Proceedings of the 2006 IEEE/RSJ
International Conference on Intelligent Robots and Systems
October 9 - 15, 2006, Beijing, China

An IEEE-1394 Based Real-time Robot Control System for Efficient Controlling of Humanoids

M. Omar Faruque Sarker, ChangHwan Kim, Seungheon Baek 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, shbaek, ybj}@kist.re.kr

***Abstract*— Modern sophisticated robots, e.g. humanoids, require significant real-time communication speed in their field-buses. Several existing field-bus technologies, like controller area network (CAN), Profibus etc., provide real-time performance and reliability, however the bandwidth is not enough to support the communication demand of the complex humanoids. In this research, we have addressed this critical issue by utilizing comparatively faster IEEE 1394 serial bus on commercial off-the-shelf hardware and Linux-based real-time software platforms. Since the lack of user-level communication protocols in IEEE 1394 standard is a major obstacle for designing a IEEE 1394 based control system, we have solved this issue by customizing the predefined IEEE 1394 packet formats. Moreover, various IEEE 1394 serial bus related features have been investigated for designing an efficient humanoid control system. In this work, we have followed the real-time modeling and analysis method using Unified Modeling Language (UML). A use case driven analysis of timing requirement and determination of real-time constrain have been shown to prove the usefulness of our approach. The ongoing implementation of this proposed system in our network based humanoid, MAHRU, has been demonstrated in both commercial RTOS RTLinux (Pro) and free OSS RTOS Xenomai. The performance measurement of basic IEEE 1394 read transaction has been shown in both platforms along with a comparison with similar experiment on CAN bus. This provides an estimation of performance improvement of a humanoid control system on IEEE 1394 serial bus over CAN bus. Moreover the results obtained from two different Linux-based RTOS platforms indicate their relative pros and cons. This also enables us to select the suitable real-time framework for designing an efficient IEEE 1394 based humanoid control system.**

***Index Terms*— Distributed control, IEEE 1394, Mobile robots, Motion control, Real time systems.**

I. INTRODUCTION

Rapid technological advances in microelectronics, computer and communication technology offer a great number of opportunities for the evolution of robot control technology. The development of high-speed single-chip digital signal processors (DSP) makes the implementation of software servo more easier for motor control in a complex robot [2]. But in order to develop any motor control application of complex humanoids, it is necessary to setup computer interface with higher real-time transmission rate and with high level motion and motor control protocols. Several existing field-bus technologies, like controller area network (CAN), Profibus, WorldFIP etc., provide real-time performance and reliability, however the bandwidth is not enough to support the communication demand of humanoids. On the other hand, comparatively higher bandwidth Ethernet is inherently not intended for real-time applications due to the lack of deterministic communication support. And like universal serial bus (USB), many new filed-bus technologies are not widely acceptable in robotics and its relevant industries [2], [3].

In such a scenario, due to the advantages of high speed, determinism, low-cost, robust and wide industrial support, we have introduced IEEE 1394 (also known as *Firewire*) [1], [4] as the solution for a faster field-bus in real-time robot control. Several motivations work behind this. The relatively higher bandwidth of IEEE 1394 (maximum 400 Mbps and 3.2 Gbps in 1394a and 1394b respectively) gives us enough flexibility to connect additional motor controllers or sensor nodes. Two operational modes: asynchronous (guaranteed data delivery) and isochronous (constant rate data transmission) are well suited for almost all control requirements of humanoid. Besides we have the provision for integrating IEEE 1394 video cameras of robot vision within a single IEEE 1394 network. Tree structure and automatic reconfiguration of this bus gives us the ability to design a modular and scalable humanoid. Moreover, the 8000Hz time division multiplexing of IEEE 1394 also ensures the worst case latencies below 250 μs which is comparatively lower in comparison with other field buses. However, there are many challenges to implement IEEE 1394 in a robot control system, e.g. absence of user-level communication protocols [20], suitable real-time operating system (RTOS) frameworks [3] etc. In this work, we have overcome some of these challenges with the following major contributions.

We have designed and developed the custom user level protocols to establish the necessary link between Linux Real-Time IEEE 1394 device driver [11] and motor-controller/sensor nodes in IEEE 1394 network. Here, various IEEE 1394 serial bus related features have been investigated, e.g. which types of IEEE 1394 transactions are appropriate in peer to peer/broadcast communication, how to minimize the delay in transactions etc. We have also followed the real-time modeling and analysis method using Unified Modeling Language (UML) [13]. A use case driven analysis of timing requirement and determination of real-time constrain have

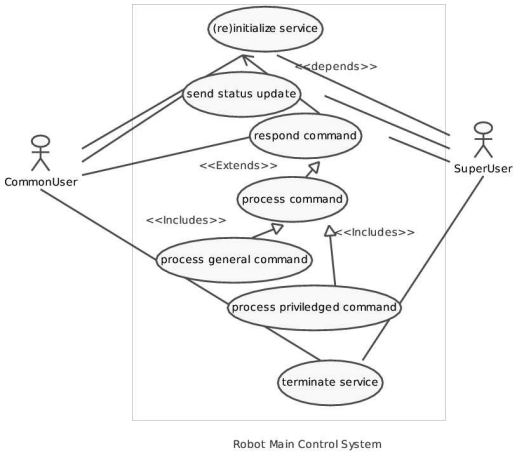

Fig. 1. Use case for external requirement capture

been shown to prove the usefulness of our approach. The ongoing implementation of this proposed robot control system in our network-based humanoid (NBH), MAHRU [6], has been demonstrated in both commercial RTOS RTLinux (Pro) [8] and free OSS RTOS Xenomai [7]. The performance measurement of basic IEEE 1394 read transaction has been shown in both platforms along with a comparison with similar experiment on CAN bus [16]. This provides an estimation of performance improvement of a humanoid control system on IEEE 1394 serial bus over CAN bus. And finally, the results obtained from two different Linux-based RTOS platforms also indicate their relative pros and cons which enables us to select the suitable real-time framework for designing an efficient IEEE 1394 based humanoid control system.

Since this work mainly focuses on the technical issues of the usage of IEEE 1394 rather than its theoretical aspects, the scope of this paper is limited to the discussion of IEEE 1394 based humanoid control system with a limited description of the distributed environment of our NBH.

Rest of this paper: Section 2 presents an overall modeling and analysis and Section 3 provides the design and development of our system. Section 4 describes the experiments. Section 5 briefly summarizes related works and finally Section 6 presents the summary and outlook.

II. Modeling and Analysis

The advantages of UML-based real-time modeling techniques are obvious in all stages of a product development cycle, e.g. requirement capturing, functional specification, design, development, verification and deployment. Based on UML methodology, at first we have constructed use-case diagrams for initial requirement capturing and role identification of the different active elements of our humanoid. The functional behaviors of main-controller have been modeled by the corresponding state diagram. Several scenarios have been constructed to identify the real-time requirement of our control applications. The consistency of the entire modeling process has been maintained by following some standard methods presented in [5], [10].

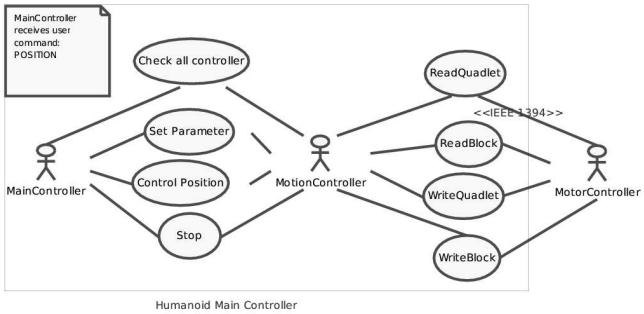

Fig. 2. Use case for internal role identification

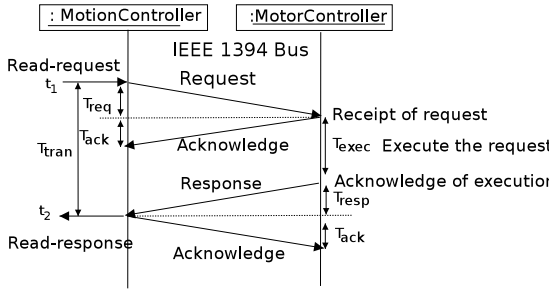

Fig. 3. IEEE 1394 read-transaction

A. Role and Requirement Identification

The use case, presented in Fig. 1, describes how two types of user, superuser and common user, use the humanoid main control system. Depending on their privileges, they can run different commands and services on the main controller over the network. The roles of the *MainController* and *MotionController* (software modules) are modeled in Fig. 2, considering the case when a user sends a position command to reach a new position. For successful execution of the position command, the MainController plays different higher-level functional roles, e.g. check all controllers, get parameter from the appropriate control algorithm process, control position, update status etc. which are shown in the sequence diagram in Fig. 4. The MotionController module assist MainController by performing lower-level functional roles. It uses IEEE 1394 real-time communication subsystem to send operational commands to the software modules of the motor-controller board or the sensor board. In Section 3, the details of this communication have been presented.

B. Timing Behavior Analysis

The timing behavior of MotionController has a significant impact on overall performance of MainController. While analyzing the timing behavior of the MotionController we have found that, asynchronous transaction, defined in IEEE 1394 standard [4], generally has the acknowledgment part in both request and response subaction. So, the communication latency between two nodes is slightly increased to support this fault-tolerant behavior. It is possible to avoid this time overhead if we use isochronous transaction or if we broadcast asynchronous messages. In case of broadcast, more than one node can be targeted and nodes will not return the

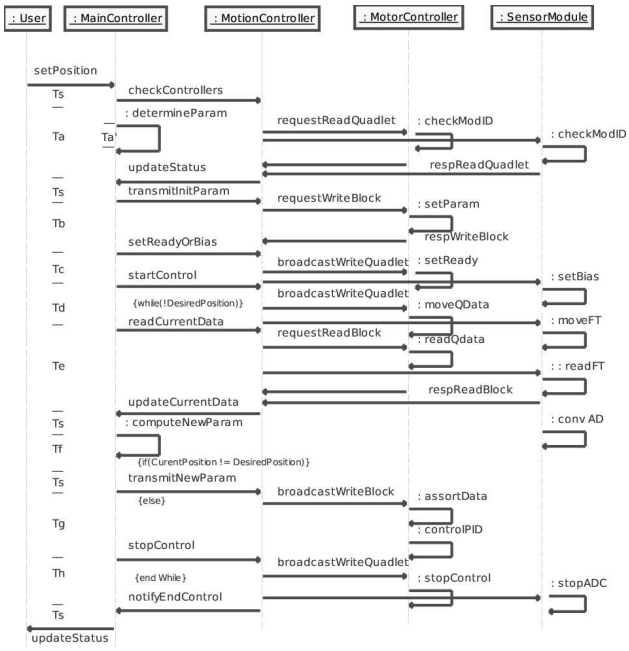

Fig. 4. Sequence diagram for a position command scenario

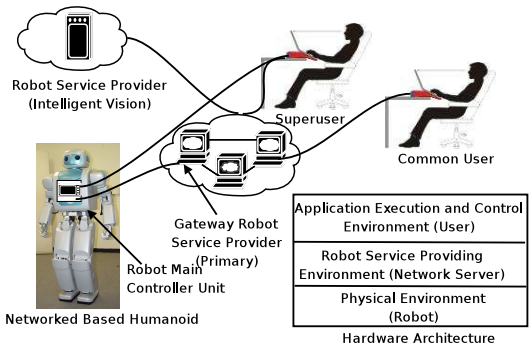

Fig. 5. System model and architecture (lower inset)

TABLE I

TIME-BUDGET OF MAINCONTROLLER FOR A CONTROL SCENARIO

Task Code	Task Name	Time-Budget Equation
T_a	checkControllers	$(N_m+N_s)\times t(PRQ)+2t(T_s)$
$T_{a'}$	determineParam	$t(T_{a'})$
T_b	transmitParam	$N_m\times t(PWB)+2t(T_s)$
T_c	setReadyOrBias	$t(BWQ)+t(T_s)$
T_d	startControl	$t(BWQ)+t(T_s)$
T_e	readCurrentData	$(N_m+N_s)\times t(PRB)+2t(T_s))$
T_f	computeNewParam	$t(T_f)$
T_g	transmitNewParam	$t(BWB)+t(T_s)$
T_h	stopControl	$t(BWQ)+t(T_s)$

acknowledgment packet. This can effectively be used to broadcast a write-transaction to send any predefined packet to a group of controller nodes. As shown in Fig. 3, the transaction time of an asynchronous operation between two nodes, e.g. read or write, consists of request time (T_{req}) response time (T_{resp}), acknowledgment time (T_{ack}) and the operation execution time (T_{exec}) of the target node. In case of read-request, the requester sends the address and transaction type to the responder and then after executing that transaction, the responder sends the read-data to the requester.

Here we have found that, the processing power of the responder has a significant impact on the required total transaction time. A powerful responder can respond the requested read-data very quickly. On the other hand, in case of write operation the powerful requester can send the write-data in a very short time.

The amount of requested data is also important. If the amount of data is a 1 byte to 4 bytes (called *quadlet* data) the required time is almost similar. But if the data is more than 4 bytes (termed as *block* data) the required transaction time increases according to the data pay load. Hence the control application should be designed with powerful local processors and minimum data payload for individual read operation. The various asynchronous read and write packet formats and their protocol overhead for different cases can be found at [1], [4].

Fig. 4 simultaneously shows the functional role of the MainController and MotionController in executing a position command from a user. We can analyze the timing behavior of the system from the outlined actions carried out by both software modules. At first, to estimate the time parameters of the MotionController, let us define 3 character codes for representing various types of transactions: the first character, P for *Peer to Peer (P2P)* or B for *Broadcast*, the middle character R for *Read* or W for *Write*, and the last character Q for *Quadlet* or B for *Block* (fixed-sized, say S byte data payload). So, we get $t(PRQ)$ and $t(PRB)$ as P2P read-transaction time for quadlet and block format respectively. Similarly, $t(PWQ)$ and $t(PWB)$ be the P2P write-transaction time in quadlet and block format, and $t(BWQ)$, $t(BWB)$ be the broadcast write-transaction time in quadlet and block format respectively.

Moreover, let N_m and N_s be the number of motor-controller and sensor module used in the system and $t(T_s)$ be the slack time between two consecutive events of MotionController or MainController (in same processor). Based on the above definitions, the time budget for the position control sequences of MainController, as shown in Fig. 4, is summarized in Table I. Using Table I, the real-time constrain of MainController can also be expressed mathematically. Suppose, if we want to operate our system at a period P unit time we must satisfy that the sum of the time required to read the current data from all nodes ($t(T_e)$), the new parameter computation time ($t(T_f)$), the new parameter transmission time ($t(T_g)$) and the twice of slack time ($2t(T_s)$) be less than the value of P. This is described in the following equation.

$$t(T_e)+t(T_f)+t(T_g)+2t(T_s)<P \tag{1}$$

As in a fast SBC processor $t(T_f)$ and $t(T_s)$ is very low, the dominating factor becomes the asynchronous read-transaction of MotionController, i.e. $t(T_e)$. $t(T_g)$ is also low due to the absence of *ack* process during message broadcast.

III. Design and Development

The design and development process of a distributed control system requires a concurrent design approach which can effectively integrate the interactions between real-time and nonreal-time requirement space and corresponding structure space [9]. Usually real-time control system is designed in three levels: architecture, mechanistic and detailed design [10]. At the initial stage of our work, a modular component-based architecture has been designed and following that design, the inter-component communication issues have been solved and the results have been implemented in the detail design of the components. The following sections describe our concurrent component-based design and development approach.

A. Architecture

The main role-players of our model, as shown in Fig. 5, are the *Network-based Humanoid*, *Superuser*, *Common user*, *Gateway to Robot Service Provider (GRSP)*, *Robot Service Provider (RSP)*. This model is consistent with our use case shown in Fig. 1. The lower inset shows our proposed hardware architecture with 3 layers.

B. Hardware Platform

The main-controller of our humanoid is designed using industrial Single Board Computer (SBC) of LV-671 series from Commell Systems Corporation [14] and the FPA-431 IEEE 1394 PCI card from SaroTech [15]. The IEEE 1394 PCI card contains the TSB43AB23 IEEE 1394 chip (TSB) from Texas Instruments (TI) [18]. To construct a motor-controller and sensor-module we have used TI TMS320VC33-150 Digital Signal Processor (DSP) which is simply termed as *VC33*. The configuration of our humanoid platform consists of 14 hardware modules, which include 10 motor-controllers (2 for each leg and 2 for each arm and 1 for each hand) and 4 sensor-modules (1 for each leg and 1 for each arm), and all of them are connected to SBC in a chained fashion starting from the 3 ports of the IEEE 1394 PCI card.

The development process of a motor-controller board, as shown in Fig. 6, includes lower level programming of DSP. The Code Composer IDE [18] is used to setup the DSP programs and then the programs are burnt into the ROM of VC33 by JTAG interface. The VC33 is interfaced with a FPGA (54-SX family) from Actel [19]. In case of a sensor-module, the H-bridge part of Fig. 6 is replaced by an Analogue to Digital Converter (ADC) and instead of using Motor/Encoder an appropriate Force/Torque (*FT*) sensor is used. The IEEE 1394 communication protocols and related software development process have been discussed in the following paragraphs.

C. IEEE 1394 Communication System

The various IEEE 1394 asynchronous packet formats, defined in [4], are modified to establish the communication link between the main-controller (SBC) and hardware modules (VC33). An example of modified packet formats is presented in Fig. 7. The modified fields are predefined and shown

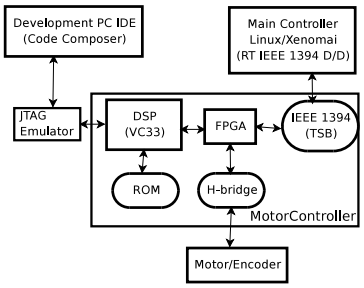

Fig. 6. Motor-controller developemt diagram

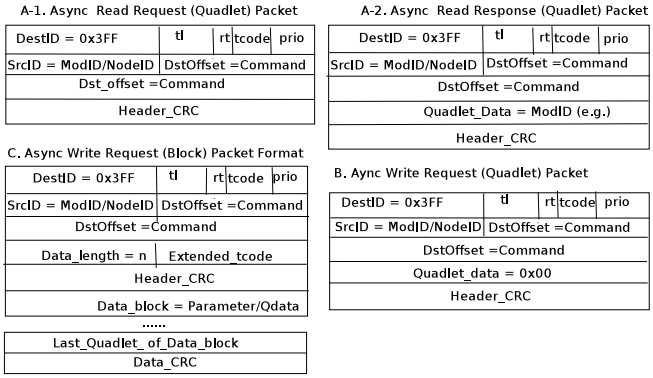

Fig. 7. Modifications of various IEEE 1394 packet formats

with an equal sign (=) with new parameter in the right side. The destination ID (DestID) is set to 0x3FF for all modules. As upon bus reset, all nodes get a new node ID which is inconvenient to maintain, so we have defined a module ID (ModID) for each node. The modified source ID (SrcID) can be either module ID (ModID) or node ID (NodeID), depending on the type of operation. In case of broadcast SBC uses SrcID as 0x3F (local bus ID). The most important modified field is the destination offset field which we have defined as command filed. A list of categorized commands is presented in Table II. As we have seen in Fig. 7, in case of checking module ID (ModID) of each module/node, SBC sends a quadlet read-request to the VC33 (shown in A.1) and then VC33 responds the read-request with a ModID in its Quadlet-data part of the read-response packet (shown in A.2). The basic commands, e.g. set ready, start or stop control etc., can be transmitted by broadcasting quadlet write-request (shown in B, Fig. 7). Similarly, we can use the block-read format if we need to read a block of data from sensor modules like force and torque, *FT data*, or motor joint position and velocity, *Qdata*.

TABLE II

IEEE 1394 CUSTOM *Command* LIST

Type of Operation	Command Name (Code)
P2P	*IDREQUEST(11)*, *IDCHECK(12)* *QREAD(17)*, *FTREAD(19)*
Broadcast	*STOP(00)*, *STRAT(16)* *READY(15)*, *SETBIAS(14)* *PARAMETER(13)*, *QWRITE(18)*

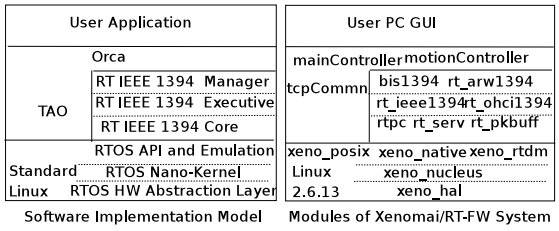

Fig. 8. Software layers and implementaion model under Xenoami

The block write-request (shown in C, Fig. 7) can be used for sending initial motor parameter (e.g. gear ratio, PID gain coefficients, etc. if PID controller is used) to each node or for broadcasting the new parameters (e.g. Qdata) to all nodes. In almost all cases, SBC plays the requester role whereas VC33 plays the responder role. The largest data block (n) sent by these responder nodes is 24 bytes (from sensor-modules) and the largest data block sent by the requester, SBC, is 320 bytes which contains the Qdata for 10 motor-controllers.

D. Software Platform

In last few years, Linux-based RTOS RTLinux and Real-Time Application Interface (RTAI) have been successfully implemented in many systems around the globe. We have also initially focused on the commercial RTOS, RTLinux (Pro) based system where the real-time IEEE 1394 device driver comes under LNet package. As RTLinux is highly expensive for mass implementation and it does not provide the operating system source code, we have ported our application to Xenomai (former fusion branch of RTAI, RTAI/fusion) with real-time IEEE 1394 device driver from Real-Time FireWire (RT-FW) project [11]. Our implementation model with the component modules are shown in Fig. 8 and they are briefly explained here.

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 [7], [8]. Adaptive Domain for Operating Systems (ADEOS) is such an approach. Applying ADEOS new generation i-pipe patch to the standard Linux 2.6.13 kernel on Debian based Ubuntu 5.10 [17] we have installed Xenomai RTOS which has the layered sub-modules as shown in the Fig. 8.

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.

3) Main Controller Application Components: The main-controller has two important application packages: *mainController* and *motionController*. mainController package has a subcomponent: controlAlgorithm (for computing the control parameters). The motionController package consists of *rt_ieee1394_ServiceManager* which interfaces with mainController and *rt_ieee1394_DriverHandler* that communicates with real-time IEEE 1394 device driver via rt_arw1394.

TABLE III
WORST CASE EXECUTION TIME (WCET) FOR THE READ-TRANSACTION IN XENOMAI AND RTLINUX

Data Payload (Byte)	Xenomai WCET (μs)	RTLinux Pro WCET) (μs)
4	61	58
32	75	70
64	90	82
128	118	108
256	172	166
496	277	272

IV. EXPERIMENT AND EVALUATION

A. Xenomai and RTLinux Based Implementation

Real-time performance of our system is evaluated on commercial RTLinux (Pro) and OSS Xenomai platform (in a same set of hardware). Here we have decided to operate our system at a period (P) of 2 ms. So, the condition of (1), as described in Section 2, must be met where the dominating parameter is $t(T_e)$ and it is depended on the value of $t(PRB)$. For evaluating $t(PRB)$, a series of asynchronous read experiments have been done and from more than 1 million sample data the Worst Case Execution Time (WCET) for different data payloads is listed in Table III.

In case of RTLinux/LNet, we have developed our custom program to record time at t_1 and t_2 point of a read-transaction, as shown in Fig. 3. The difference (T_{tran}) is the required read-transaction time. In case of Xenomai/RT-FW, the rtping application kindly provided with the RT-FW package has been used. In the ping packet format [1] a *type* field value of *0000b* designates the extended physical packet as a ping packet. In response, the target node broadcasts its self-packet and the requester can measure the delay between sending the ping packet and the return of the self-ID packet, as shown in Fig. 3.

In our system-design, we have used maximum read-request data payload 24 bytes for each sensor-module and 32 bytes for each motor-controller. So now, we can calculate the required $t(T_e)$ using the time budget described in Table I and the data provided in Table III, i.e. $t(PRB)$ for highest data payload of 32 bytes. For all 14 hardware modules as explained earlier, the value of $t(T_e)$ for RTLinux is $(14 \times 70) = 980\mu$s and that for Xenomai is $(14 \times 75) = 1050\mu$s and in RTLinux (Pro), the worst case value of $t(T_g)$ is recorded as 40 μs and in Xenomai it is assumed as 50 μs (recall that $t(BWB)$ has no acknowledgment process, so $t(BWB)$ will become lower than $t(PRB)$). Now, if we add the largest SBC processing time for $t(T_f) = 500\mu$s and $t(T_s) = 20\mu$s respectively, then the sum of all four quantities become 1.56 ms in RTLinux and 1.64 ms in Xenomai which is less than the target P value of 2 ms.

From our experience in both RTOS platforms, we have seen that, the additional overhead created in Xenomai based system is due to the context-switch time between the user-space and kernel-space. However it is preferable to use user-space programs which is more flexible as they can use native programming libraries. Since RTLinux/LNet uses

TABLE IV
TRANSACTION TIME OF CAN AND IEEE 1394 IN HUMANOID CONTROL

Implementation	Send/Request Time (μs)	Recv./Resp. Time (μs)	Total Time (μs)
Linux/CAN	4000	6500	10500
RTLinux/IEEE 1394	27	31	58
Xenomai/IEEE 1394	29	32	61

kernel-space, its execution time is slightly lower. But here, program debugging is more complex and overall capabilities are limited by the higher implementation cost and proprietary APIs. On the other hand, in case of OSS Xenomai, a lot of programming libraries and debugging tools are available in user-space for free. The main weakness of Xenomai based system is that it is loosely integrated due to the diverse open source development approach. In our case, although the absence of real-time driver module for raw read-write access of IEEE 1394 bus is delaying us to perform more experiments on current Xenomai/RT-FireWire framework, a custom IEEE 1394 driver module to serve this purpose is underway.

B. Comparative Analysis with CAN-Based System

The experimental results of message scheduling for a humanoid in CAN, presented by [16], is used here for comparing the performance between IEEE 1394 and CAN. The data of the above mentioned CAN system, shown in Table IV were captured at 20 ms period and CAN-bandwidth 1 Mbits/s, where data payload was 8 bytes. From Table IV we can compare the relative performance increase of both RTLinux and Xenomai based systems over CAN based system. The RTLinux/LNnet system gives roughly about 181 times and Xenomai/RT-FW gives about 172 times lower latency in comparison with the CAN based system.

V. RELATED WORK

The limitations of traditional centralized control systems led some pioneer researchers to evaluate the impact of field-bus in robotic systems [2], [21]. Comparative study on different types of filed-bus in distributed process control can be found at [2]. Many researchers implemented the CAN based systems in comparatively simpler robot platform. Although we have found only a few IEEE 1394 based simple control system, e.g. plant control used in [3], still IEEE 1394 is not very common in complex robot control. [3] also presented the coupling of IEEE 1394 with the real-time networking framework for Linux, RTnet [12], in the form of IP over Firewire. But this work is in infant stage from the view point of its practical application. Presently in robotics, the isochronous mode of IEEE 1394 is widely used for developing robot vision system. Although based on proprietary RTOS (e.g. QNX) and IEEE 1394 communication protocol some systems were developed for real-time control of robot manipulator, e.g. parallel robot control architecture developed by [20], a large number of the benefits offered by IEEE 1394 serial bus have not been properly utilized yet.

VI. SUMMARY AND OUTLOOK

In this paper, we have presented the development process of our IEEE 1394 based real-time humanoid control system. Since the lack of user-level communication protocols in IEEE 1394 standard is a main obstacle for developing such a system, we have overcome this by customizing the predefined IEEE 1394 packet formats. In our work, we have followed the model-based system development approach using UML. Our open and modular system architecture enables us to implement our system in commercial RTOS RTLinux (Pro) and OSS RTOS Xenomai. The experimental results shows that both platforms have almost similar hard real-time performance in IEEE 1394 bus, although OSS Xenomai has the higher advantages regarding the flexibility in application development and cost for implementation. The performance of our system, in comparison with a CAN-based system, also demonstrates the future prospects of IEEE 1394 in humanoid control applications.

Our current work is mainly focused on the development of an OSS Xenomai based complete humanoid control system. The major challenges are to standardize the Real-Time IEEE 1394 device driver with necessary driver modules and user libraries for developing distributed robot control applications in the near future.

REFERENCES

[1] Anderson, D., *FireWire System Architecture*, Addison-Wesley; 2003.
[2] Lin, S.-Y, Ho, C.-Y., Tzo, Y.-Y, Distributed motion control using real-time network communication techniques, *Power Electronics and Motion Control Conference, 2000. Proceedings. PIEMC 2000. The Third International*, Volume 2, 15-18 Aug. 2000 Page(s):843 - 847.
[3] Zhang, Y., Real Time Network for Distributed Control, MSc.-Report 031CE20055, Control Laboratory, University of Twente, the Netherland , August 2005.
[4] IEEE. *IEEE standard for a high performance serial bus, Std 1394-1995 and amendments*, 2002.
[5] Kuster, J., Stroop, J., Consistent design of embedded real-time systems with UML-RT, *Object-Oriented Real-Time Distributed Computing, 2001. ISORC - 2001. Proceedings. Fourth IEEE International Symposium on*, 2-4 May 2001 Page(s):31 - 40.
[6] *http://humanoid.kist.re.kr*
[7] *http://www.xenomai.org*
[8] *http://www.fsmlabs.com*
[9] Li, Q., Zhang, W.J., Chen, L., Design for Control - A Concurrent Engineering, *Mechatronics, IEEE/ASME Transactions on*, Volume 6, Issue 2, June 2001 Page(s):161 - 169.
[10] Douglass, B., P., *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns*, Addison-Wesley; 1999.
[11] *http://rtfirewire.dynamized.com/*
[12] *http://www.rtnet.org/*
[13] *http://www.omg.org*
[14] *http://www.commell-sys.com/Product/SBC/LV-671.HTM*
[15] *http://www.sarotech.com/*
[16] Sung, Y.-K., et al, Messages scheduling for a humanoid robot in the CAN, *Industrial Electronics Society, 2004. IECON 2004. 30th Annual Conference of IEEE* Volume 3, 2-6 Nov. 2004 Page(s):2470 - 247.4.
[17] *http://www.ubuntulinux.org/*
[18] *http://www.ti.com/*
[19] *http://www.actel.com/*
[20] Kohn, N., et al, PROSA-A Generic Control Architecture for Parallel Robots, [Online] Available at: *http://www.cs.tu-bs.de/ips/diethers/Dokumente/diethers-MeRo2004.pdf*
[21] Cavalieri, S., et al, Impact of Fieldbus on Communication in Robotic Systems, *Robotics and Automation, IEEE Transactions on* , Volume 13, Issue 1, Feb. 1997 Page(s):30 - 48.