO development, to standardize the extended specification as FSM4RTC, and to expand the range of HRTM application to external research organizations and fields other than robotics.

References

- (1) Hirose, M., Takenaka, T.: Development of the Humanoid Robot ASIMO, Honda R&D Technical Review, Vol. 13, No. 1, p. 1-6
- (2) Shigemi, S., Kawaguchi, Y., Yoshiike, T., Kawabe, K., Ogawa, N.: Development of New ASIMO, Honda R&D Technical Review, Vol. 18, No. 1, p. 38-44
- (3) Shigemi, S., Kawabe, K., Nakamura, T.: Development of New ASIMO - Realization of Autonomous Machine -, Honda R&D Technical Review, Vol. 24, No. 1, p. 37-45
- (4) Ando, N., Sehiro, T., Kotoku, T.: A Software Platform for Component Based RT-System Development: OpenRTM-aist, International Conference on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS (SIMPAN 2008), p. 87-98 (2008)
- (5) <http://www.omg.org/>
- (6) <http://www.omg.org/spec/RTC>
- (7) <http://www.omg.org/spec/CORBA>
- (8) <http://www.ros.org/>
- (9) <http://www.dh.aist.go.jp/jp/research/assist/ART-Linux/>
- (10) <http://www.ros.org/repos/rep-0003.html>
- (11) Bruyninx, H.: Open Robot Control Software: the OROCOS project, Proceedings of the 2001 IEEE International Conference on Robotics and Automation, Vol. 3, p. 2523-2528 (2001)
- (12) <http://www.omg.org/spec/UML>
- (13) <http://daqmw.kek.jp/>
- (14) <http://www.sec.co.jp/robot/openrtmnet/introduction.html>
- (15) <http://www.gostai.com/products/rtc/>

■ Author ■



Makoto SEKIYA



Norio NEKI



Zenta MIYASHITA



Nobuyuki ONO