Virtual BUS: An easy-to-use environment for distributed resources

Atsushi Takahara Sei-Ichiro Tani Shinya Ishihara Toshiaki Miyazaki
Mitsuo Teramoto Tomoo Fukazawa
Kazuyoshi Matsuhiro
NTT Network Innovation Laboratories
Hikari-no-oka 1-1, Yokosuka, Japan, 239-0847
{taka,tanizo,shinya,miyazaki,tera,fuka,mat}@exa.onlab.ntt.co.jp

Abstract

This paper discusses how a distributed environment providing effortless networking can be efficiently implemented in a network system. To formalize the distributed environment, we introduce the concept of "Virtual BUS". It is similar to the concept of the computer system's bus architecture. We define user behavior as accessing the resources in a network through his/her own bus. Based on this simple formalization, we discuss the key issues in implementing distributed environments to support different requirements for realizing the quality of service desired. We implement an experimental effortless networking environment based on the Virtual BUS concept and show that the concept realizes effortless networking while guaranteeing QoS.

1. Introduction

High throughput and high speed networking technologies have made the distributed environment more realistic. *Effortless Networking*[1] is a trend based on these technologies. Effortless networking provides a network environment wherein users can easily access resources in a network without tedious setting up operations like installing device drivers. In other words, a *real* plug and play network environment can be provided.

Several activities have been directed towards developing such distributed environments (TINA[8], Jini[7]). TINA provides a software framework for supporting open multi-provider and open multi-vendor environments. TINA is implemented on the Distributed Processing Environment (DPE) which is an extension of OMG CORBA. DPE functions have been implemented on different platforms. Communications among objects in DPE or platforms rely on signaling mechanisms supported by the Kernel Transport Network (KTN). Jini has the same structure as TINA. Jini is based on Java while TINA is based on CORBA. Jini uses Java's Remote Method Invocation (RMI) mechanism for communication between users and providers/vendors. A special function, the so-called *lookup service*, is provided to locate the provider(s) that meet the user's requirements [9]. Frameworks for application-specific distributed environments have also been proposed[2, 3].

These technologies were designed with the goal of realizing uniform distributed computation environments. Moreover, they concentrate on the functionalities that simplify usage or development processes. Unfortunately, there has been little discussion on performance or application-specific requirements in general distributed environments.

In effortless networking, various quality of service (QoS) levels must be guaranteed to increase working efficiency. Requests from applications are orthogonal in a network layer. Currently, network nodes should refer the upper layer information to achieve effective traffic flow control. Once the communicate process between applications and network layers is settled, the network resources can be used effectively and QoS can also be guaranteed by referring to the application layer requirements. Unfortunately, no well defined communication methods exist to link applications and the network layer.

This paper discusses how the distributed environment can be well supported by the network layer. The key point to realize this is the method of notifying the usage or requirements of the applications to the network layer control processes and vice versa. We define a simple model for representing the behavior of applications in a distributed environment;, it is called *Virtual BUS*. In Virtual BUS, the organization of distributed objects is modeled as the bus architecture of the conventional computer system. The interactions among distributed objects are modeled as transactions through the bus. We also introduce a weak relationship between a user and a resource. In Virtual BUS, a user accesses a resource through proxy agents. This weak relationship gives freedom to the network layer control. If different locations in the network offer the same resource, the most suitable location is determined according to network traffic status with the goal of more effective QoS control.

The rest of this paper is organized as follows. Section 2 defines the Virtual BUS concept model. In Section 3, we show several QoS guarantee methods that achieve effective network control meeting the application's requirements. Section 4 describes our experimental implementation of effortless networking based on the Virtual BUS concept and several applications. Section 5 concludes the paper.

2. A formalization of the distributed environment

2.1. Service-in-the-box

We assume that effortless networking is a sort of box in which various services are stored. We call this view the *service-in-the-box*(Fig. 1). A user picks his/her favorite services from this box and constructs his/her own computation environment. In the current networking environment, picking up a service is not an easy task because you must 1)locate the service using the network, 2) install appropriate device drivers and 3)manage network throughput or latency.

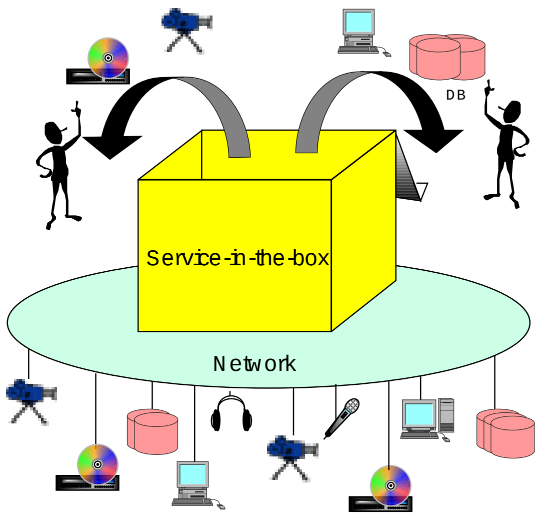

Figure 1. Service-in-the-box

In the service-in-the-box concept, a user can access a service without searching through many directories. Uniform access methods for services are supported to make it easier to use the services. The network layer guarantees the QoS level needed by observing the traffic pattern or service usage. Three factors are needed to realize the concept of service-in-the-box: virtuality, uniformity, and cooperation between network systems and services.

Virtuality To make it easier for the user, virtual service access is needed. A user can access a resource by specifying a virtual name instead of the exact service name or location. For example, the user wants to compress some video data. The user does not want to be forced to specify the exact location of the compressor nor be forced to look up the sites offering such devices. The user wants these things to be hidden. The user merely wants to say " compress this data".

This is effective for supporting QoS. The relationship between the user's request and an exact service is weak so that there is freedom in choosing the location of a service. An application in Virtual BUS can choose the best location for each service considering the network's traffic status.

Uniformity The method of accessing services should be uniform to make the system easy to use. Otherwise, accessing a different implementation of a service would require an implementation specific interface.

Cooperation between Network and Services Different types of services require different types of QoS. Many QoS control mechanisms have been proposed for the network layer. If the requirements of users are known, the appropriate QoS control mechanism can be selected. Moreover, resource reservation and congestion control mechanisms have the freedom to choose the communication path used because the locations of services are not precisely specified initially. By considering the status of network traffic, the network layer can choose the best service location. This significantly improves the possibility of network layer traffic control achieving the desired QoS.

2.2. Virtual BUS

We introduce a simple model that can represent the distributed environment to allow discussion of the required functionality and implementation issues. To define the model, we consider the user's behavior. When the user requires a service, he/she should prepare their computation environment. For instance, when a user wants to use a video on demand (VOD) service, he/she must prepare a decoder. Some negotiations are conducted between the encoder and decoder before the video stream can be sent. Forming the distributed computation environment is very similar to constructing a personal computer. One buys a video encoder/decoder board, a CPU, a display and so on then attaches them to the same PCI bus. From this observation, we introduce a bus model for representing the user's behavior in the distributed environment. We call this model *Virtual BUS* (Fig. 2).

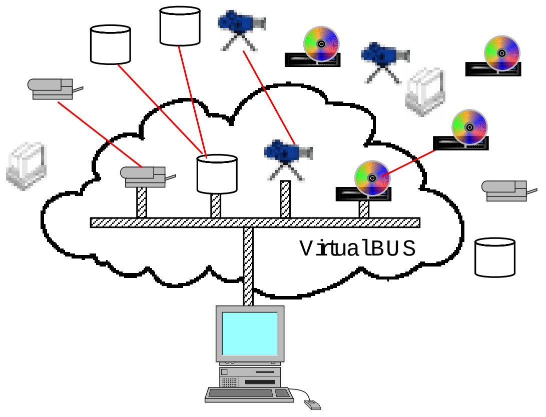

Figure 2. Virtual BUS

The user constructs his/her own virtual bus in a network and then attaches the resources available in the network as needed. Virtual BUS is similar to the usual computer's bus architecture, such as PCI, because a distributed computation environment can be constructed attaching/detaching resources to/from a bus. The difference between Virtual BUS and computer's bus is the lifetime of the environment so constructed. A virtual BUS is created/destroyed whenever a user accesses/releases resources while a computer bus has a much longer existence.

2.3. Primitives of Virtual BUS

The virtual BUS consists of four primitive modules.

1. Virtual BUS Application Interface (VBA)
2. Virtual BUS Resource Interface (VBRDV)
3. Virtual BUS Resource Database (VBDA)
4. Virtual BUS daemon (VBUSD)

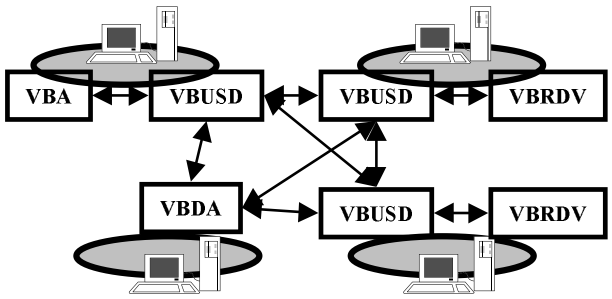

Figure 3. Configuration of Virtual BUS primitive modules

The structure of these modules is depicted in Fig. 3.

VBA is the library of functions that allow access to Virtual BUS; they are used by the applications. VBA offers the following functions.

OPEN/CLOSE Function OPEN creates a Virtual BUS in the network. Function CLOSE destroys a Virtual BUS in the network.

GET/RELEASE Function GET searches for a resource in the network and attaches it to the current Virtual BUS when found. To specify a resource in function GET, only the virtual name of the resource required can be used. Function GET searches for an actual resource by referring to the resource database in VBDA and attaches the resource selected to the current Virtual BUS. This mechanism realizes "virtuality" and is completely hidden from the user. Function RELEASE detaches a resource from the current Virtual BUS.

WRITE/READ A uniform interface is required to access resources. In the Virtual BUS environment, a resource is accessed using memory-mapped I/O style access functions. The tuple of address and data is supplied when accessing resources.

VBRDV is the interface between the Virtual BUS and each resource. An actual resource is encapsulated by VBRDV. VBRDV processes requests from VBA and controls the resource using its encapsulated driver functions. VBRDV works as described below.

- When a resource starts providing a service, VBRDV registers its own resource information to the resource database in VBDA. This information enables an application to access the resources.
- When VBRDV receives GET request, VBRDV checks whether resources of the right type can be

used by the user or not. VBRDV returns an acknowledge message if one is available and VBRDV registers the fact that the resource is being used in the resource database in VBDA. If no resources are available, VBRDV rejects the request GET.

- When VBRDV receives requests of WRITE/READ, the requests are analyzed and VBRDV executes the appropriate resource driver functions.

- When a service is terminated, VBRDV deregisters the resource information from the resource database in VBDA.

VBDA is the resource database manager in Virtual BUS and lists the currently available resources. For each resource, the key data and its status are specified.

VBUSD manages communications among applications and resources. VBUSD provides proxies for VBA and VBRDV. Even if VBUSD switches from one resource to another, VBA can continue operation. The proxies provided by VBUSD achieve virtuality and free the user-resource relationship.

2.4. Protocol

A simple example is used to explain how primitive modules work together in realizing the Virtual BUS environment. We assume that the user wants to watch " Movie#1" which is stored as an MPEG2 encoded video stream. A typical access sequence is depicted in Fig. 4.

1. Both an MPEG2 encoder and a decoder are needed in this situation. When these resources start their service, they register their information in the VBDA resource database.

2. A user starts the application "VOD" . VOD opens its own Virtual BUS and tries to attach a decoder and an encoder.

3. VOD attaches a decoder by using request GET of VBA. This request is sent to VBUSD. VBUSD looks up in VBDA resource database and locates an appropriate decoder with specified attributes such as encoding format. In this example, the decoder and VOD are managed by the same VBUSD so request GET is sent to the decoder resource directly. If the decoder is available, it sends acknowledge to VOD and registers the fact that the resource is being used in the VBDA database. Otherwise, the decoder notifies that the resource is not available.

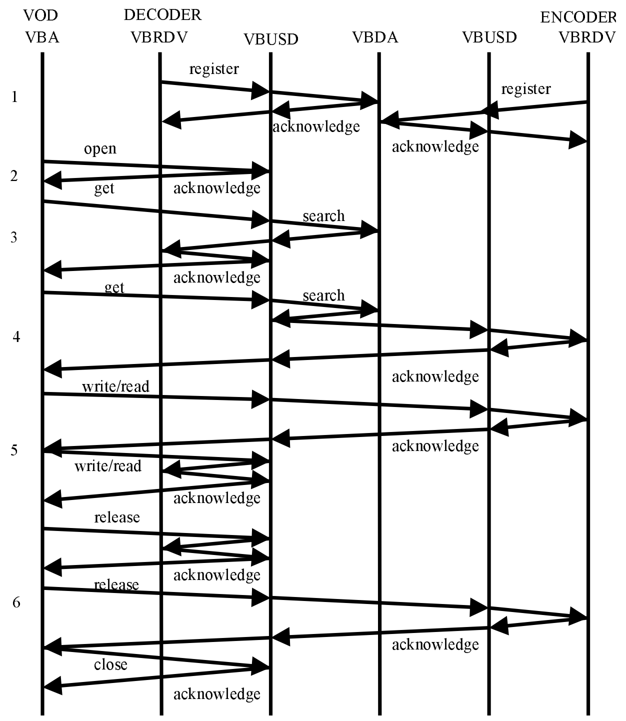

Figure 4. A typical sequence of accessing resources

4. VOD then tries to get an encoder. VOD sends request GET to VBUSD. This request may be relayed from one VBUSD to another. Except for this sequence, the other sequences are as same as those used to attach the decoder.

5. After attaching resources, VOD can locate and communicate with the resources selected. Control commands or video streams are accessed by WRITE/READ requests. These requests are also relayed through the VBUSD chain.

6. When an application finishes, attached resources are released from the Virtual BUS. The application sends a request RELEASE to each attached resource through VBUSD. A resource that receives request RELEASE registers the fact that it has become available to the VBDA resource database. Finally, VOD closes its Virtual BUS.

3. QoS guarantee methods

This section discusses the advanced features of Virtual BUS that support different levels of QoS guarantee capability.

3.1. Naming policy and resource search

To support virtuality, the naming policy of resources is important; the user wants to use only simple terms to get the appropriate devices. Let's use the previous example again. How do you specify "Movie#1"? You need to find an encoder and a decoder. What kind of encoder/decoder do you need? What kind of formats do you need? Possibilities include "encoder for MOVIE#1" and "MPEG2 format encoder for MOVIE#1". The former has more freedom because no coding format is specified.

The resource information is categorized into two forms.

1. Resource type

   This gives basic resource information. Typically, type names are words which the user can easily guess, such as memory, CPU, disk, encoder, decoder and so on.

2. Resource attribute

   This gives the resource specific information. Typical attributes are encoding format, video title, traffic rate, location and so on.

To guarantee QoS, resource performance and status should be also registered in the resource database. This information is useful for traffic control. When a user requests a VOD service, the application chooses the most suitable resource by referring to the database. Notice that these operations are invisible to the user. The available resources are registered in the VBDA database. The application can browse the list of up-to-date resources without instigating a search for all resources available in the entire network. Of course, the status of the network is changing frequently so the database is not so accurate.

3.2. Composite resources

In Virtual BUS, the combination of resources can be modeled easily by the joint action of VBRDV and VBA as depicted in Fig. 5. A composite resource has the same structure as a resource except that a VBRDV creates another Virtual BUS to attach other resources. When an application accesses a composite resource, it sends requests to the composite resource, which then relays these requests to the attached resources.

The creation of composite resources can provide virtually unbounded resources such as composite memory.

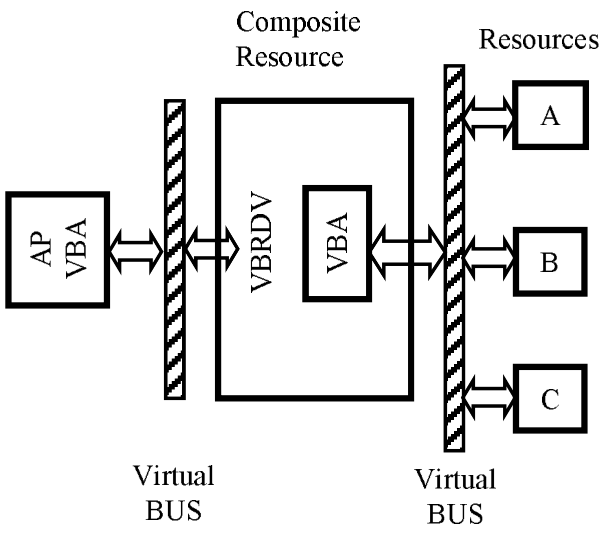

Figure 5. The structure of composite resource

Using this type of memory, the user does not need to worry about the amount of memory available. Composite memory grows according to the user demand by attaching new memory resources to its Virtual BUS. This capability guarantees another attribute of QoS.

Composite resources can also realize redundancy. A composite disk resource can offer the mirroring of data for fault tolerance and multi striping of disks to accelerate the rate of data access as in the RAID technology. The quality of VOD service can also be improved by using a composite resource. A composite VOD server links video sources in different locations. It selects the source whose connection to the application can fully support the traffic rate needed. If the current server fails to maintain adequate quality, for instance it starts to drop frames, the composite VOD server then switches to another source. This realizes dynamic resource roaming and guarantees QoS to the user.

A composite resource can thus provide QoS guarantees in terms of the function provided.

3.3. Cooperation with network node systems

The above two features reflect the capabilities of the application and middle-ware. Network routing/switching systems have, of course, very basic functions for guaranteeing the QoS such as rate control and signaling.

A composite VOD resource can effectively work with the network nodes. Two different VOD servers (A, B) are joined to form a composite VOD server that communicates through switching nodes (Node A, Node B) in the network as depicted in Fig. 6. If the traffic load

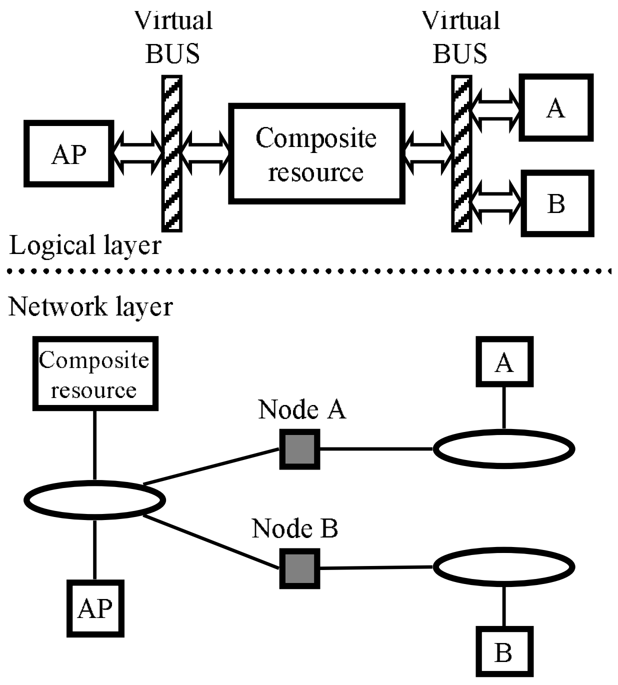

Figure 6. The network nodes communicating with a composite resource

of node A is high and the required video stream traffic rate can not be guaranteed, the composite resource switches from resource A to resource B to guarantee the video stream traffic rate. In the logical layer, the application communicates with the composite resource and so is not affected if a resource is switched.

4. Experimental system

We developed an experimental distributed environment based on the proposed Virtual BUS concept. In this section, we introduce our implementation method, the experimental system structure, and the applications developed so far.

4.1. Implementation

PVM (Parallel Virtual Machine)[5] and Java[6] are used in our implementation platform. These two software tools run on different platforms and have sophisticated communication mechanisms. These features are useful in implementing the distributed environment.

VBA and VBRDV are implemented as $C++$ library calls using PVM communication functions. VBUSD is implemented as a daemon written in $C++$ using PVM communication functions. VBDA, the resource database manager, is written in Java. VBUSD looks up the database through VBDAC which is the bridge between the PVM communication and Remote Method Invocation (RMI) functions of Java. The resource information in the current VBDA version consists of four items.

- resource type
- resource attributes
- expiration time for the resource information
- network wide unique resource identifier

The current implementation provides one VBDA in the network and VBUSD runs on each machine hosting resources and applications.

4.2. Network architecture

The network architecture of our experimental system is depicted in Fig. 7. The specifications of platforms, operating systems, network nodes, and supported resources are summarized in Table 1.

Table 1. Specifications of network nodes and machines in the experimental system

Platforms	PC (Pentium II 400MHz) Sparc Station (Ultra Sparc 300MHz)
Operating systems	Windows NT 4.0 Solaris 2.6 FreeBSD 2.2.8
Network nodes	ATM (Fore 200BX) 100Base-T 16 port Hubs ATTRACTOR (Programmable telecommunication nodes)
Resources	MPEG2 Encoder/Decoder Motion JPEG Encoder/Decoder Video camera with pedestal control serial line controllable robot

There are three ATM switches for handling video stream data and 100Base-T Ethernet connections to process Virtual BUS control protocols. We also used two ATTRACTOR[4] nodes, which are programmable

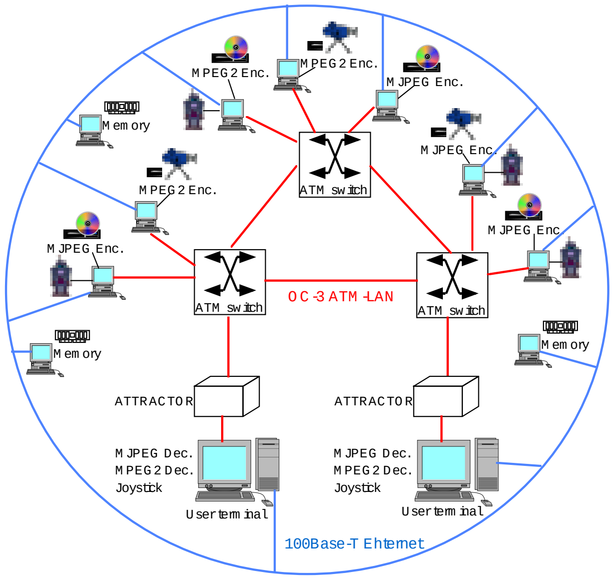

Figure 7. The network architecture of the experimental system

telecommunication nodes. ATTRACTOR can select the best ATM PVC path between an application and a resource according to the traffic situation in the network. Actual resources such as memories, encoders/decoders for MPEG2 and Motion JPEG (MJPEG), camera controllers, and robot controllers are hosted by 12 PCs and workstations.

4.3. Applications

We have implemented several applications, three of which are described below.

Composite memory Composite memory allows memory resources spread over the network to be combined and accessed by the user.

Figure 8 shows the performance of composite memory. In this experiment, we constructed 64, 96, 128M byte composite memories by combining 32M byte memory resources. The Y-axis shows the elapsed time for writing/reading data to/from a composite memory, access was by 8K byte block/16K byte block units, and the X-axis shows the size of memory. In the current implementation the access time for a composite memory is almost twice that of real memories.

VOD services Several VOD services were implemented. The current environment supports MPEG2 and Motion JPEG encoders/decoders. A flexible VOD watcher service was realized. If the

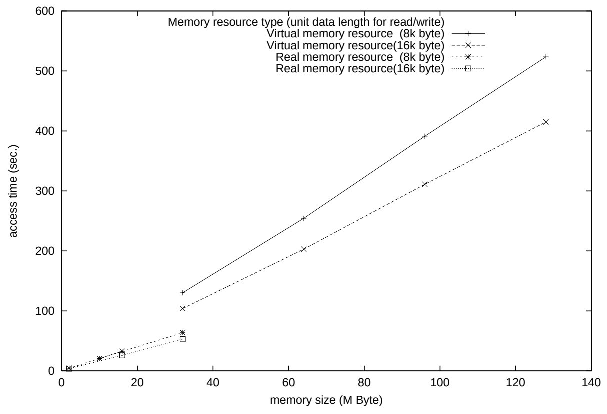

Figure 8. A composite memory performance

program desired is available in both MPEG2 and Motion JPEG encoding formats, the VOD watcher selects the appropriate encoding format according to the traffic situation.

Another service is the composite VOD server with network node controls. The composite VOD server controls VOD services in different locations. It uses ATTRACTOR to link the encoder to the decoder by PVC selection according to the traffic. The path from the encoder to the decoder is switched to guarantee the required traffic rate. ATTRACTOR can switch form one PVC to another within 1 milliseconds for 50Mbps Motion JPEG flows.

Robot game This application utilizes several resources. They include remote robot control, the camera which tracks robot movement and the MPEG2 encoder/decoder used to transfer the camera view to the user. The user controls one robot and one camera at the same time; the objective is to push the ball into other user's goal(Fig. 9). A user can join the game freely from any place.

In this game, a robot/camera pair is occupied by one user at a time. Virtual BUS functions can support this exclusive control of camera/robot while a dedicated game server would be needed to realize this normally. This game demonstrates how virtual BUS functions effectively support implementing network-wide interactive applications.

In the current system, VBA is the master of the Virtual BUS created. In other words, VBA takes care of all resources. However, it is desirable for the encoder and decoder to transfer video streams autonomously. Thus auto negotiation mechanisms among resources would be helpful in developing applications. For large scale networks, we should consider a hierarchical resource database for efficient database search and maintenance. The protocol of the implemented systems should have more robustness by offering tolerance to network malfunctions. These are future tasks.

5. Conclusions

We proposed the concept of Virtual BUS; a distributed environment model that aims to realize effortless networking environment. The essential points of distributed environment, virtuality, uniformity, and cooperations between applications and network nodes are clearly modeled in Virtual BUS concept. The required QoS for the effortless network is also categorized and the guarantee of these QoS is well supported by sophisticated resource search, composite resources in Virtual BUS concept. We implemented an experimental system based on the Virtual BUS concept and showed that Virtual BUS concept realizes effortless networking with the capability of guaranteeing the QoS.

References

[1] J. Callahan. Moving Toward Effortless Networking. *IEEE Computer Magazine*, pages 12–14, November 1998.

[2] H. Guyennet, J.-C. Lapayre, and M. Tréhel. Distributed shared memory layer for cooperative work applications. In *The 22nd Annual Conference on Local Computer Networks(LCN'97)*, pages 326–334, 1997.

[3] C. S. Hong, Y. K. Dai Kashiwa, and Y. Matsushita. A multimedia service networking architecture and its applications in tina-like model. In *The 21st Annual Conference on Local Computer Networks(LCN'96)*, pages 326–334, 1996.

[4] T. Miyazaki, K. Shirakawa, M. Katayama, T. Murooka, and A. Takahara. A transmutable telecom system. *Proc. 8th International Workshop on Field-Programmable Logic and Applications, FPL '98 (LNCS 1482)*, pages 366–375, 1998.

[5] Oak Ridge National Laboratory. PVM: Parallel Virtual Machine. http://www.epm.ornl.gov/pvm/pvm_home.html.

[6] SUN Microsystems. Java(tm) Technology Home Page. http://java.sun.com/.

[7] SUN Microsystems. *Jini(tm) Connection Technology, http://www.sun.com/jini.*

[8] TINA Consortium. *TINA-C Home Page, http://www.tinac.org/.*

[9] J. Waldo. *Jini^{TM} Architecture Overview*, 1998.

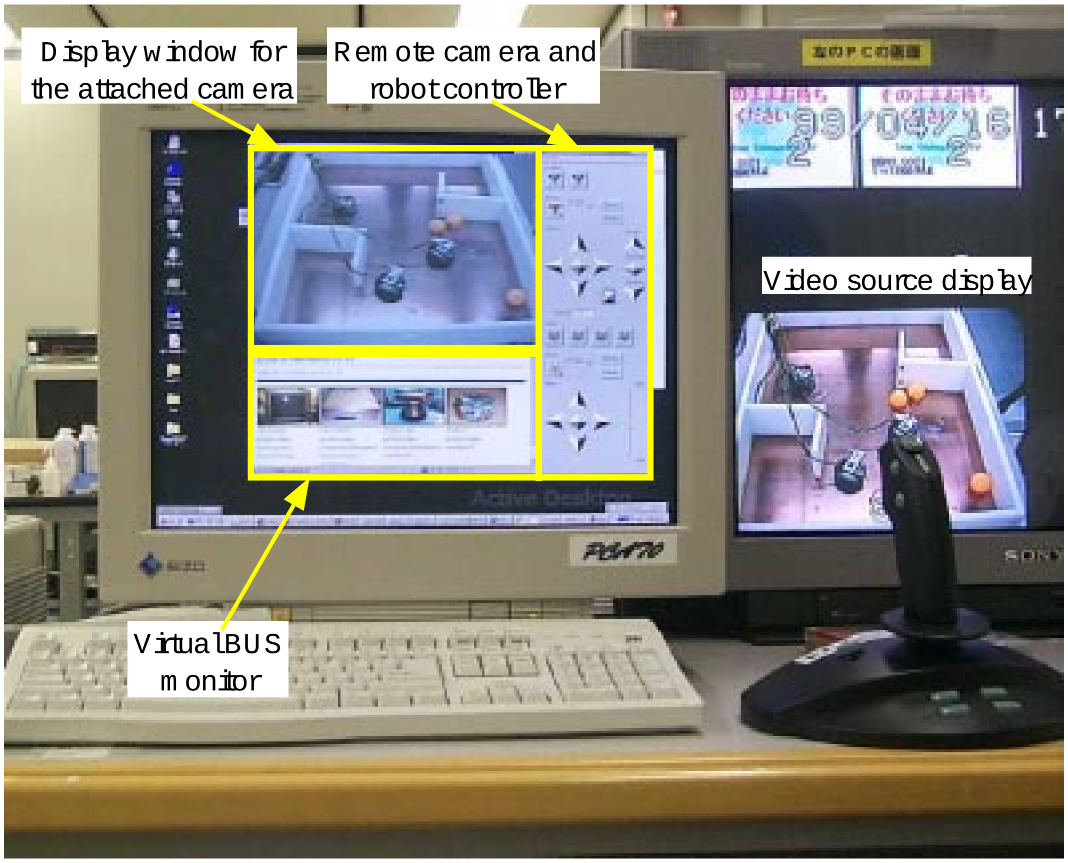
Display window for the attached camera
Remote camera and robot controller
Video source display
VirtualBUS monitor

Figure 9. Robot game: an example of Virtual BUS based application