Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

The Software Architecture for Module-based Robot System Supporting Heterogeneous Network Interfaces

Kwang Koog Lee*, Seong Hoon Kim*, Vitaly Li, Sun Hee Choe*, Hong Seong Park*, Sung Hoon Kim
Jung Bae Kim (RSA)****

**Electronics and Telecommunication Engineering Department, Kangwon National University, Chun-Cheon, Korea, (e-mail: kwangkooglee@gmail.com, {bs99018, shchoe}@control.kangwon.ac.kr, hspark@kangwon.ac.kr)*
*** Intelligent Robot Research Division, Electronics and Telecommunication Research Institute, Daejun, Korea, (e-mail: {saint, jjkim}@etri.re.kr)*

Abstract: On developing modern robot systems, intellectual robots can be designed as multiple modules, where the module means the autonomous hardware units performing robot specific tasks. Further, each module can be connected with non-unified network interfaces due to a heterogeneous feature of robot system. In this manner, one of technical challenges is interoperability to support stable and effective communication between such disparate devices. Focusing on this issue, this paper proposes a middleware named HERM (Heterogeneous nEtwork-based Robot Middleware). HERM is divided into three layers; (i) Network Interface Layer, which abstracts various heterogeneous network interfaces as logical channels, (ii) Network Adaptation Layer, which provides addressing strategies and routing service for communication between modules, and (iii) Application Support Layer, which manages robot applications and transforms application data into a standard format for transmitting over a network. By this layered design, HERM provides standardized interfaces to control various heterogeneous network devices, supporting transparent and facilitates integration of different module which constitute a robot system. The results of implementation and experiment show that HERM is suitable for a module-based robot.

1. INTRODUCTION

Recently, the interest of robotics research has been moving from conventional industrial robots to intellectual service robots (Cho et al. 1999). Traditional robots focusing on uniform and repeatable tasks are viewed a centralized system, which processes all tasks in a processor. On the other hand, the intellectual robots are equipped with a number of sensors, actuators, and vision devices to provide diverse services. Since the number of devices that can be connected to a robot is limited to the number of I/O ports a processor supports (Taira et al. 2005), the intelligent robots might be difficult to be designed as only one centralized system. In this manner, they can be designed as a module-based system, where the module means the autonomous hardware units performing robot specific tasks as shown in Fig. 1. The modular robotics systems model robot resources as distinct functional modules. In addition, they might be efficient, both in parallelism implied by many resources operating at once and in the increased usage of resources (Fryer et al. 1997; Dubek et al. 2003).

In a modular robot system, each module can be connected with a variety of heterogeneous I/O interfaces. Actually, since bandwidth required for distinct task execution in each module might be different, there is high demand for heterogeneity of network interfaces. This makes the development of robot's applications non trivial task, because developer often cannot predict on what platform the application would be deployed and what network interface will be available. In consequence, the integration of such heterogeneous devices is required for modular robotics research.

In general, the solution for integration of heterogeneous system is to use middleware (Bermstein 1996). There are a number of middleware solutions, which have been studied in present like DCOM, ICE, and CORBA. However, they have lack of support for heterogeneous network. Even though there exist several network interfaces which supports IP like IPv6 over IEEE 1394, those middleware does not consider mandatory network such as CAN for real-time control of robot and cannot facilitate to attach new device, not supporting IP into the given framework. In addition, the main focus of middleware so far is to provide application-specific services. Therefore, conventional middleware seems to be unsuitable for a robot system supporting various heterogeneous network interfaces such as USB, IEEE 1394,

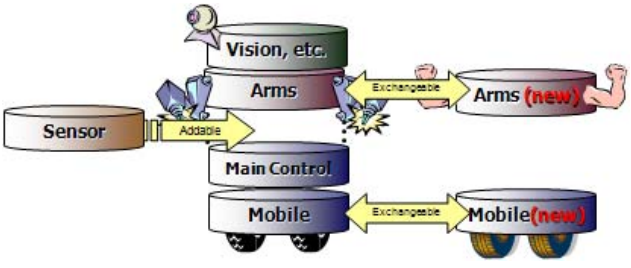


Fig. 1. The Concept of Modular Robot System

CAN, RS232C and Ethernet. In this manner, it is necessary to develop new middleware supporting interoperability for module-based robot connected through various network.

In this paper, we propose a communication middleware named HERM (Heterogeneous nEtwork-based Robot Middleware) for a module-based robot system with various heterogeneous network interfaces. HERM consists of the three core parts; (i) Network Interface Layer (NIL), (ii) Network Adaptation Layer (NAL), and (iii) Application Support Layer (ASL). By this layered design, HERM provides standardized interfaces to control various heterogeneous network devices, supporting transparent and facilitates integration of disparate modules which constitute a robot system. The results of implementation and experiment show that HERM is suitable for a module-based robot.

This paper is organized as follows. In the following Section, we explain the concept of distributed module-based robot system. We then present our proposed middleware architecture in Section 3. Section 4 explains interoperability between modules. In Section 5, we show implementation of our middleware. Conclusions are drawn in the last Section.

2. DISTRIBUTED MODULE-BASED ROBOT SYSYEM

This section considers a distributed module-based robot systems. The term "module" is used to define a self-contained component of a robotic system that provides a robot-specific service. Usually modules are associated with hardware components assuming that one hardware component provides one robot-specific service. Such association is derived from real-time requirements of most robot tasks. However, since an application area of intellectual robots could demand non-real-time tasks assuming that one hardware component could contain several software components with each providing a robot-specific task. Therefore, further on term "module" can also regards a software component that provides a robot-specific task. In this paper, we define the module as an autonomous hardware unit performing robot specific tasks.

There are two main approaches for designing robotic systems from the point of view of hardware architecture.

The first one is a centralized architecture and the second one is a distributed architecture (Taira et al. 2005). The centralized architecture views a robot as a single hardware part containing one or several modules. The centralized architecture has an advantage of simplicity in a software development. However, there are many weak points to distributed approach such as the adaptability of such robotic systems would decrease with increasing a complexity and a number of peripheral devices. Also reliability of a whole system would depend on how well a central part performs its services. The distributed architecture views a robot as a network of several autonomous modules. Each module contains one or several I/O interfaces. Modules could freely be added or removed from network providing an expansion or a reduction of functionality of a robotic system.

Advantages of the distributed architecture are a potential for adaptability and reliability based on a redundancy of services from the point of view their physical location. The main disadvantage of distributed systems is complexity in software design.

This paper considers robot as a distributed architecture. The robot is a network of independent modules connected by network interfaces with each other. Currently, there are several network interfaces suitable for use in robotic systems, such as Ethernet, USB, CAN, RS232 and IEEE1394. Several interfaces could be used for connecting different types of nodes according to different demands. For example, CAN interface could be used to meet high reliability demands while Ethernet could be used for an integration of a robot to the home LAN and IEEE1394 for multimedia purposes. Other reason for use of several network interfaces is to provide a redundancy of connections between modules in order to meet reliability demands of a robotic system. In order to achieve modularity it is necessary to provide an interface for modules communications. Such interfaces are known as middleware (Bernstein. 1996).

The middleware for distributed module-based robot systems should handle following tasks: handling addition and removal of modules, abstraction of heterogeneous network interfaces, modules discovery and routing between modules, communication protocol and marshalling.

This paper proposes a middleware for distributed module-based robot systems with heterogeneous network interfaces called HERM (Heterogeneous nEtwork-based Robot Middleware). Following section describes the architecture of HERM.

3. THE ARCHITECTURE OF HERM

In this section, we describe the software architecture of HERM in detail. As shown in Fig. 2 below, HERM consists of three layers; Network Interface Layer (NIL), Network Adaptation Layer (NAL), and Application Support Layer (ASL). NIL is responsible for abstracting various heterogeneous network interfaces. It defines a logical unit

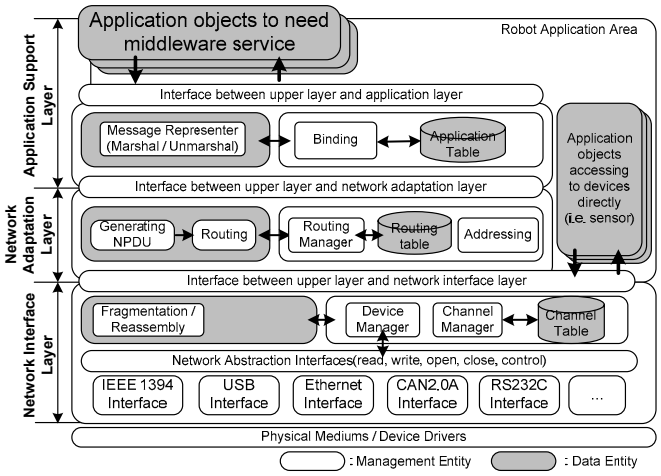


Fig. 2. The Architecture of HERM.

called channel and wraps network-dependent programming interfaces to control each network interface up in interfaces which channel provides such as open, control, read, write, and close. On the other hand, NAL copes with communication between modules in a module-based robot linked through the disparate devices. For stable and effective communication, NAL provides addressing strategies and routing service for which network-dependent properties such as bandwidth and maximum transmission unit (MTU) are considered. Finally, ASL registers robot applications presented in a local or remote module into an Application Table and periodically manages them. Further, it takes a role in presenting application data into a standard format for transmitting over a network. The following Sub-Sections give more detailed description of each layer in sequence of bottom-to-top.

3.1 Network Interface Layer

Network Interface Layer (NIL) is mainly responsible to provide a unified access to various network interfaces such as CAN, IEEE 1394, USB, RS232C, and Ethernet to an upper layer. In other words, none of the upper layer concerns what kinds of network interface are connected. To abstract network interfaces, NIL creates or terminates a logical channel along with a request from the upper layer. After creating the channel, NIL stores and maintains it in a Channel Table. Each channel wraps network-dependent interfaces to control real network interfaces and provides common control interface such as open, control, read, write, and close. In addition, fragmenting and re-assembling messages are also responsibility of NIL because, in some case, the size of a message is not suitable for particular network interface like CAN.

3.2 Network Adaptation Layer

Network Adaptation Layer (NAL) provides addressing strategies to indentify each module and routing service for communication between modules. By static or dynamic addressing strategy, a module should assign a unique address that allows identifying and locating any other modules. Static addressing is to directly assign it through a pre-defined address pool, while dynamic scheme needs negotiation with other modules. After addressing, each module recognizes other modules through a routing algorithm based on distance-vector. Especially, that algorithm supports multiple routing metrics including hop count, bandwidth, and MTU to consider heterogeneous network interfaces.

3.3 Application Support Layer

Application Support Layer (ASL) manages robot application objects (APOs) and performs binding between APOs. In ASL, application objects are separated into two types; remote and local. Remote type of APOs acts as a proxy APOs to provide transparency to local APOs. Hence, an actual execution of a proxy APO is fulfilled in a remote APO and then an execution result of the proxy APOs is asynchronously

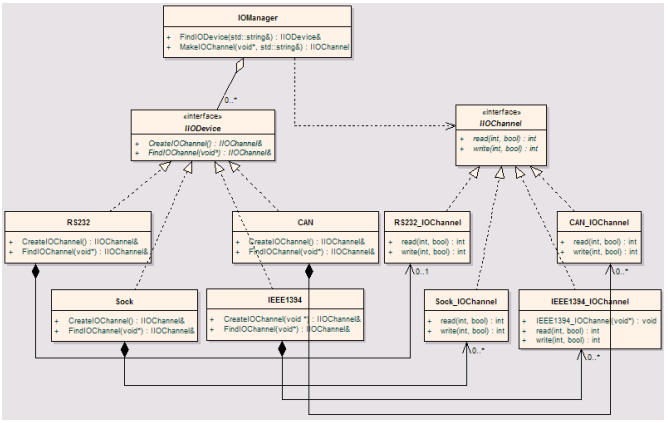


Fig. 3. A class diagram indication relationship among diverse devices and channels (simple diagram)

forwarded to the local APO executing the proxy APO through call-back interface after the completion of the execution request. With regard to binding between APOs, ASL is capable of discovering other APOs by issuing queries based on broadcast and of binding local APOs with the discovered APOs.

4. INTEROPERABILITY BETWEEN MODULES

4.1 Abstracting heterogeneous network interface

When a local module perceives a new module, NAL need a way to access to the new module. And NAL should use the network interface transparently without concerning kinds of network interfaces. In this respect, NIL provides IOChannel abstracting heterogeneous network API as a logical channel. A class diagram indicating relationship between devices and channels is shown in Fig. 3. As shown in Fig. 3, to make diverse devices as a common interface, design patterns (Gamma et al. 1999) such as abstract factory, factory method, and flyweight are applied. And for special channels capable of fragmenting and re-assembling messages, decoration pattern (Gamma et al. 1999) is employed.

On a basis of the structure, interaction among devices and channels to create a new IOChannel is preceded as illustrated in Fig 4. When a client, which can be NAL, ASL, or APOs, wants make a new IOChannel, it requests creation of a new channel with a common device address and a device name to IO manager responsible for managing IODevices and

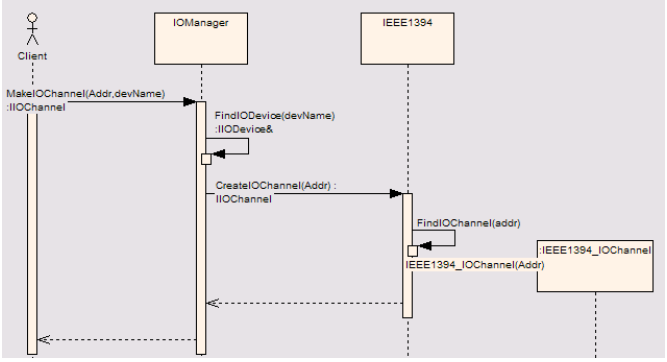


Fig. 4. Interaction Diagram of Creating IOChannel for IEEE1394 (simple diagram)

IOChannels. On receipt of the request, the IO manager finds an IODevice having the same name as the passed device's name. Note that, from the viewpoint of IO Manager, the found

IODevice is not the type of IEEE1394 IODevice but the type of a base class that abstracts details of diverse network interface. Finally, the IO manager forwards the request to the IEEE1394 IODevice. The IEEE1394 IODevice creates a new IOChannel using the passed device address and subsequently returns the IOChannel to the clients.

4.2 Module Discovery

Before recognition of remote modules, each module has to be identified as a unique address. To assign an address, NAL selectively uses two kinds of strategies. As the first strategy is static module addressing, each MId is assigned through pre-defined addresses to the categorized modules such as main controller module, right/left arm module, vision module, or sensor module. On the other hand, the dynamic addressing strategy is accepted for the case that behaviour of a module is not categorized. In this strategy, a module randomly creates a MId and broadcasts message with the generated address to all modules over a network. If another module is already using the corresponding MId, the module broadcasts to inform duplication of the address. Then, the module created the duplicate address randomly creates a new MId and broadcasts the address information until there should be none address collision.

The routing of HERM is similar to distance-vector routing protocol. Each module constructs a vector containing the costs (distances) to all other modules and a forwarding list in which passing network interface's information is contained (i.e. one cost and one forwarding list is a pair to build the routing table) and gives that vector and list to immediate neighbor modules via all channels. The module receiving the pairs, then updates own routing table. A routing table contains the following information; (i) destination module address, (ii) next channel index, (iii) routing costs, (iv) hop count, and (v) forwarding list. If the pair information has new destination, the module adds a routing entry into own routing table and put corresponding information into it. For example, when a module receives the pair, (8, CAN2.0) for the remote module, 0x42 via the second channel with Ethernet interface, it adds the routing entry, (0x42, 02, 8 plus computed cost, 2, CAN2.0-Ethernet). Even though a module receives that information by multiple channels, it has to update that routing table as well. In other words, multiple paths can be generated in a routing table. In addition, the pairs according to passed interface's bandwidth and MTU are also updated. After updating, a module also sends the updated pairs whenever a triggered update from another module causes it to change its routing table. In accordance with such repeatable procedure, all the modules recognize other module's existence in a robot system.

4.3 Routing Scheme between modules

When data reaches arrives at NAL, a module needs to be able to look at a data's destination and then determines which of the channels is the best choice to get the data. The module makes this decision by consulting a routing table. The routing algorithm of HERM supports multiple routing metrics including hop count, bandwidth and MTU to find the lowest-cost path between any two modules connected through heterogeneous network interfaces. To compare routing costs between multiple routes, a formula (1) below is adjusted, wherein n, c, D, M, B equals to hop counts, relative coefficient for delay, total size of packet, MTU, bandwidth.

$$\sum_{i=0}^{n} 10 \times c_i \times \lceil D_i / M_i \rceil \times \log(10^8 / B_i) \tag{1}$$

Routing metrics for each interface is given in Table 1. On routing, the cost can become different whenever data sends to any module, because it depends on the total size of packet as known through that formula above. To provide efficient routing, static costs for routing maintain in routing table. It is derived from a case that Di/Mi always equals to 1. After NIL receives the data, it checks whether fragmentation is necessary. If so, the data is divided into several packets and is sent to the desiring module through a physical medium of the corresponding channel.

Table 1. Routing Metrics for Network Interfaces

Interfaces	Bandwidth(bps)	MTU(byte)	Cost
UBS 1.1	12,000,000	1023	19
USB 2.0	480,000,000	1024	3
IEEE 1394a	400,000,000	4096	4
IEEE 1394b	800,000,000	8192	1
CAN 2.0 A / B	1,000,000	8	30
ETHERNET	100,000,000	1500	10
RS232C	19,200	32 (user-defined)	47

5. IMPLEMENTATION AND EXPERIMENTS

The proposed structure has been implemented using C++ with Ethernet, IEEE1394, CAN 2.0A, and RS232C as a network interface. To exemplify HERM, we developed a remote application for regularly transmitting sensed value to

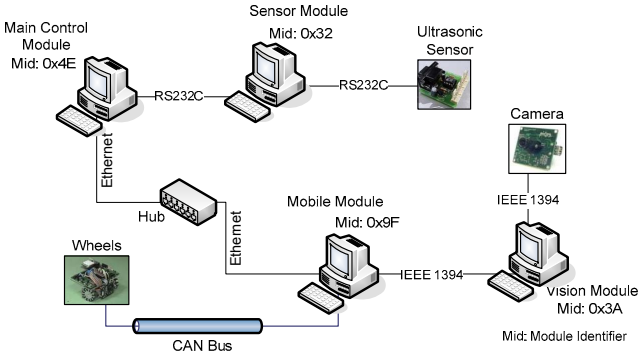


Fig. 5. Testbed Configuration for HERMES's Experiment

any remote module, and a local application reading sensor value.

5.1 Testbed Configuration

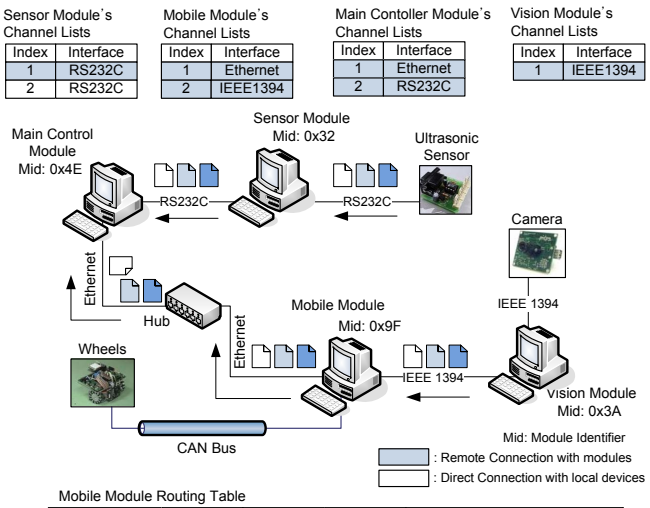
In order to demonstrate HERM, we have built up a testbed. As shown in Fig. 5, four modules are connected with heterogeneous network interfaces, RS232C, IEEE1394, Ethernet, and CAN. The system specification for each module is given in Table 2 running the Windows XP operating system. Additionally, one embedded board with AVR Atmega 128 MCU is connected with the sensor module via RS232C and it contains SRF04 ultrasonic sensor. Note that we assume wheel and camera depicted in Fig. 5 virtually exist.

Table 2. System Specifications for Testbed

Module	System specifications
Sensor	Pentium IV 1.6GHz, 512MB (RAM)
Main controller	AMD Sempron™ 2800+ 1.61GHz, 1GB
Vision	Intel Pentium IV 1.5GHz, 512MB
Mobile	AMD Sempron™ 2800+ 1.61GHz, 512MB

5.2 Experimental Scenario

In this subsection, we show how HERM In this sub-section, we show how HERM supports interoperability in a module-based robot system linked with non-unified interfaces. To verify interoperability, we experimented on a scenario about communication between modules. In scenario, every module periodically transmits sensed data to the main controller module via each network interface. As shown in Fig. 6 below when the vision module sends data, it has to pass through mobile module, because there is any interface to directly connect with main controller module. Therefore, the mobile module has to be operated as a router module. We assumed all the modules assign their addresses by the static addressing strategy introduced in Section 4.



Mobile Module Routing Table

Dest. Module	Next Ch.	Hops	Status	Forwarding Channel List
0x9F	-	0	Active	-
0x4E	1	1	Active	Ethernet
0x3A	2	1	Active	IEEE 1394
0x32	1	2	Active	Ethernet / RS232C

Fig. 6. Experiment Procedure

Ultrasonic sensor regularly sent a sensed value every 30 seconds to the sensor module via RS232C. This value arrived at the local application object in the module. For delivering this value to a remote module, we set that if the sensor module gathers ten sensing values, it starts to send them to the main controller module, using a remote application object. When the sensor module sent the data through the second channel, RS232C, it was fragmented into two packets because of MTU. On the other hand, the vision module also sent the vision information to main module via the mobile module every 60 seconds. To cope with this, we set timer in the remote application in the mobile module. We assumed that the size of the information is 100 bytes. This data was sent through the first channel, IEEE 1394. After the mobile module received the data from the second own channel, IEEE 1394, it then reassembled two packets into one. Since the data is not for the mobile module, it starts to routing service. As shown in Fig. 6, the mobile module maintains a routing entry for destination of the main controller module, 0x4E. Therefore, the data was forwarded through the first channel, Ethernet. Finally, the main controller module periodically received data from distributed modules. As we have shown in this section, the proposed middleware works well under heterogeneous and distributed environment.

5. CONCLUSIONS

Module-based robot concept enables a robot system to make a distributed network-based structure. Due to feasible structure that a module can be easily exchanged, added, or removed, the module-based robot system contributes on reducing complexity and engineering costs followed by system integration, operation, and maintenance. Focusing on module-based robot, this paper proposed the middleware called HERM (Heterogeneous nEtwork-based Robot Middleware). HERM consists of three layers; Network Interface Layer (NIL), Network Adaptation Layer (NAL), and Application Support Layer (ASL). Based on the layered design, HERM provided standardized interfaces to control various heterogeneous network devices, supporting transparent and facilitates integration of disparate modules which constitute a robot system. To verify the proposed architecture, it has been implemented using C++ and some experiment scenarios have been achieved. Consequently, HERM can be adaptable for module-based robot system and help robot application developers to weave applications without dependency of network interfaces.

Acknowledgement

This work was supported by the IT R&D program of MIC/IITA. [2005-S033-03, Embedded Component Technology and Standardization for URC]

REFERENCES

CAN specification Part A and Part B.

Distributed Programming with ICE (2007) (Internet Communications Engine), Revision 3.2.1, http://zeroc.com

DCOM (Distributed Component Object Model), Microsoft.

IEEE standard for a High Performance Serial Bus, IEEE std 1394-1995, IEEE 1394 std 1394a-2000.

J. A. Fryer, G. T. McKee, and P. S. Schenker. (1997). Configuration Robots from Modules: an Object Oriented Approach" *IEEE International Conference on Advanced Robotics (ICAR `97)*. Monterey, Canada.

Gamma. E, Helm. R, Johnson. R. and Vlissides. J. (GoF, Gang of Four), Design Patterns: Elements of Reusable Object-Oriented Software, ISBN 0-201-63361-2

G. Dubek and R. Sim (2003). RoboDaemon – A Device Independent, Network-oriented, Modular Mobile Robot Controller, *IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, Sep. 2003. pp. 3434-3440.

M. T. Kaikkonen, and J. Hakala (1991), Interfacing Functional Modules within Mobile Robots, *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 1991)*

Object Management Group. (2002) CORBA: Common Object Request Broker Architecture: Core Specification Revision 3.0. http://www.corba.org

P. A. Bernstein (1996). Middleware: A Model of Distributed System Services, *Communications of ACM*, **Vol. 29**. No. 2.

RFC 3146: Transmission of IPv6 Packets over IEEE 1394 Networks. 2001.

T. Taira, N. Kamata, N. Yamasaki (2005). Design and Implementation of Reconfigurable Modular Humanoid Robot Architecture. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005)*, Edmonton, Canada, pp. 3566-3571.

Y. J. Cho, B. J. B. You, S. R. Oh, and C. W. Lee (1999). A Compact/Open Network-Based Controller Incorporating Modular Software Architecture for a Humanoid Robot. *Journal of Intelligent and Robotics Systems,* **Vol. 25,** 341-355.

Universal Serial Bus Specification revision 2.0 (2000).