Development of a Network-based Real-Time Robot Control System over IEEE 1394: Using Open Source Software Platform

M. Omar Faruque Sarker, ChangHwan Kim, Jeong-San Cho and Bum-Jae You
Intelligent Robotics Research Center
Korea Institute of Science and Technology(KIST)
P.O. Box 131, Cheongryang, Seoul, 130-650
Korea
writefaruq@gmail.com {chkim,chojs,ybj}@kist.re.kr

***Abstract* – Network-based distributed processing and hard real-time requirement significantly increase the complexity of a robot hardware and software system. This paper describes the open and modular approach to design and implement a real-time robot control system using Open Source Software (OSS) which include a Linux-based Real-Time Operating System (RTOS), Xenomai (formerly known as the fusion branch of Real Time Application Interface, RTAI/fusion), real-time IEEE 1394 device driver and other middleware components. The real-time performance of our system has been evaluated in a mobile robot control platform and we have compared the performance of our system with that of a commercial RTOS, RTLinux Pro, based system using the same set of hardware. From the comparison it is seen that our system is also capable of satisfying the similar hard-real time requirements with a greater flexibility in overall implementation. In this work, the high performance IEEE 1394 serial bus is used as the field-bus of our robot control system. Though IEEE 1394 (also known as FireWire) is widely used in consumer electronics or home video network we have successfully adopted this emerging technology in robot control and as a side effect of our research, we have seen that IEEE 1394 can be used as a better alternative to the old generation slower filed-bus like CAN bus.**

I INTRODUCTION

Rapid technological developments in computer, control and communication technologies make the modern robotic systems more sophisticated and complex. In order to satisfy the demand of high processing power of this complex system and to keep implementation cost minimum, network based robot control systems have been introduced [1], [4]. Early mobile robot control systems had no significant real-time requirement in their implementation because of their simplicity in design and practical application [1]. But modern sophisticated mobile robots, human-shaped biped robots (*humanoids*) and almost all complex robotic systems require a very high deterministic control and predictability in their actions and behaviors. Like other real-time systems, these robotic systems are also supposed to guarantee a precise timely response to an external event where either all deadlines must be met to prevent a fatal fault (hard real-time) or a few deadlines can be missed occasionally by compensating overall system performance (soft real-time) [3]. In such a scenario, the essential hardware and software components of a complex robotic system are manufactured by different vendors and are used by different user groups with different application requirements.

So, various Open Architecture Control (OAC) systems were introduced during the last decade [6]. Though, in this way, the openness issue is limitedly addressed most of the proprietary implementations still have serious disadvantages as they do not provide the source code of the underlying software system and also some systems are highly expensive for mass level implementation.

Having the best hardware in hand, it is not easy to develop the desired modular, scalable or general purpose robot control system with a proprietary expensive RTOS (e.g. QNX, VxWorks etc) [12]. Consequently, this phenomenon prevents the robot manufacturers from offering wide-range of real-time robotic systems to the common market. However, the appearance and growing maturity of the open source Linux-based RTOS like RTLinux [15], RTAI [16], Xenomai [17] and related other software projects like Orca [18] , TAO [19] have created the opportunity for the researchers to use Open Source Software(OSS) in robot control system. So, the constant demands for implementing these software in new hardware platforms, using emerging technologies like IEEE 1394, have been created to support many promising applications of the sophisticated robots [21].

In this paper, using these freely available OSS technology, an open architecture for network-based secure robot control system along with the IEEE 1394 serial bus based example implementation is presented. At the start of work, we design the architecture in an implementation-independent manner. Then we proceed to our implementation by selecting Linux based RTOS Xenomai and IEEE 1394 real-time device driver from RT-FireWire Project [22]. Upon these layers we also demonstrate a secure application model and an example robot control program running in our system.The real-time performance of our system has been evaluated in our mobile robot control platform and we checked the feasibility of our system by comparing the performance of our system with that of a similar commercial RTOS, RTLinux Pro, based system.

Rest of this paper: Section 2 presents our system architecture. Our specific implementation is described at Section 3. At Section 4, we have evaluated the performance of our system by a real-time application. An overview of relevant work is summarized at Section 5. Finally, summary and outlook are placed at Section 6.

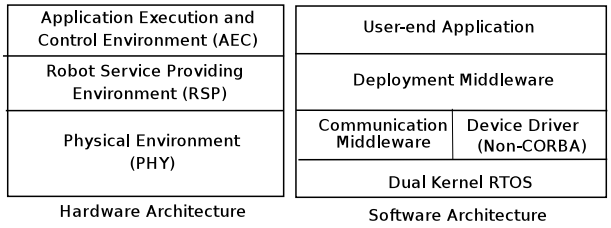

Figure 1
System architecture

II System Architecture

As our complex robot control system has to deal with different hardware and software vendors, application developers, system integrators and end-users, so at first, we have defined our open hardware and software architecture to minimize those integration gaps.

A. Hardware Architecture

Three distinct hardware environments, as illustrated in Fig.1, have been proposed:

1) Robot Physical Environment (PHY): contains the robot itself and any supporting peripheral connected with it. All internal electro-mechanical systems are managed and regulated in real-time by the software working in this environment.

2) Robot Service Providing Environment (RSP): consists of different application and network servers which provide the middleware applications to the end-users by using the hardware resources in PHY environment. A main RSP server, acting as a Gateway to the Robot Service Providers (GRSP), provides security and authentication services to the available resources in the network.

3) Robot Application Execution and Control Environment (AEC): is the client layer on top of RSP and any useful service offered by RSP can be used by this layer in a remote PC or laptop or even in a hand-held PDA device.
In this model, direct user access to the robot (in PHY) from AEC is discouraged except for maintenance and emergency-handling by the super user or robot owner. This ensures safe and secure usage of the system.

B. Software Architecture

Our component based software architecture can be divided into four layers as illustrated in Fig. 1. We have designed this open architecture considering the different open standards like RT-CORBA v1.1[23] which has already been implemented by different OSS projects (e.g. TAO [19]).

1) RTOS: The RTOS is responsible for providing the real-time guarantees to the upper layers of the architecture. Many existing real-time systems use proprietary RTOSes and their custom applications. So a flexible OSS RTOS should be capable of emulating other RTOSes to support rapid application development and porting to a target RTOS. In order to keep this flexibility in our design, we have sub-divided our RTOS layer into three sub-layers:

i. RTOS Hardware Abstraction Layer (HAL), to facilitate the abstraction between the operating system kernel and the underlying hardware. All hardware interrupts are managed in this layer.

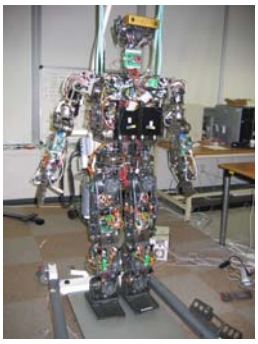

Figure 2
A walking humanoid skeleton

ii. RTOS Nano-Kernel Layer is the heart of the RTOS and it provides the required RTOS facilities to the real-time applications: real-time memory management, interrupt-handling, scheduling, timer management and other important real-time needs of the system.

iii. RTOS API and Emulation Layer is the means to communicate and interface between the real-time applications and the Nano-kernel of the RTOS.

2) Communication Middleware and Device Drivers: This layer is composed of two sub-layers: the RT-CORBA compliant communication middleware and the non-CORBA device drivers. The non-CORBA part is designed to be integrated with the RTOS and it also resides in the Physical Environment (PHY) of the hardware architecture to provide the basic hardware service primitives to the upper-layer satisfying the real-time requirement.

3) Deployment Middleware: This layer is open to the application developers to utilize the communication middle-ware and primitives offered by the device drivers to create any application for exploiting and managing the hardware resources. Any control algorithm or a set of tasks can be implemented in this layer by using the abstracted device objects offered by the device drivers and communication service objects offered by the communication middleware.

4) User-end Application: The top layer of the software architecture is intended for end-user applications which are constructed by using the components of the deployment middleware. These applications can be designed to run in any low-end machine residing in the AEC.

III Implementation

Following the system architecture described in Section 2, we have implemented a walking robot control system using open source solutions. The following sections provide our hard-ware and software implementations as well as application development and porting experience using IEEE 1394 field-bus system.

A. Hardware Platform

1) Robot Platform: To abstract the hardware specific view of our platform we have separated the components of the robot node into several important modules including Main Processing Modules (*Main Controller*), Motor Controller

Table 1
Robot Main Controller specifications

Component	Specifications
CPU	Micro-FCPGA Mobile Intel Pentium M 1.5 GHz
Memory	512MB DDR333 SDRAM
LAN Interface	Intel PRO/1000 GbE
WLAN Interface	Intel PRO/Wireless 2100 (802.11a/b) with Mini-PCI
Real Time Clock	Intel ICH4-M chipset built-in RTC

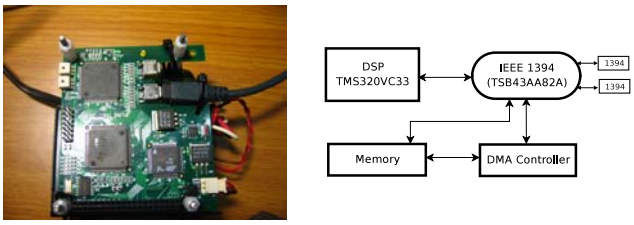

Figure 3
Motor Controller board (left) with block diagram (right)

Module (*Motor Controller*), Sensor Interface Module (*Sensor Module*), Vision Module, etc. A skeleton of our walking robot is shown in Fig. 2. Main Controller of the robot system is designed using Industrial Single Board Computer (SBC). In our work, we have used LV-671 series SBC from Commell Systems Corporation [24] and the FPA-431 IEEE 1394 PCI card from SaroTech [25]. The important properties of our *Main Controller* are given in Table 1. The IEEE 1394 PCI card contains the TSB43AB23 IEEE 1394 chip from Texas Instruments [27] and it is capable of transferring data between the 33-MHz PCI bus and the IEEE 1394 bus at 100M bits/s, 200M bits/s and 400M bits/s. This implementation is fully compatible with IEEE std 1394a-2000 [9]. To construct a *Motor Controller* and *Sensor Module* we have used Digital Signal Processor (DSP) from Texas Instruments and the we have connected these hardware modules with our Main Controller via IEEE 1394 serial bus interfaces provided by these modules. A *Motor Controller* board consisting of TMS320VC33 DSP, TSB43AA82A-EP IEEE 1394 controller and other accessories is shown in Fig. 3 (left). The corresponding block diagram is also presented in Fig. 3 (right).

The TMS320VC33 DSP is a 32-bit, 150 million floating point operations per second (MFLOPS) capable floating point processor. The TSB43AA82A-EP IEEE 1394 controller is IEEE 1394a-2000 compliant integrated 400 Mbps two port Physical Layer which provides automated read response for ConfigROM register access. This is very useful for automated testing and debugging the *Motor Controller* using different RTOS applications. We have used IEEE 1394 serial bus to link between *Main Controller* and the *Motor Controller* or *Sensor Module*. A system configuration of our robot control system is shown in Fig 4. Here it is seen that, 14 hardware modules are connected using 3 ports of the IEEE 1394 PCI card. IEEE 1394 peer to peer host architecture makes it possible to connect up to 63 nodes per bus and a total of 1023 buses in total. So we can connect more nodes easily in this system allowing the scalability and reconfigurability of this system.

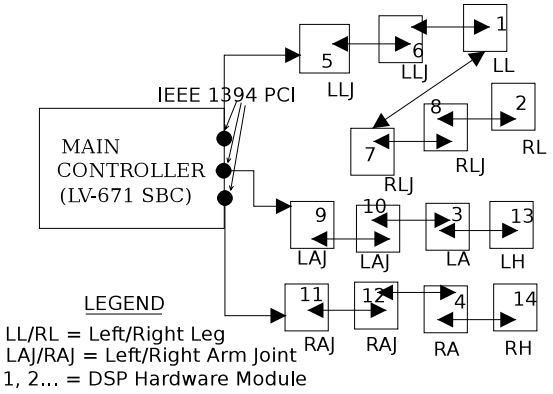

Figure 4
Design of our robot control system using IEEE 1394

The Main Controller of our robot can maintain dedicated connectivity with RSP network servers via wireless LAN. The LV-671 has the flexibility to use Mini-PCI interface to attach an wireless network adapter Intel PRO/Wireless 2100 (802.11a/b). It also has a built-in Ethernet interface Intel PRO/1000 GbE which we have used for static testing with our robot.

2) RSP and AEE Hardware: To implement the RSP environment of our hardware architecture we have used a server class Intel Pentium IV PC having 3.0 GHz processor and 1 GB SDRAM memory. On the other hand, to run user application in AEC environment we have use a low-end Intel Pentium III PC having 500 MHz processor and 128MB memory.

B. Software Platform

A significant recent development of different OSS in real-time applications makes our work easier to select the desired OSS software components and develop our complex application. Firstly, to consider a Linux-based RTOS we have checked different Linux-based RTOSes such as RTAI, RTLinux, Xenomai and KURT and study the comparisons among them [8]. After some initial testing, we have selected dual kernel RTOS, Linux and Xenomai (former RTAI/fusion) for its clear advantages in hard real time support in user-space programs, rich set of APIs for proprietary RTOS emulation and overall flexibility and modularity in design. As the non-real time Linux IEEE 1394 device driver is not capable of providing hard real-time performance [11] so we have integrated the OSS real-time IEEE 1394 driver [22]. We have also proposed an OSS middleware integration which is illustrated in Fig. 5.

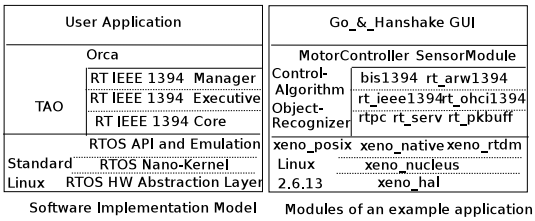

Figure 5
Our software implementation model (left) with an example (right)

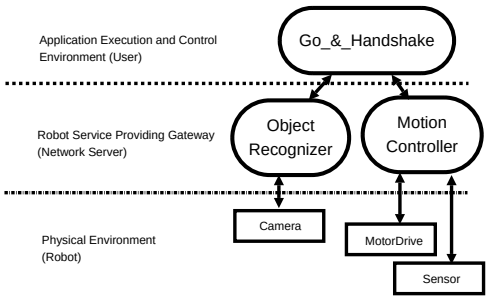

Figure 6
A typical middleware based implementation of our system

1) Linux/Xenomai Dual Kernel RTOS: Generally standard Linux kernel does not provide the hard real time capabilities. Several approaches tried to make the Linux hard real-time capable [3], [8]. Adaptive Domain for Operating Systems (ADEOS) is such an approach [17]. Applying ADEOS new generation i-pipe patch to the standard Linux 2.6.13 kernel on Debian based Ubuntu 5.10 [26] we have compiled and installed Xenomai RTOS. When Xenomai modules are loaded we obtain two domains: the primary real-time Xenomai domain and secondary non-real time Linux domain. All real-time tasks are done in primary domain and ADEOS interrupt event pipe-line sends all hardware interrupts first to this domain. The first Xenomai module, *xeno-hal*, provides these basic hardware abstraction and the abstract RTOS core and then, *xeno-nucleus*, provides all basic real-time facilities. In our implementation, we have used the native skin, *xeno-native*, for user space real-time support and posix skin, *xeno-posix*, to port a POSIX compliant application from RTLinux. Another skin called Real Time Driver Model (RTDM), *xeno-rtdm*, is used by our Real-Time FireWire device driver to interface with Xenomai.

2) Real-Time IEEE 1394 Device Driver: The Real-Time FireWire driver works on top of RTOS layer of our software architecture. we have incorporated this driver with our own applications by the layered sub-modules as shown in the Fig 5. *RT IEEE 1394 Driver Core*, consisting of the modules *rt-pkbuff*, *rt-serv* and *rtpc*, provides the real-time memory management, task scheduling and interrupt handling support to the upper layer of the driver. These modules directly interface with Xenomai RTOS. *RT IEEE 1394 Driver Executive*, consisting of the modules *rt-ieee1394*, *rt-ohci1394* and *bis1394*, provides the FireWire character device interface to user applications, supports OHCI boards and does some basic bus management functions. *RT IEEE 1394 Service Manager* is the top layer, developed in this work as per our requirement, provides the primitives to abstract the functionalities of IEEE 1394 device. For example using the APIs exported by one the modules of this layer, *rt-async-rw*, it is possible to create a *MotorController* or *SensorModule* abstract device in the deployment middleware layer. The *MotorControlle*r will use the asynchronous read and write facilities of this module.

3) Middleware Implementation: In this model, we have proposed the OSS middleware TAO [19] on top of the RTOS layer as the communication middleware as described in Section 2.

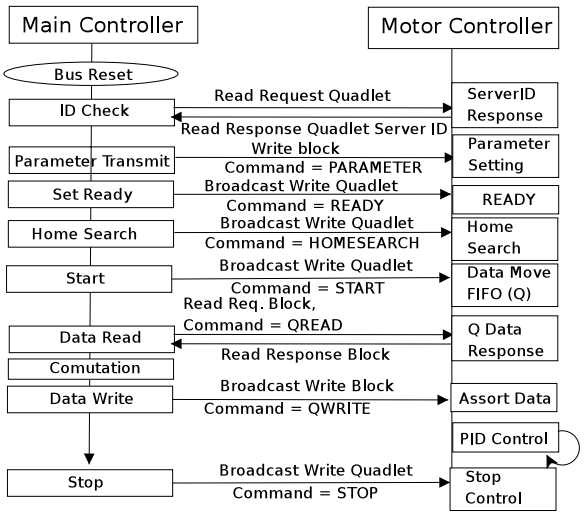

Figure 7
IEEE 1394 based robot control sequences

TAO is a C++ ORB that is compliant with CORBA 3.x specification, which includes the Real-time CORBA specification. On top of TAO we have considered to implement Orca[18] which is an open-source framework for developing component-based robotic systems. Orca has a repository of free, reusable components which can be used to build a general, flexible and extensible system. An example implementation of this system is described in Fig. 6.

4) Implementation of IEEE 1394: IEEE 1394, also known as FireWire [2], is a high performance serial bus for connecting heterogeneous devices. The bus topology of FireWire is tree-like, i.e. noncyclic network with branch and leaf nodes. The physical medium supports data transmission up to 400 Mbps in 1394a specification. Two types of data transaction are supported on FireWire: asynchronous and isochronous. In our work, we have used asynchronous transaction as this provides guaranteed data delivery targeted for non-error-tolerant applications. An asynchronous transaction is split into two sub-transactions: request to access a piece of address on another node and response. Coordination between request and response is ascertained by the transaction layer protocol. A sequence diagram of asynchronous read event is presented in Fig. 8 to show how we have measured the transaction time between two nodes.

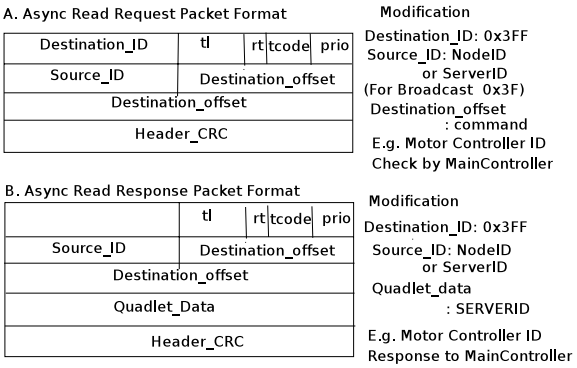

Figure 8
Asynchronous read packet format

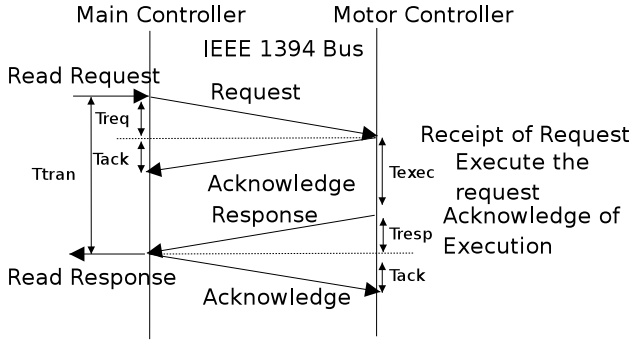

Figure 9
Transaction latency measurement of an asynchronous read event

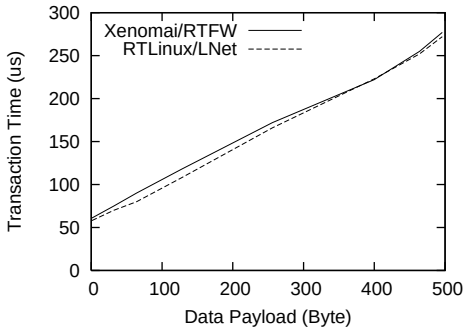

Figure 10
Read-transaction time of Xenomai and RTLinux for varying data payload

C. Robot Control Application

We have considered a simple application to control the movement of arms and/or legs of the our robot as shown in Fig. 2. For this work, three software modules interacts inside the deployment middleware. The *MotionController* module receives necessary parameters from the *ControlAlgorithm* module and sends to the *MotorController* and *SensorModule* (which are abstract devices talking with the RT IEEE 1394 *Service Manager* module, *rt-async-rw*, residing inside the RT IEEE 1394 Device Driver Layer). The *ControlAlgorithm* usually sends Joint data (Q) parameter for MotorController and force and torque data (FT) for the *SensorModule*. The control sequence diagram as illustrated in Fig. 7 describes these cases. To implement this application the hardware modules like Motor Controller or Sensor Module also needs to be prepared and pre-programmed so that it can react correctly on receiving a custom packet, some parts of that packet have some special meaning to do something special.

Here, we have modified the IEEE 1394 quadlet read/write and block read/write packet format to communicate between *MotionController* and *RT IEEE 1394 Service Manager* (*rt-aynsc-rw*). The modified quadlet packet format of IEEE 1394 asynchronous read request and read response is shown in Fig. 8.

Upon each IEEE 1394 bus reset event, each IEEE 1394 node ID is subject to change. To handle this problem we have assigned every IEEE 1394 node (or hardware module) a fixed Server ID (e. g. we have assigned *ServerID*=1 in the source ID offset 0x100 of our first Motor Controller node, *ServerID*=2 in the source ID offset 0x200 of our second Motor Controller node and so on) and this is mapped by our program.

Similarly, each hardware module recognizes the commands as they are preprogrammed in their memory and that is also defined in our program. For example if we send a read request packet using *destination offset = 11* to a node, the node understand that it has to response its *ServerID* as the quadlet-data inside next read response packet. Here *destination offset=11* is interpreted as `ID Check` command to that node.

Table 2
Robot Main Controller specifications

Data Payload (Byte)	Xenomai WRTT (microseconds)	RTLinux Pro WRTT) (microseconds)
4	61	58
32	75	70
128	118	108
256	172	166
496	277	272

IV EXPERIMENT AND EVALUATION

A. Latency Measurement

Real-time performance of our system has been evaluated by a simple robot control program in terms of the time required to travel a physical packet between two nodes. In the ping packet format [2], a *type* field value of *0000b* designates the extended physical packet as a ping packet. Thus the PHY-ID specifies the the node that is to be *pinged*. In response to a ping packet, the target node broadcasts its self-packet(s) and the requesting agent can measure the delay between sending the ping packet and the return of the self-ID packet. So using this information the two way travel-time is calculated as shown in Fig. 9.

Using the standard real-time ping application, *rtping* [22], we have measured the round-trip time (RTT) between the Main Controller and a Motor Controller using different data loads. To compare the performance we have also collected the similar data from RTLinux Pro IEEE 1394 LNet platform [15] using the same set of hardware. Here we have also get the similar results in this platform. Such as, in worst case WRTT of a single read request/response of a quadlet (4 Byte packet) is 58 microseconds in RTLinux/LNet, whereas, in Xenomai system the worst case latency of a quadlet WRTT is 61 microseconds and both cases can satisfy our hard real-time requirement which is 80 microseconds defined in our control application.

B. Performance Evaluation

Here we have found that the additional overhead created in our system is due to the context-switch time between user-space and kernel-space. However we prefer to use user-space programs which are more flexible as they can use native programming libraries.

Since LNet uses kernel-space its execution time is slightly lower, but program debugging is more complex and its capabilities are limited by the proprietary APIs. On the other hand, in case of our system a lot of programming libraries and debugging tools are available in user-space. The results are shown in Table 2. The relation between the WRTT and data payload for both Xenomai and RTLinux based system have been presented in Fig. 10. In both cases, results are satisfactory to maintain the hard real-time of our control system.

V Related work

The OROCOS project [15] was one of the important initiatives for developing robot software framework using OSS. Traditionally, the development is done considering simple hardware and software platform [6], [7]. But for a complex robot like a humanoid, few researchers considered to design an open hardware and software architecture. In [4], a similar architecture named Robot Software Communications Architecture (RSCA) has been presented. In spite of having a good number of services, RSCA is limited to a closed system which also lacks in secure communication.

Specific robot platform based control architectures and applications are described in [7] which have the limitations imposed by their proprietary hardware and software implementations. The performance of a networked robot is also subject to the real traffic conditions of the network. A method is devised by [5] to compensate delay based on RTAI system. Since our implementation uses Xenomai which has the similar architecture and modules with RTAI, so it is quite possible to implement the same delay compensation method in our system.

VI Summary and outlook

In this paper, we have presented a novel approach for developing IEEE 1394 based real-time networked humanoid control system. Because humanoids require much higher level of deterministic control we have shown the effectiveness of IEEE 1394 with the help of two Linux based RTOS platforms. As the field of robotics is rapidly changing, we have focused on the design of an open and modular system so that it can quickly adopt a new promising technology by just replacing the corresponding module of the system and we need not to redesign the entire system for adopting some small changes in hardware and/or software. The main contributions of our work are, to provide a more secure and OSS supported model by discouraging direct access to the robot hardware from the user applications and introducing a robot gateway or a GRSP server and the effective use of IEEE 1394 field bus as a potential alternative of CAN bus.

As our proposed system has significant real-time capabilities it can be used for fast and smooth motion control of a sophisticated robot. It also offers more flexibility and robustness comparing with the traditional RTOS based system as many other RTOS based applications can be ported here.

Acknowledgment

This research has been conducted as a part of the Network-based Humanoid Project funded by the Ministry of Information and Communication (MIC) of the Republic of Korea.

References

[1] Young Shin Kim, Hong Seong Park, Wook Hyun Kwon, “An architecture for a network based robot control system”, *Emerging Technologiesand Factory Automation, 1999. Proceedings. ETFA ’99. 1999 7th IEEEInternational Conference on*, Vol. 2, 18-21 Oct. 1999 Page(s):875-880

[2] D. Anderson, *FireWire System Architecture*, Addison-Wesley, 2003

[3] J. W. S. Liu., *Real-Time Systems*, Prentice Hall, Upper Saddle RiverNJ, 2000

[4] J. Lee, J. Park, S. Han, and S. Hong., “RSCA: middleware supporting dynamic reconfiguration of embedded software on the distributed URC robot platform”, *The First International Conference on UbiCon-trol Laboratory,quitous Robots and Ambient Intelligence (ICURAI)*, pp. 426-437, Seoul, Korea, December 2004

[5] Gustavo Hommerding Alt and Walter Fetter Lages, “ Networked robot control with delay compensation”, Available Online at: *ftp://ftp.realtimelinuxfoundation.org/pub/events/rtlws-2003/proc/lages.pdf*

[6] Dario Dallefrate, D. Colombo and L. Molinari Tosatti, “Development of robot controllers based on PC hardware and open source software”, Available Online at: *ftp://ftp.realtimelinuxfoundation.org/pub/events/rtlws-2005/DarioDallefrate.pdf*

[7] J. L. Gonzlez, E. Baeyens, F. Gayubo, J. P. Turiel and J. C. Fraile., “Open architecture controller for a SCARA Yamaha YK7000 industrial robot”, Available Online at: *ftp://ftp.realtimelinuxfoundation.org/pub/ events/rtlws-2003/proc/gonzalez.pdf*

[8] Daniel Aarno, “Evaluation of real-time Linux derivatives for use in robotic control”, Available Online at: *ftp://ftp.realtimelinuxfoundation.org/pub/events/rtlws-1999/proc/p04-arduini.pdf.zip*

[9] IEEE. *IEEE standard for a high performance serial bus, Std 1394-1995 and amendments*, 2002.

[10] A. Speck, G. Gruhler, W. Kuchlin, “Object-oriented robot control framework”, *Industrial Electronics Society, 1998. IECON ’98. Proceed-ings of the 24th Annual Conference of the IEEE* , Vol. 3, pp.1663 -1666, 31 Aug.-4 Sept. 1998

[11] Y. Zhang, “Real-time network for distributed control”, MSc.-Report 031CE20055, Control Laboratory, University of Twente, the Nether-land , August 2005

[12] R. Ostrovrsnik, A. Hace, M. Terbuc, “Use of open source software for hard real-time experiments”, *Industrial Technology, 2003 IEEE International Conference on*, Vol. 2, pp.1243 -1246, 10-12 Dec. 2003

[13] DongYu, Hu Lin, Ruifeng Guo, Jiangang Yang, Pengfei Xiao, “Research on real-time middleware for open architecture controller”, *Embedded and Real-Time Computing Systems and Applications, 2005. Proceedings. 11th IEEE International Conference on* , pp. 80 -83 , 17-19 Aug. 2005

[14] S. Cavalieri, A. di Stefano, O. Mirabella, “Impact of field-bus on communication in robotic systems”, *Robotics and Automation, IEEE Transactions on* , Vol. 13, Issue 1, pp.30 -48 , Feb. 1997

[15] *http://www.fsmlabs.com*

[16] *http://www.rtai.org*

[17] *http://www.xenomai.org*

[18] *http://orca-robotics.sourceforge.net/*

[19] *http://www.cs.wustl.edu/schmidt/TAO-status.html*

[20] *http://www.realtimelinuxfoundation.org*

[21] *http://www.orocos.org*

[22] *http://rtfirewire.dynamized.com/*

[23] Object Management Group, “Real-time CORBA specification revision 1.1”, *OMG Document Formal/02-08-02*, August 2002

[24] *http://www.commell-sys.com/Product/SBC/LV-671.HTM*

[25] *http://www.sarotech.com/*

[26] *http://www.ubuntulinux.org/*

[27] http://www.ti.com/