2002년도 대한전기학회 하계학술대회 논문집 2002.7.10-12

차세대 퍼스널 로봇 소프트웨어 프레임워크에 관한 연구

김홍렬*, 김대원*, 김홍석**, 이호길*
명지대학교 정보제어공학과*, 한국생산기술연구원**

Toward the Personal Robot Software Framework

Hong Ryeol Kim*, Dae Won Kim*, Hong Seok Kim**, Ho Gil Lee**
Department of Information Control Engineering, MyongJi University*, KITECH**

Abstract - In this paper, a software framework is proposed for the personal robot located on home network. The proposed software framework is divided into four layers-a transparency layer, a behavior layer, a distributed task layer, and a mission scenario layer. The transparency layer consists of a virtual machine for platform transparency, and a communication broker for communication transparency among behavior modules. The communication architecture includes both server/client communication and publisher/subscriber communication. A mission scenario is assumed to be a composition of sequentially planned distributed tasks. In addition to the software framework, a new concept, personal robot design center platform is proposed in this paper with its implementation mechanisms. The personal robot design center is defined as a developing and a managing environment for high-level behavior modules, distributed tasks, and mission scenarios.

1. Introduction

The market of personal robots, being enlarged rapidly, is led by many companies of small or medium size, while the market of industrial robots is led by a few companies of huge size. The trend can give a chance to the market to be enlarged more rapidly along with the emergence of functional hardware module manufacturers. The emergence of functional hardware module manufacturers makes customers assemble their own personal robots with their favors and manufacturers reduce their manufacturing costs with mass production.

Hardware modularization can be achieved with a standard software framework that can guarantee interoperability among the modules and capability of processing distributed tasks. Also the software frameworks should preserve existing know-hows of manufactures in addition to the above requirements.

The personal robot has the other difficulties concerning its operation compared to them of the industrial robot as follows:

1) Usually customer or end user can be a manager of the personal robot, while specified managers are needed for the industrial robot.
2) It can be expected that various kind of requirements for the personal robot are needed according to personal usages, and the life time of the requirements is short.
3) Even if the modularization mentioned previously is possible, it shall be difficult to make programs for the distributed modules.

In this paper, a personal robot software framework is proposed to resolve above problems. The proposed framework is divided into four layers-a transparency layer, a behavior layer, a distributed task layer, and a mission scenario layer.

The transparency layer is composed of a virtual machine for platform transparency, and a communication broker for communication transparency among behavior modules. The communication architecture includes both server/client communication and publisher/subscriber communication.

With the transparency layer, the behavior modules, practical implementation units of functions, can have interoperability among them and real-time capability. The behavior modules with concurrency are software components for the easiness of maintenance and replacement.

The distributed tasks, logical groups of distributed behavior modules can perform specified tasks on the distributed environment.

The mission scenario, a sequential logic of the distributed tasks is a minimum service function in the view point of customers.

In addition to the software framework, a new concept of personal robot design center platform is proposed in this paper with its implementation mechanisms. The robot design center is defined as an environment for programming, installing, managing and maintaining behavior modules, distributed tasks, and mission scenarios. The robot design center provides GUI(Graphic User Interface) environment and function of file trasfer between database repository embedded and personal robots through internet remotely or home network locally.

In the following chapters, related works previously done are described in chapter 2. A new personal robot software framework and a robot design cente platform with their mechanisms are proposed in chapter 3. Finally conclusions are described in chapter 4.

2. Technical Backgrounds

2.1 Robot Control Framework

Prior to the emergence of component concept in the modern software engineering, the behavior-based robot framework[1] was proposed as a robot control framework.

A behavior is a minimum functional unit for achievement of a goal. Each behavior has layered architecture according to its computation complexity and frequency. With more complex computation and lower frequency, a behavior is placed on higher layer. With the behavior-based robot framework, highly-layered behaviors can override and limit outputs of lowly-layered behaviors. Consequently, the structure of the framework is known to be adequate for autonomous robots to meet real-time constraints.

A robot framework for robotic workcells[2] employs components concepts of the modern software engineering, design by composition and reusability of components. The framework is composed of machine block components, hardware interface components, and workcell manager components, the structure of which is similar to well-known 3-tier structure.

2.2. Virtual Machine

Recently the trend of compiler is that the compiler is designed to be divided into two parts. The first part makes intermediate codes from source codes, and the second part

makes object codes from the intermediate codes. The division makes it possible to have the first part transparent to platforms and portable. P-code of Pascal, byte code of Python, and another byte code of Java are well-known intermediate codes, and related virtual machines serve as the second parts. JVM(Java Virtual Machine)[13] is the stack-based virtual machine of Java byte codes optimized for code minimization to achieve network mobility, thread handling, and exception handling.

2.3. Communication Transparency

To reduce complexities and to increase flexibilities in various distributed environments, communication transparency should be guaranteed.

The most well-known component systems with communication transparency are COM/DCOM from Microsoft and CORBA from OMG(Object Management Group). Particularly the CORBA provides both early binding(compile-time binding), and late binding(run-time binding), with both of server/client and message communication. The transparency is guaranteed by ORB(Object Request Broker) embedded. A study on the real-time ORB[4] was performed recently.

Neuron C[5] of Lonworks systems is another example of transparency provider. In case of Neuron C, ANSI C-extended definers provide ways for users to define network variables bond lately. Only the limitation of Neuron C is that the transparency is for network variables among hardware platforms separated physically.

2.4 Real-Time Scheduling

Real-time scheduling is divided into real time operating system scheduling and real-time communication scheduling.

1) Real-time operating system

The real-ime operating system is the operating system with a real-ime task scheduler (or a real-time thread scheduler) and a shared data manager.

The real-time scheduler maintains real-time scheduling of all tasks in multi-tasks environment (or real time of all threads in multi-threads environment). RMS (Rate Monotonic Scheduling)[6] with static scheduling method and EDS(Early Deadline Scheduling)[7] with dynamic scheduling method are the most well-known scheduling methods currently.

The shared data manager manages priorities to access shared data. Priority inheritance protocol[8] is one of the most outstanding mechanism that can prohibit priority inversions.

2) Real-time communication

Communications among software modules can be divided into cyclic communication and acyclic communication.

The most well-known cyclic communication method is message communication. The message communication can be divided into the point-to-point message communication and the publisher/subscriber communication. Usually the message communication is achieved through queues and so can guarantee real-time features. While the message service provider is not needed in case of the point-to-point communication, it should be needed to schedule publishing and subscribing in case of the publisher/subscriber communication.

The JMS(Java Message Service)[9] is a framework that can provide both the point-to-point communication and the publisher/subscriber communication. There is another study that suggested real-time mechanisms for the publisher/subscriber communication using the UDP(User Datagram Protocol)[10].

One of the most well-known acyclic communication method is the server/client communication. Usually the server/client communication can not be used for real-time communication because of event synchronousness between a server and a client. However the server/client method can compensate event handling. The RPC(Remote Procedure Call)[11] and the RMI(Remote Method Invocation)[12] are server/client communication methods with distributed environments.

2.5 Run-Time Maintenance

JAVA provides dynamic loader that can load software module at run time[3]. The most outstanding features of the class loader is that it enables software components loaded or unloaded at run time, and that it can be handled by programmers. Similarly, Linux provides mechanisms to access dynamic linker that can enables dynamic linking libraries to be loaded or unloaded at run time.

3. Personal Robot Sodtware Framework

3.1 Personal Robot Software Framework

A personal robot software framework extended from the existing behavior-based framework is proposed here. The whole layered architecture of the framework is shown in Fig. 1.

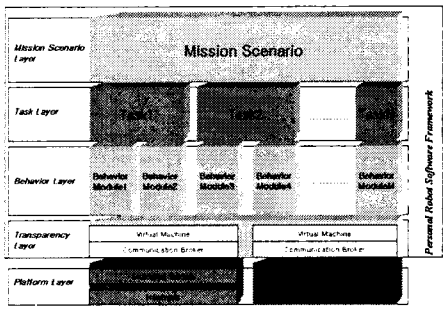

Fig. 1. Layered Architecture of the Personal Robot Software Framework

3.1.1 Robot Platform Layer

The robot platform layer includes know-hows and uniqueness of hardware module that module makers can embed independently.

3.1.2 Transparency Layer

The transparency layer is the media between above layers and the robot platform layer. This layer is composed of a virtual machine and a communication broker, and lets the above layers be independent on the robot platform layer.

3.1.2.1 Virtual Machine

The proposed virtual machine is a light stack-based byte code interpreter and the compiler that has ANSI C-extended definers.

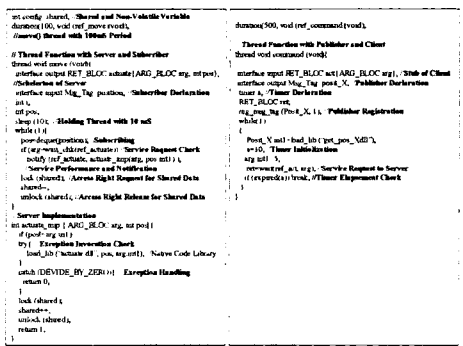

Fig. 2. Source Codes for the Proposed Virtual Machine

The followings are descriptions on the ANSI C-extended features of the virtual machine compiler and the example of the source codes are shown in Fig. 2.

1) Software timer
Methods for handling software timers like timer declaration, timer initialization, and timer expiration check are provided.

2) Exception handling
Methods for catching exceptions and handling the ones are provided.

3) Expressive reference
Since exsisting pointers of ANSI C are very complex to adopt for a virtual machine and they make the virtual machine dependent on a hardware platform, no pointer-related feature is provided for the virtual machine proposed. Anyway there are reference methods to data or functions expressively.

4) Thread control
The compiler of the virtual machine proposed provide a definer to define thread functions expressively. And other functions related to thread controls such as thread period set, thread hold for time synchronousness, event release and wait, event release check, and event notification for event synchronousness. The functions for event synchronousness are used mainly for server/client communication. Shared data and shared functions can be defined outside of any thread function, and the virtual machine handles the access of the shared sections.

5) Communication control
Expressive definers of lately-binding variables are provided for transparency. Server/client communication control can be implemented using the previous thread control functions. A message tag definer, a publishing registration function, and a message queue subscribing function are also provided for publisher/subscriber communication.

6) Non-volatile variable
The robot platform is a kind of embedded system, so it should provide a way to store variables in non-volatile memory area, like EEPROM or flash memory using a definer.

7) Native code interface
Method for handling robot platform directly, not through the virtual machine is provided. Software modules that handle robot platform with native codes should be able to be implemented as dynamic linking libraries which are able to be loaded in threads.

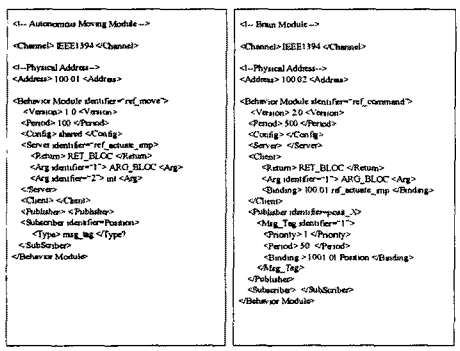

Fig. 3. Example of Robot Platform Template File

8) Generation of robot platform template file
When a user compiles a source codes, a robot platform template file is generated automatically by the compiler. Here, the platform template file is a XML(eXtensible Markup Language) format including such information as platform identification, configurable properties, and interface binding information. The template files of the example codes in Fig. 2. are shown Fig. 3. The template file is used by personal robot design center platform, communication broker, and dynamic thread loader.

3.1.2.2 Communication Broker

Location transparencies of server/client communication and publisher/subscriber communication are provided by the communication broker proposed. The binding information that enables communication transparency is included in template files.

Server/client communication is used for non-real time and acyclic communication. Event synchronousness functions for threads are used for the communication implementation. A argument type and a return type are provided for universal use as shown in Fig. 2. The implementation mechanisms are the same as follows:

Step1 :To request service to a server, a client delivers the service function reference and an argument type instance to the below virtual machine. The reference defined within the client has same function as generally known stub.

Step2 :The virtual machine pushes the client into wait queue, and delivers the service function reference and the argument to the below communication broker.

Step3 :The local broker investigates the binding information of the service function through robot platform template file. If the service is local platform service, it shall set event release for the practical service. If the service is remote platform service, it shall deliver the request through inter-broker protocol referred physical address bond.

Step4 :For the delivered service request, the broker on the remote platform sets event release for the above server.

Step5 :The server calls the existing service function to serve the service request and notify the execution of the service by delivering a return type instance and the service function reference to the below virtual machine.

Step6 :The virtual machine delivers the return type instance and the service function reference to the below broker.

Step7 :The broker investigates the reference. If with local platform request, it shall resume the client and deliver the result. If the request is remote platform request, it shall deliver the request through inter-broker protocol.

Step8 :For the delivered return, The originating broker resumes the client and delivers the return.

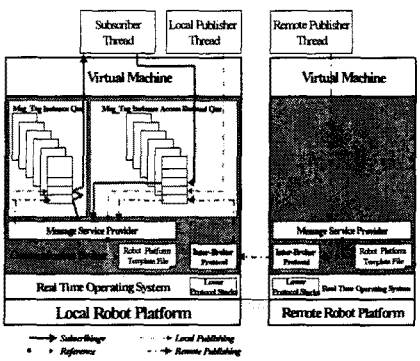

Fig. 4. Publisher/Subscriber Communication Mechanism

Publisher/Subscriber communication is used for real-time and cyclic communication, and the real time feature is guaranteed by a communication broker. The broker manages message instance queues for message tags.

A message tag is assigned to a subscriber, and several publishers can publish messages of the message tag instance. There is a request queue to manage access requests from a subscriber and publishers for a message instance queue, and access acceptances are allowed by priority scheduling.

A publisher calls a publisher registration function with priority as an argument before the first publishing. A publishing period is defined according to the argument by

communication broker. The priorities of all publishers are always lower than it of the subscriber.

A broker configures access periods according to the priorities and current binding information extracted from robot template file. Periods configuration is invoked when a publisher calls the registration function or the binding information is changed. Publisher/subscriber communication mechanism with transparency is shown in Fig. 4.

3.1.3 Behavior Layer

The behavior layer is composed of distributed behavior modules. A behavior module is defined as a minimum functional component with interfaces and functions provided by a robot module maker or personal robot design center. One and more behavior modules are delivered on a robot platform and communicates with each others to exchange task information. A behavior module is implemented into a thread, and consequently can run concurrently with others.

For an example, in case of autonomous vehicular robot, "Driving" and "Detecting Obstacles" are thought to be concurrent functions for achieving a goal, "Moving", "Driving" and "Detecting Obstacle" are behavior modules for a task, "Moving".

Each behavior module serves a specified function concurrently with each others. And the real-time processing of the functions is guaranteed by the virtual machine. Proposed real-time scheduling method of the virtual machine in this paper is the RMS(Rate Monotonic Scheduling). As described previously, each thread can be assigned a period. The following formula is for the real-time schedulability, *Wn* of the RMS:

$$Wn = n(2(1/n) - 1) \text{ where, } n : \text{Number of Task} \quad (1)$$

As described previoulsy, message tag instances assigned to a behavior module can accept several publishing messages. Consequently, behavior modules can be implemented to check published information and plan reactions automatically. The above means that the proposed framework is functionally similar to the previous behavior-based framework[1].

3.1.4 Distributed Task layer

A task means the minimum unit of work process, and practically a logical group of connected behavior modules for achieving the function of the task. Practically, a request for a task performance is implemented as request for a service to the server located on the highest logical layer on the group of connected behavior modules. Since task should be structured guaranteeing its transaction, there should not be external interrupt that can affect on the task performance. All things that can affect on the task performance should be embedded in the task as behavior modules.

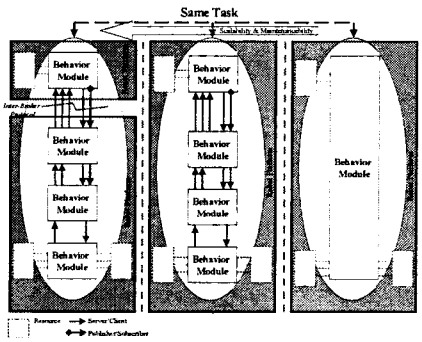

Fig. 5. Scalability of Distributed Task

Because the locations of behavior modules are transparent by the communication broker mentioned above, a task can be distributed. With the above example "Driving" and "Obstacle Detecting" can be on a same robot platform, or on different robot platforms separately. The transparency consequently guarantees the scalability of robot platforms. The concept of scalability with distributed tasks is shown in Fig. 5. In Fig. 5. behavior modules can be scalable to achieve a goal. Anyway, the smallest-sized behaviour modules make it flexible to replace or maintain them seamlessly and comfortably.

3.1.5 Mission Scenario Layer

A mission scenario being composed of sequential handling of the distributed tasks is a minimum service unit in the view point of customers. In the extended case of the above example, "Holding at a Room" can be a mission scenario that is composed of a task, "Moving" and another following task, "Holding". The mission scenario layer is not a implementation layer, but a logical layer implemented as a administrating task with a specialized behavior module. The behavior module has only service-call statements through server/client communication and some sequencial control statements for the time and event synchronousness among the processes of tasks.

Usually the behavior module implementing the mission scenario is embedded in a brain-like robot platform. Logically centralized form of the mission scenario enables the scenario to be clear to customers and still holds the benefits of distribution by decentralized process of tasks.

The relationship between mission scenario and distributed task is shown in Fig. 6.

3.2 Personal Robot Design Center Platform

Personal robot design center is a platform for installing, maintaining, and replacing two or more home networked robot platform on internet or home network.

Installation of robot platform means software composition by late binding of behavior modules and consequently, implementation of distributed tasks. Maintenance means development of mission scenarios, development of higher-level behavior modules and installation of the modules. Replacement means replacement of exsisting behavior modules on robot platforms. Concept of the personal robot design center with its communication framework is shown in Fig. 6.

3.2.1 Communication Frameworks

When a personal robot design center is located on internet, the personal robot design center can access home network indirectly through home server. Consequently, a home server is only the node that communicates with personal robot design center on internet. The communication protocol between a home server and a personal robot design center is FTP(File Transfer Protocol). Personal robot design centers can upload robot platform template files and can download updated robot platform template files again with intermediate codes through the home server if required.

The home server can gather all robot platform template files from all robot platforms connected on the home network, and also distribute template files and intermediate codes to specified robot platforms. The communication protocol between the home server and the robot platform is local version of FTP. The local FTP and inter-broker protocol mentioned previously are application protocols based on the same lower protocol stacks. The personal robot design center also has the local FTP that enable the robot design center can be a node on home network.

3.2.2 Late Binding

Late binding is implemented with updating robot platform template files. The robot platform template files can be uploaded through the internet FTP or the local FTP, and can be downloaded again for communication brokers and dynamic thread loader to refer them.

The personal robot design center provides both a GUI environment and a text environment, and they can be exchanged with each other.

The GUI environment proposed here is a VISIO[14]-based application implementing the VISIO object clients. The VISIO acts as COM servers and provides Automation. The VISIO provides good solution for late

binding analysis. The binding is implemented using The VISIO Connection objects, and the object provides wealthy properties that can enable analysis more accurately. Analyzed results are updated on robot platform template files.

A Shape object in the VISIO means a behavior module in the personal robot, and a Sheet object means a sub-network of one or more robot platforms connected.

3.2.3 Development of Higher Behavior Modules

Development of higher behavior modules is same as that of lower modules. The personal robot design center provides a virtual machine source code editor and a compile environment. The personal robot design center investigates the periods of higher level behavior modules, and assesses the RMS schedulability referred to robot template files that contain periods of all behavior modules. When the real-time processing is anticipated to be violated, the personal robot design center resets the period with user's notification.

3.2.4 Development of Mission Scenarios

Development of mission scenario is a special case of higher level behavior module development with only the server/client communication excluding the publisher/subscriber communication. The period of the behavior module is assigned by the personal robot design center. The robot design center provides not only the virtual machine source code editor and the compile environment, but also the GUI environment using the VISIO in the format of classical flow-chart.

3.2.5 Replacement of Behavior Modules

The replacement of behavior modules is run-time replacement mechanism. A dynamic thread loader on a robot platform performs thread loading and unloading without intervention of users. Thread loaders investigate robot platform template files and extract version information from them, and make decision of replacement, elimination, and the addition of thread.

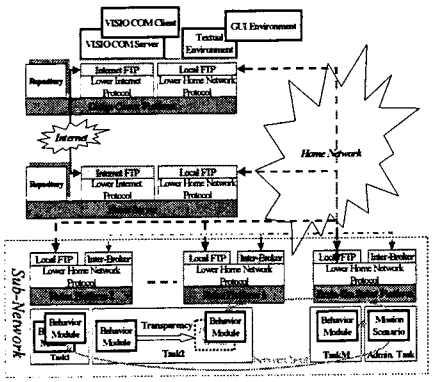

Fig. 6. Concept of Personal Robot Design Center

4. Conclusions

In this paper, a personal robot software framework and its mechanisms are proposed.

The personal robot software framework is layered structure, composed of a transparence layer, a behavior layer, a distributed task layer, and a mission scenario layer. Through the personal robot software framework, the interoperability and distributed task implementation can be achieved.

In addition to the personal robot software framework, for the rapid installation, maintenance and replacement of personal robot platforms, the concept and the mechanisms of the personal robot design center platfrom are proposed in this paper with its GUI features.

As further researches, real-time control features of the personal robots with the personal robot software framework should be studied. Also, run time debugging and simulation mechanism will be proposed with the robot design center platform.

References

[1] Rodney A. Brooks, "A Robust Layered Control System for a Mobile Mobile Robot", *IEEE Journal of Robotics and Automation*, Vol. RA2, No. 1, pp.14-23, 1986.

[2] Beck, J.E., J.M. Reagin, T.E. Sweeney, R.L. Anderson, and T.D. Garner, "Applying Component-Based Software Architecture to Robotic Workcell Application", *IEEE Transactions on Robotics and Automation*, Vol. 16, No. 3, pp.207-217, 2000.

[3] Thomas Jensen, Daniel Le Matayer, and Tommy Thorn, "Security and Dynamic Class Loading in Java: a Formalisation", *IEEE Proceedings on Conference on Computer Language*, pp.4-15, 1998.

[4] D.C. Schmidt, D.L. Levine, and S. Mungee, "The Design and Performance of Real-Time Object Request Brokers", *Computer Communications*, Vol. 21, pp. 294-324, 1998.

[5] Raza S. Raji, "End to End solutions with Lonworks Control Technologies", *http://www.lonmark.org/solution.*

[6] J. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior", *IEEE Proc on Real Time Symposium*, pp.166-171, 1989.

[7] H. Chetto and M. Chetto, "Some Results of the Earliest Deadline Scheduling Algorithm", IEEE Transactions on Software Engineering, Vol. 15(10), pp.1261-1269, 1989.

[8] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority Inheritance protocol: An Approach to Real-Time Synchronization", *IEEE Transaction on Computers*, VOL 39, No.9, pp.1175-1185 September 1990.

[9] Richard Monson-Haefel, Java Message Service, *O'reilly Press*, 2000.

[10] Ragunathan Rajkumar, Mike Gagliardri, and Lui Sha, "The Real-Time Publisher/Subscriber Inter Process Communication Model for Distributed Real-Time System: Design and Implementation", *First IEEE Real-Time Technology and Applications Symposium*, 1995.

[11] Sun Microsystems, RPC: Remote Procedure Call Protocol Specification Version 2, *http://www.faqs.org/rfcs/rfc1831.html, 1995.*

[12] William Grosso, Java RMI, *O'reilly Press*, 2001

[13] Tim Lindholm, Frank Yellin, The Java Virtual Machine Specification, *http://java.sun.com/docs/*, 1996.

[14] Microsoft Corp., Developing Microsoft Visio Solutions, *Microsoft Press*, 2001.