Virtual BUS: A Network Technology for Setting up Distributed Resources in Your Own Computer

Toshiaki Miyazaki, Atsushi Takahara, Shinya Ishihara,
Seiichiro Tani, Takahiro Murooka, Tomoo Fukazawa,
Mitsuo Teramoto, Kazuyoshi Matsuhiro

NTT Network Innovation Laboratories
Y-807C, 1-1 Hikari-no-oka, Yokosuka 239-0847, JAPAN
e-mail: miyazaki@exa.onlab.ntt.co.jp

Abstract

A novel distributed-resource abstraction environment is introduced. You can access any resource in a computer network as a memory mapped I/O device, as if it was attached to the local bus of your PC. This network technology gives us several benefits. From the application development viewpoint, no network-related programming is required, and we don't need to modify the applications even if the network topologies and protocols are changed. On the other hand, network maintenance and upgrading can be done anytime without worrying about the application users, because the environment completely separates or hides the network from the applications. The API (Application Program Interface), a resource abstraction mechanism, and a directory service are implemented. In addition, a reconfigurable hardware technology is adopted to perform autonomous network control using a low layer protocol. Furthermore, we introduce a testbed that allows heterogeneous resources to be utilized, and demonstrate the feasibility of our concept using some applications.

1. Introduction

Computer networks are rapidly growing and we can access many services across the networks. For example, it is possible to obtain various types of information using the WWW (World Wide Web), as well as accessing remote resources such as hard disks and printers. However, we must know the locations and specific access methods of the resources to use them. To remedy this situation, some technologies have been presented [1][2]. SUN's Jini [1] is a pioneer in abstracting network resources. For instance, Jini allows us to use remote printers in the same way without worrying about the differences of their physical interfaces. However, Jini is realized within a Java environment, so performance strongly depends on the lower layer protocols; its portability is, however, excellent. In addition, the current version of Jini does not support any abstraction method that can integrate more than one physical resource and present them as one abstracted resource. TINA [2] provides a machine-independent network-resource collaboration environment. With TINA, we can easily realize a new service by utilizing distributed resources in computer networks. However, it is implemented as a middleware on CORBA-based platforms, so it fails to handle the lower layer protocols well. HAVi [3] is a home-use audio/video (AV) equipment interface based on IEEE1394. We can integrate different types of AV equipment and control them using the same interface and commands, but not computers and their peripherals located far from each other because of the IEEE1394 limitations. The "Globus" project [4] is dedicated to realizing a metacomputing world by connecting multiple supercomputers and workstations. It succeeds in terms of providing huge computing power, but the aim is integrating computers, not peripherals.

Compared to the above works, our goal is to realize a network platform through which we can easily access any distributed resource without worrying about individual resource information such as its location and access method. We call each abstracted resource a "virtual resource". In our platform, the network itself is a "virtual-resource pool", and does not just provide data transmission. This is summarized in the "service-in-the-box" metaphor shown in Fig. 1. That is, users can select resources from a resource pool, i.e. the network, with the same ease as selecting a toy from a toy box, and construct their own computation environment with them. This concept is based on the "Virtual BUS", or "VBUS", architecture; its basic functions are described in this paper.

This paper introduces the VBUS model and its characteristics in Section 2. Next, its current implementation is described in Section 3. A testbed and some applications that demonstrate the VBUS functions are shown in Section 4. Conclusions and future works are described in Section 5.

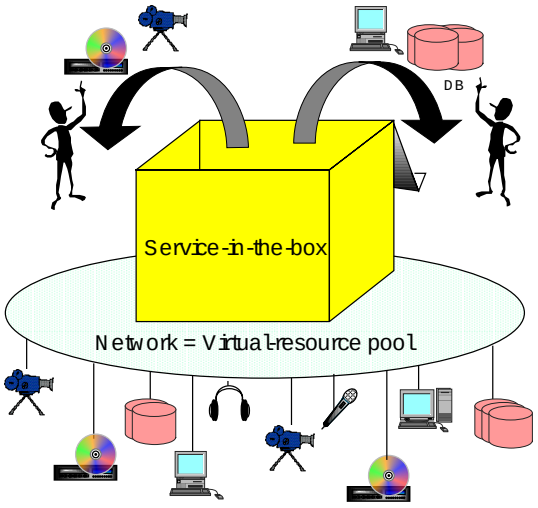

Figure 1. "Service-in-the-box" concept. We can get any distributed resource without worrying about the network and detailed access method.

2. Virtual BUS Model

To realize the "service-in-the-box" concept, we adopt a bus model as shown in Fig. 2. In this model, every resource in a network can be attached to VBUS and accessed using memory mapped I/O calls. For example, if we want to observe a remote site, all we have to do is to issue a request to VBUS. The environment provides the video camera as well as a video encoder and decoder to transfer the video signal across the network, and attaches them to VBUS automatically. A VBUS is established for each user or application as needed.

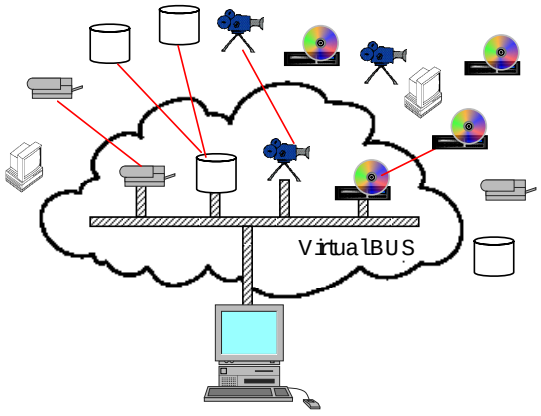

Figure 2. The virtual bus or VBUS model.

Figure 3 shows the protocol stack of the VBUS environment. Instead of the OSI seven layer protocol model, we adopted a simple three layer scheme: the service, logical, and physical layers. VBUS is defined as an application program interface (API), and lies between the service and logical layers. From the service layer, we can access all distributed resources in the network in the same way without worrying about the details of their locations or access methods. In other words, the "service-in-the-box" concept shown in Fig. 1 is realized at this layer. The logical layer hosts a resource abstraction mechanism so each abstracted resource can be accessed though the VBUS API in the same way. In addition, a directory service that enables us to access the abstracted or virtual resources using their attributes, not physical locations such as the URL (Uniform Resource Locator) used in the Jini and some other directory services. The lowest layer, i.e. the physical layer, offered by VBUS provides significant advantages. Unlike conventional network protocols, some network control functions such as signaling and routing can be automatically activated according to the traffic across the network node and/or the requests from the upper layers. This contributes to protocol simplification. To realize this mechanism, we introduced a network node constructed with programmable or reconfigurable hardware to the physical layer.

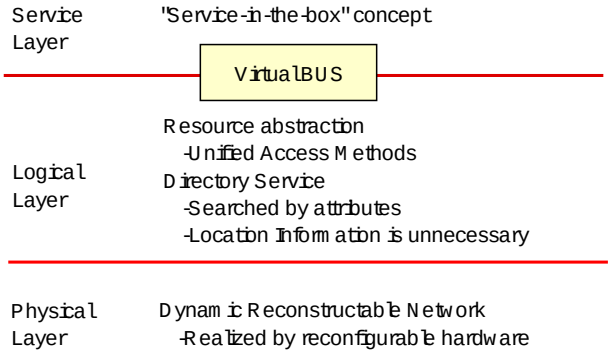

Figure 3. Three layer protocol stack model for the virtual bus environment.

The characteristics of VBUS are summarized as follows:

- Hiding the network environment from the user: Dynamic service-in/out and network maintenance can be done anytime without losing service continuity.
- Resource abstraction mechanism: This mechanism enables us to write applications using any conventional coding style without recourse to any network-related functions such as "socket" as is required in the UNIX operating system. This is a significant benefit in that we do not need to modify the applications even if physical network protocols are changed.
- Composite resource support: New services can

easily be created by combining existing resources. In other words, virtual resources can be created hierarchically. We believe that this function is very important in stimulating the establishment of second service providers.

3. Implementation

3.1. Logical Layer

The logical layer shown in Fig. 4 was realized on the above protocol stack. Each resource is encapsulated with a virtual resource driver, so it can be accessed from outside using the same interface. The resource database manages the availability of the virtual resources. The basic VBUS operations are summarized as follows: First, the application creates a VBUS by passing a request to the VBUS daemon (VBUSD). Second, the application requests a resource to the resource database across the VBUSD. Next, the database returns the identifier of a candidate as a search result to the VBUSD. The VBUSD then connects the identified resource to the VBUS. After the resources are attached to the VBUS, they can freely be accessed from the application through the API functions. The resource and VBUS release process is basically the reverse of establishment process.

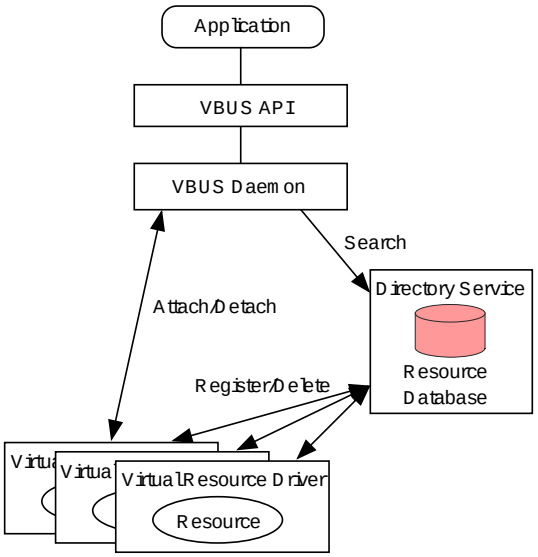

Figure 4. VBUS implementation model.

3.2. API

The API of VBUS was realized as a C/C++ callable function package. It contains the following six functions.

- "Open": is used to create a VBUS.
- "Close": deletes a VBUS.
- "Get": function is used when we want to secure a virtual resource. If this function is called, the most suitable resource is attached to the VBUS.
- "Release": is used to detach a virtual resource from the VBUS.
- "Write": sends commands to an attached resource. Each attached resource is defined as a memory mapped I/O device. Thus, we can access it by writing data or command to the corresponding address.
- "Read": is used to receive data from attached resources. This is the opposite function of "write".

The VBUS applications can be constructed quickly by using these simple functions.

3.3. Resource Database and Search

To get appropriate virtual resources, a resource database with a search mechanism is indispensable. This function is categorized as a directory service in the distributed network environment. In fact, our implementation follows a directory service specification [6]. Each entry consists of "type", "attributes", and "expiration time," and we can search virtual resources using the attributes as well as the type. The expiration time is used to delete or update entries in the database.

3.4. Software Platform

We used PVM [7], Java [8], and jPVM [9] as the software platform in implementing the logical layer functions. PVM is used as a general platform to invoke remote processes and to provide a communication mechanism between them. On the other hand, Java, especially its RMI (Remote Method Invocation) mechanism, is applied to implement the resource database and its access methods. jPVM is used to connect the database environment to each process handled by the PVM environment.

3.5. Physical Layer

The physical layer support is a unique point of the VBUS technology. Our goal for this layer is to construct an adaptive network, which means that routing and flow control are autonomously established according to the current traffic and network status. To realize this environment, we developed a reconfigurable telecom system called ATTRACTOR, which uses many FPGAs (Field Programmable Gate Arrays) and microprocessors (MPU), and features both reconfigurability and high performance [5]. An overview of ATTRACTOR is shown in Fig. 5. The system consists of heterogeneous boards for I/O, ATM (Asynchronous Transfer Mode) cell switching (ATM-SW), common purpose

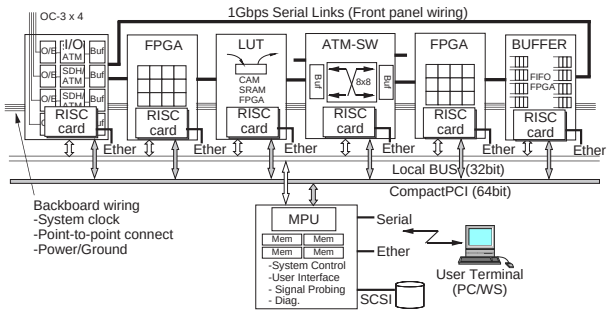

Figure 5. An overview of the reconfigurable telecom system ATTRACTOR.

look-up tables (LUT), data buffering, common purpose reconfigurable logic (FPGA), and a main MPU. Each board, except the "MPU", has a RISC-type MPU card (RISC card) as the on-board controller, and each card can be booted individually. In addition, all logic circuits in each board, not just FPGA boards, are implemented as FPGAs. Thus, each function of the board can be tuned to the desired one by reconfiguring the on-board FPGAs.

There are two key innovations in this system. One is the board-level modularity concept that allows different functions to be implemented on different boards individually. The other is a high-speed serial link mechanism that provides excellent inter-board communications without sacrificing performance. Thus, this system can be completely reconstructed to suit different applications by creating different board combinations and reconfiguring each board.

With ATTRACTOR, we can realize almost all network node functions such as routing, and traffic monitoring and shaping mechanisms, and implement them very quickly. In addition, system reconfiguration can be controlled across the network as well as from the main console. Thus, if we realize protocol behavior in the physical layer of VBUS, node functions can be changed dynamically. An example of using ATTRACTOR is shown in 4.3.

4. Testbed

We constructed a testbed as shown in Fig. 6 to check the VBUS concept. Most of the PC's were connected to two LAN's; 100Base-T Ethernet and OC-3 (155Mbps)-interfaced ATM LAN. The Ethernet was mainly used to transfer the logical layer protocol, while the ATM LAN was used to transfer heavy traffic such as encoded video signals. For ATM, three ATM switches were connected to each other to form a ring topology. We provided some virtual resources, i.e., memories, video cameras, Motion-JPEG (MJPEG) encoders and decoders, MPEG2 encoders and decoders, and miniature robots. Each resource was physically connected to one of the PC's in the testbed across the PCI bus or a serial port, and its virtual driver runs on the PC. The resource database was run on one of the PC's connected to the Ethernet. Two PC's in this testbed were provided as user terminals, and equipped user interface software, MJPEG and MPEG2 decoders, and joysticks. Furthermore, ATTRACTOR, a reconfigurable telecom system, was structured to realize a dynamic VC (virtual channel) switch function (described in 4.3), and was located downstream of each user terminal.

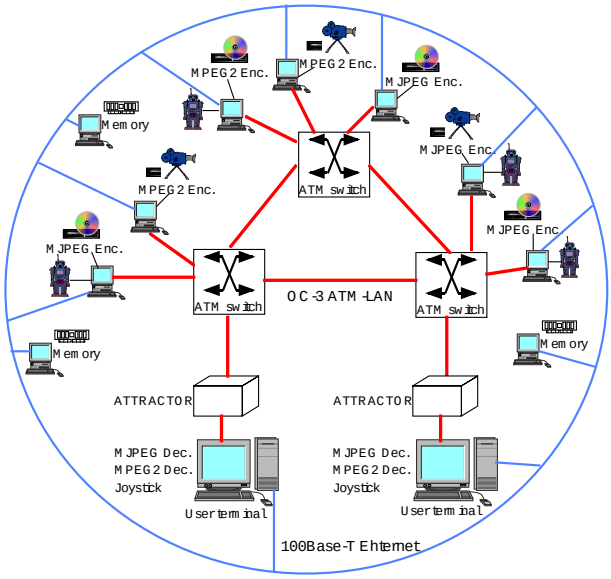

Figure 6. An overview of the current VBUS testbed.

4.1. Network-Wide Virtual Memory

The first function was virtual memory. It allows us to use the memory resources of the remote PC's as local memory. A unique point of this function is that we can use a large memory, which may be too expensive to install on one PC, by combining chunks of memories offered by many PC's. From the implementation point of view, this is a simple example of the composite resources that VBUS can realize. An experimental result of virtual memory access is shown in Fig. 7. Four different virtual memories, 32, 64, 96, and 128M bytes, were set. Each was constructed using 32M byte real memory resources provided by several PC's. The y-axis represents the elapsed time for writing/reading data to/from the virtual memory (access by 8k/16k byte block units), while the x-axis represents the size of virtual memory. In the current implementation, the access time of the virtual memory is almost twice that of real memory. However, this function is scalable; thus we can logically access one huge memory if a lot of PC's

are connected to VBUS.

The current version of virtual memory does not employ any sophisticated techniques such as data caching [10]. In addition, the main performance bottleneck is low speed of the Ethernet system used. In fact, according to our preliminary experimental results, the total setup time for writing to and reading from the virtual memory is less than one second in all cases shown in Fig. 7. Thus, the access time can be greatly enhanced by adopting a distributed shared memory technique as well as a higher-speed network platform.

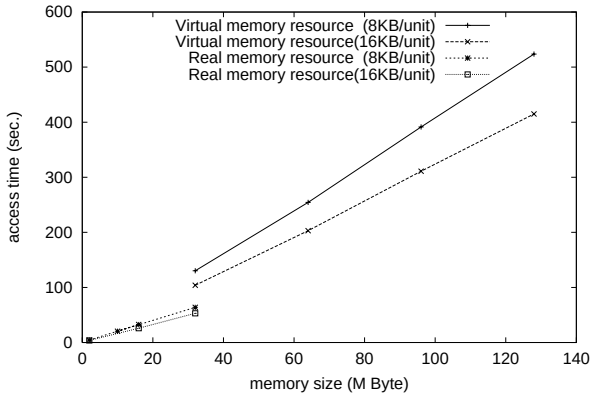

Figure 7. Evaluation of network-wide virtual memory.

4.2. VOD Service with Adaptable Quality to Match Individual Situation

A video on demand (VOD) service was created using the VBUS environment. A unique point of our VOD service is its ability to automatically choose the combination of the service source and its encoding method that provides the highest quality service possible to the user. For example, given a wide-band traffic path, e.g. more than 30 Mbps, the resource database of VBUS recommends the MJPEG site if both MPEG2 and MJPEG sites are available, because MJPEG quality is better. However, VBUS would select the MPEG2 site under a narrow-band situation, because the required traffic rate of MPEG2 is 6 Mbps while that of MJPEG is 30 Mbps. This action is performed at the logical layer in our protocol stack, and so is hidden from the application layer. Thus, we can receive the VOD service with the best quality possible given the current network environment.

4.3. Dynamic VC Switch

We implemented a dynamic VC (Virtual Channel) switch using ATTRACTOR. As shown in Fig. 8, ATM cells coming from n-input ports ($n \geq 2$), which are identified by the VCI in the ATM header, are continuously monitored. The VC that can carry the largest traffic load is automatically connected to the output port. This action is repeated periodically, e.g. every second. Thus, for example, if we transfer the same video source across different VC's, i.e. different ATM paths, this system can choose the highest quality path consistently. To demonstrate this function, we sent the MJPEG data over two different ATM paths with different traffic rates through an ATTRACTOR system. We then confirmed that this system switched quickly from the higher traffic path to the lower one, if the higher traffic path were cut.

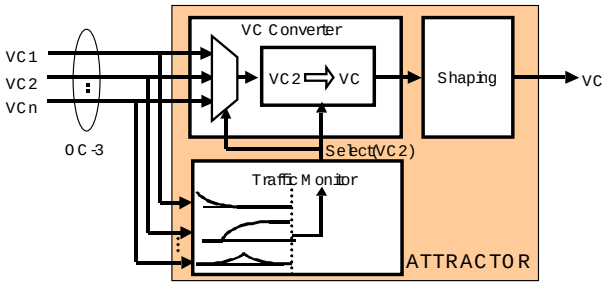

Figure 8. A simplified block diagram of the dynamic VC switch function implemented in ATTRACTOR.

This is a basic mechanism needed to realize resource roaming. For instance, if the same VOD services exist in a network and they are synchronized, the user can be consistently connected to the video site that offers the highest quality by dynamic switching of the routing path. In addition, if we combine this mechanism with the automatic and appropriate service-selection method described in 4.2, more efficient services, which can change the service site as well as the routing path autonomously, can be constructed.

4.4. Remote Robot Control

Finally, we created a remote robot control game by integrating the above mentioned services and several kinds of resources such as miniature robots, video cameras, and a joystick controller. Figure 9 shows a snapshot of the game. The user display is divided into three parts; a remote camera and robot controller, a display window for the attached camera, and a VBUS monitor. We can freely get and release several remote cameras and/or robots using the controller panel. The zoom-in/out and viewing angle of the attached camera are manipulated using the joystick as well as the mouse and keyboard. In the case of robot control, the joystick is used to change its moving direction and speed as well as to choose the robot to be directed. The at-

tached camera view is displayed on the user's screen. The VBUS monitor is convenient to check the resources currently attached to the VBUS. The monitor display is refreshed every second and the most recent status of the VBUS is reported. As mentioned before, if we request a camera, a set of video encoding and decoding hardware resources, which are needed to send the video signal from the camera to the user's display, are also attached to the VBUS automatically. Here, the cameras and robots only accept the commands sent from one user at a time. This is performed by writing an occupied flag to the resource database when the corresponding resource is occupied. Because the resource database does not report the occupied-flagged resources as available resources, exclusive control is naturally realized. For example, if we want to make a network-wide interaction game that involves several players, a dedicated game server would be needed to manage the game status. However, we can create such a game without any dedicated servers if we utilize the exclusive control function of the resource database. In fact, we can create a kind of "robot soccer game" just by invoking the same application program shown in Fig. 9 on several PC's at the same time.

The VBUS environment provides dynamic service-in/out or "plug-and-play" capability for all resources and services. In this example, we can add and release extra-robots to/from this game at any time, and the changes are reflected immediately on the user display. This means that the networks and resources can be maintained without losing service continuity if we apply the VBUS to actual networks. The plug-and-play capability greatly benefits service providers and users.

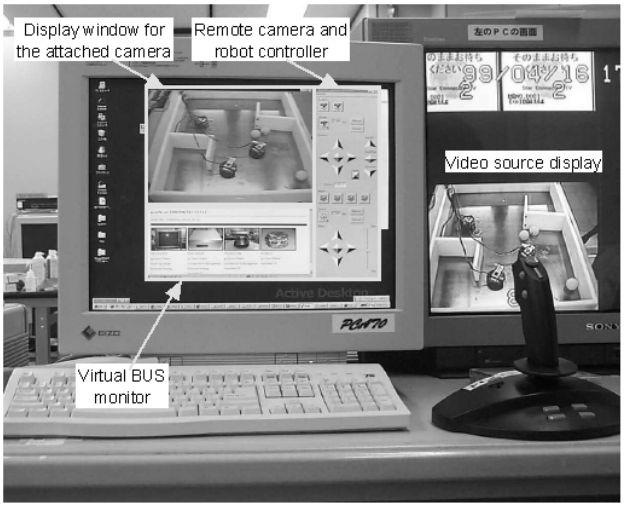

Figure 9. A snapshot of the remote robot control game.

5. Conclusions and Future Works

A novel distributed-resource abstraction environment was introduced. In the environment, we can access any distributed resource in computer networks as a memory mapped I/O device, as if it were directly attached to the PC. In addition, a composite-virtual resource can be realized by combining several existing resources. This network technology gives us several benefits. From the application development viewpoint, no network-related programming is required, and we don't need to modify the applications even if the lower-layer network topologies and protocols are changed. On the other hand, network maintenance and upgrading can be done at any time without worrying about the application users, because our environment completely separates or hides the network from the applications. To create a testbed, we wrote several software libraries for the API, resource abstraction, and a directory service. In addition, a reconfigurable hardware technology was adopted to realize service-oriented network control in the lower protocol layer. The testbed combined many PC's and peripherals using an ATM LAN and Ethernet, and several applications were demonstrated to show the feasibility of our concept.

Unfortunately, the information interchange between the different protocol layers implemented as software and hardware is not smooth, thus synchronizing the protocol state transition between them is difficult in the current implementation. Easing this problem is one of our future tasks. Another future task is to expand our directory service mechanism from the current concentrated implementation to a multi-agent style in order to handle a lot of distributed resources efficiently.

References

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

[2] TINA Consortium, "TINA-C Home Page," (http://www.tinac.com/).

[3] "The HAVi Architecture," Version 0.8, 1998.

[4] I. Foster, C. Kesselman, "The GRID: Blueprint for a New Computing Infrastructure," Morgan Kaufmann Publishers. Inc., 1999.

[5] T. Miyazaki, K. Shirakawa, M. Katayama, T. Murooka, and A. Takahara, "A Transmutable Telecom System," Proc. FPL'98 (Springer LNCS 1482), pp. 366-375, Tallinn, Estonia, September 1998.

[6] "Service Location Protocol," IETF RFC-2165, 1997.

[7] "PVM: Parallel Virtual Machine," (http://www.epm.ornl.gov/pvm/).

[8] "Java," (http://java.sun.com/).

[9] "jPVM: A native methods interface to PVM for the Java platform," (http://www.isye.gatech.edu/chmsr/jPVM/).

[10] P. K. Sinha, "Distributed Operating Systems— Concepts and Design," IEEE Press, 1996.