Seogwipo Cheju July 7-9, 1988

The Harmonic Connection Concept in the Trinity Initiative of Aoyama Gakuin University

Masayuki Ida* and Keisuke Tanaka†
Computer Science Reserach Lab.
Information Science Research Center
Aoyama Gakuin University
Address: 4-4-25 Shibuya, Shibuya-ku, Tokyo Japan 150

Abstract

This paper describes the harmonic connection concept, and its design and an implementation of us starting from April 1988 on.

The harmonic connection is the concept to build a distributed environment among widely interconnected computers or local area networks, in one organization. The characteristic of the harmonic connection is to provide one virtual distributed OS with moderate comminication lines.

Apostle for the Trinity Initiative of Aoyama Gakuin University is a case of the harmonic connection and is currently under developement. *Apostle* connects three campuses of the university. Local Area Network of each campus is connected with each other, and one loosely coupled distributed computing environment is provided. In this environment, traffic on the link for inter-campus connections is controlled to low. The major components of *Apostle* are the information server, the user interface, and the file cacher.

1 The Harmonic Connection

– A loosely coupled distributed OS –

1.1 Distributed OS and Network Integration

There are increasing needs to have distributed OS or network integration tool over the computers on one network. Distributed OS approach has the following characteristics:

1. computers/workstations on one network form one virtual machine/OS.

And as the results of (1),

2. resources are (or can be) shared among workstations.
3. it can have some load balancing mechanisms.

And for users,

4. it provides an uniform access to his environment. (he can log-in any machine on the network and can use the same environment with the same fashion.)
5. it can save/lessen the cost of maintenance/administration and the cost of training, since each machine has a same mechanisms.

These characteristics/merits are attained with the costs of

(a) traffic overhead on the network, which is usually high, and

(b) cpu/memory/disk resource consumption overhead on each machine.

* ida%aoyama.junet@relay.cs.net
† keisuke%cc.aoyama.junet@relay.cs.net

We count Mach[Accetta86] as typical distributed OS. In our paper, we neglect the systems structures as in butterfly[BBN87] which are originally designed for parallel processing and are to be used as a whole. Mach is a distributed OS originally developed at CMU. It has a process migration mechanism. It has a portable design and implemented on several different machine architectures.

Meanwhile, network integration is mainly achieved with network file systems like NFS[SUN86a]. And with further assistance of YP[SUN86b], a system may act as one environment though each workstation is distinguishable. They are viewed as a connection facility which shares environments. To have an convenient system, NFS assumes an ethernet.

1.2 A Definition of the Harmonic Connection

We are thinking of the connection of the machines whose physical locations are quite different but the organization to which they belong is the same and the machines are shareable.

We define the harmonic connection, our concept of integrated system of location independent computers, as a loosely coupled distributed OS.

The characteristics of the harmonic connection are;

1. It is for a distributed OS.
2. the whole component computers do not always reside in the same location.
3. a linking medium does not always be required a high bandwidth. Even if a medium is slow, the system should work as convenient as possible. Implementation should cope with variations of network media.

Table 1 shows the position of harmonic connection among several different connection scheme; tight coupling, network integration, communication, and message routing.

2 The Trinity Initiative

2.1 The Environment

– a case for the harmonic connection –

The harmonic connection concept is the heart of the trinity initiative of Aoyama Gakuin University (AGU), and is also a generally applicable concept for a distributed environment inside one institution.

The trinity initiative started in April 1988 and is a project inside the university to tie up the computing facilities which have dispersed among campuses.

Aoyama Gakuin has 118 years history and has divisions from kindergarten to university with graduate schools. The institution has as its objective a coherent education dedicated to the spirit of the Methodist Church of the Christian faith. AGU has six colleges and has three campuses, that is, Aoyama campus, Setagaya campus, and Atsugi campus. We have about 19,000 students. 10,000 students belong to Aoyama, 2,000 to Setagaya, and 7,000 to Atsugi. Fig. 1 shows the physical locations of these three campuses. The Aoyama campus has the institution's headquater and is located in the central part of Tokyo. The Setagaya campus is for the college of Science and Engineering except for freshmen. The Atsugi campus was built in 1982. Freshmen students of all other colleges and schools study for two years at Atsugi and move to Aoyama. All evening Division and Graduate Division students study exclusively on the Aoyama campus.

The distance between Aoyama and Setagaya is 13 km, Aoyama and Atsugi is 45km, and Setagaya and Atsugi is 35km.

2.2 The Characteristics of the Trinity Initiative and its needs

Information Science Research Center (ISRC) is in charge of the whole computing facilities of AGU and has branches for each campus. At the Setagaya campus, the super computer SX-1EA is scheduled to be there in October 1988. The Atsugi campus has hundreds of personal computers for teaching freshmen and introductory courses. The Aoyama campus has the gateways

for several networks including JUNET, and several commercial network services. Computer Science Research Lab. (CSRL), our group, is newly formed in April 1988 and is located in the Aoyama campus. Fig. 2 shows the connection of the computing facilities. ISRC has experiences with the DDX-P connection and then high speed digital line connection for 5 years for hierarchical main frame TSS system, and N-1 link to University of Tokyo. We have been coped with several different networking requests. Namely, personal computer connection, main frame TSS connection and UNIX-oriented network. These types of connection should have equal services for the three campuses of us. The Trinity Initiative is to connect these three different *persona* with a uniform or a single network. Fig. 3 shows the model of the Trinity Initiative. Trinity is the integration of three campuses. And, Trinity is the integration of three levels of computing facilities, that is, super-computer-TSS, workstations and personal computers.

.We need an uniform environment among the three campuses. Because lots of users, i.e. the professors, must migrate among the campuses. There is a data that 30% of some 500 permanent faculty members have their roles on two or three campuses. They need the same environment and they need to access the same file system. Since almost all of them are not computer science related professors, they need a simple , same and easy to use environment. They need mailing and data retrieving facility at first. To cope with this migration of persons, the simplest approach is to carry portable WS or PC by themselves. But it is not realistic solution. The best way is to have a tight connection of three LANs with very high speed lines (and provide adequate workstations everywhere). But its not feasible for our case, since the total traffic seems to be not so high to afford expensive connection. This is the reason why we need the harmonic connection.

3 System Architecture

3.1 Design Considerations

We consider two types of considerations. One is on the management/administration, and the

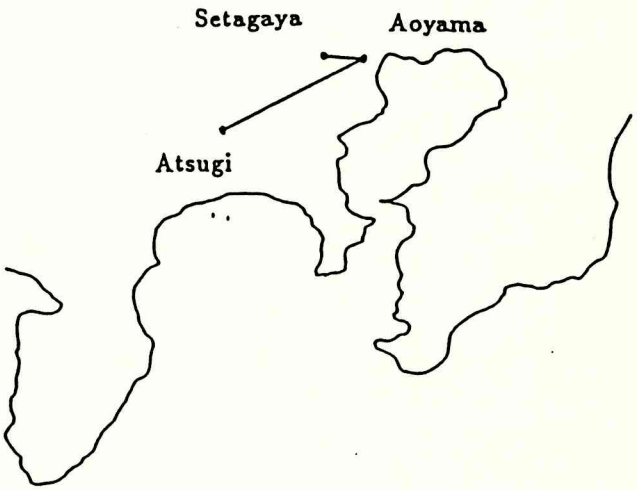

Figure 1: three campuses of Aoyama Gakuin University

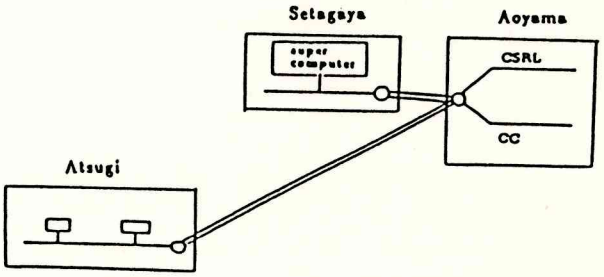

Figure 2: The connection of computers of three campuses

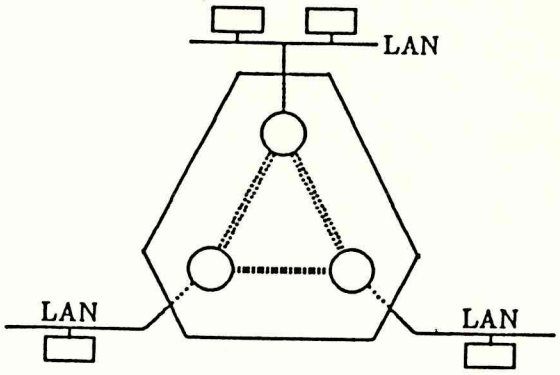

Figure 3: The Trinity model

other is on the extensibility.

A. Management/Administration issues

We point out the following items.

1. security consideration:
   Since the users are basically casual, the system should be secure.
2. name management:
   Since there are lots of subdivisions in one organization, the name management facility should have the same hierarchical structure. It should be separated from '*root*' and divided.
3. compatibility with the current commercial OS:
   The system should share the fruit of the current technology.
4. request of a user to integrate his own machine which is out of concern for the Trinity:
   The system has no provision for this issue.
5. allocation of files:
   The system is upward compatible with UNIX file system.
6. auditor:
   Accesses and modifications of administrators should be auditorable.

B. Extensibility

We need the ability to freely upgrade the components without the destructive replacement of the whole.

3.2 A System Configuration

Our requirements are summarized as follows.

A) Uniform OS: for user education, and for staff management/training
B) Extensibility/Expandability: the ability to evolve/upgrade
C) Advancedness:
   Better environment/software. International Grade and compatibility. high performance
D) Moderate line to connect

The principles for implementations of the harmonic connection in the Trinity Initiative are:

1. use a moderate line to connect.
   The traffic inside the backbone should be minimized as possible. To do so, the process the user invoked by typing a command or by daemon process, should be executed on the machine he logged in and *not* on the machine he has a home environment. It is not feasible for us to have a tight connection to cope with this assumption.
2. provide user an uniform environment.
   User can access his home directory and the same environment automatically wherever he login.
3. compatibility with the commercial OS.
4. employ TCP/IP technology for LAN link to cope with different higher (application) level protocols employed by different vendors.

With these principles, we design the system of us with the following decisions:

1. Use SUN-4 and SUN Internetwork Router [SUN87] which is for a physical and data link layer under IP level over a synchronous line, for backbone connection. It means this uniform OS becomes SUN OS, whose 4.0 or later version is said to be compatible with AT&T UNIX System V R4.0 or later.
2. Use two 64K NTT super digital lines to connect three campus gateway processors. (Actually, we are trying to integrate them to Multi Media Muxes of NEC, which AGU is already using.)
3. As a whole, the three machines become one large machine from users point of view.

Fig. 4 shows the whole system. Three Sun-4s are the backbone of the system and are tied together with the harmonic connection concept.

4 Design of *Apostle*

The system which we are implementing is called '*Apostle*'. *Apostle* has three component computers. They are arranged to three campuses each by each as shown in FIg. 4.

4.1 Requirement Analysis

Apostle is analyzed to have the following characteristics:

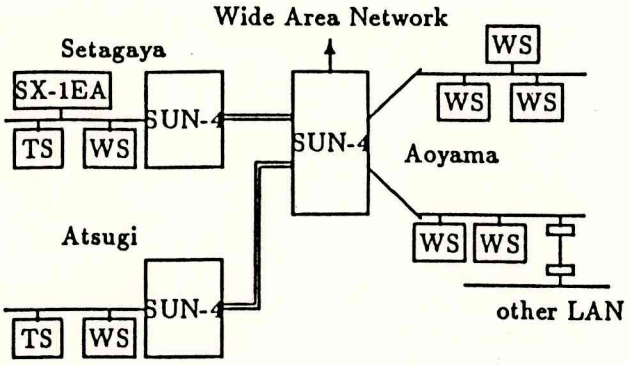

Figure 4: The system configuration

1. Since all accesses are defined to be secure, everyone is prohibited to access private files of others. He may know who uses the system or what is his name in real life and so on. But he cannot know even the existence of private files of others. To implement this scheme, we adopt a restricted shell technique as a base. This restricted shell requires the division of the whole file system into two categories; public files and private files. Accesses to private files of others are inhibited by this shell.
2. Each user has one virtual environment. He is given his own environment independent from his physical location. To achieve this scheme, information about his working environment must be shared among components on the network.
3. The component computers in every campuses have the copies of the database, which means a set of information about users. There is no needs to cause a traffic across campuses to refer a record of the database.
4. Private files are cached when the owner of them accesses from different component.
5. A person is not allowed to log-in from different place at the same time. This assumption gives some merits to the implementation of *Apostle*. (File cacher described in 5.3 becomes simpler.)
6. The system will be implemented with faculty of Inter Process Communication (IPC). The functions for *Apostle* are arranged to some sub systems or processes. These processes has communication channel for IPC and their interface uses Remote Procedure Call (RPC) mechanism.

4.2 Structure of *Apostle*

Apostle consists of three sub systems; an information server (IS), a user interface (UI) and a file cacher (FC).

The information server has three his *alter ego* on each component computer. Each ego of the information server is called a replica of IS. The user interface is a shell described in 4.1(1). Each component computer has its own file cacher.

An information server provides a mechanism to share information of individuals among component computers. A user interface provides a single virtual environment for each user. A file cacher provides the method to access a private file even if a user loges in different component. Fig. 5 shows the structure of *Apostle*. Each replica of IS is in each campus. UI interprets the command line given by a user. UI judges whether the user is permitted to execute the command and to access a file whose name is typed on a command line. For this judgment, UI communicates with a replica of IS on the component computer on which UI itself works, and refers a record of the database. After this judgment, UI executes the command and accesses the file. If the file is a private file of the user and it is located on the different component, UI invokes a function of file cacher. FC transfers the copy of the specified private file into the component which the user loges in. If the copy of the private file was already transferred, FC omits the transfer and UI accesses it immediately. With the assumption of 4.1(5), this mechanism is not so difficult to implement. The detail is described in 5.3.

5 Implementation of *Apostle*

5.1 The Information Server

5.1.1 Structure of the Information Server

Each component computer of *Apostle* has a replica of IS. Each replica has the copy of the same database. The reason is as follows:

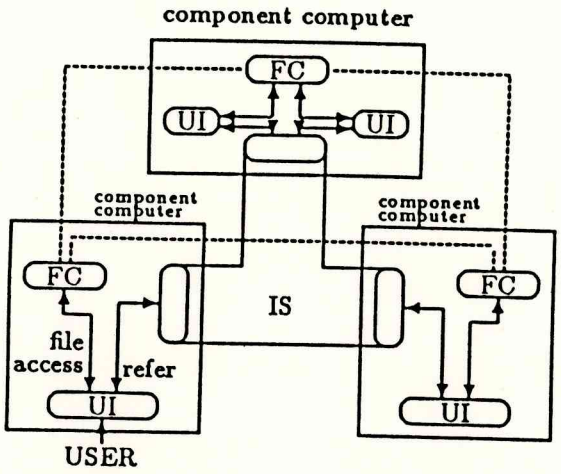

Figure 5: The structure of *Apostle*

1. references to a record of the database is more frequent than change, and
2. the reference across the inter-campus link needs more cost than reference in one component do.

Each replica communicates with each other, and keeps the copy up-to-date.

A user interface or other applications communicate with a replica to refer/change a record of the database.

5.1.2 The Contents of the Database

The information server provides the information about users. The database contains the following information:

- user name:
  This name is a symbol given as a human-readable string. Every users on *Apostle* are identified by this name. This name is effective to exchange messages with persons out of *Apostle*.
- user account code:
  This is another identifier for a user. A local operating system of each component uses this code. Because a person is identified by his user name, he does not have to know his account code. A system administrator may refer this code for his operations, such as user accounting.
- information about user working environment:
  This information tells where his private files are located, and where his mail box is. Currently, three types of slots are arranged for this information; 1) home directory for private files, 2) mail box to hold messages, and 3) group directory for private files of several persons. These slots contain a) the name of a component on which files are located, and b) the path name for a directory or a file.
- private information:
  This is a personal information of user; the name of a user in his real life, his address, and so on. *Apostle* itself does not need this information but some applications on *Apostle*, such as directory service, may refer/use this information.

5.1.3 Database Access Mechanisms

The scheme to refer to a record is very simple. To access the database, UI or some applications are only required to communicate with a replica of IS on the same component.

To keep the integrity of the database, the procedure to update an entry of the database needs special treatment. To avoid the inconsistency between copies of the database and to avoid a heavy load for checking the inconsistency or repairing it, we design the procedure to update as follows:

1. A replica of IS accepts the request to change an entry of the database.
2. The replica refers a record of his own copy of the database, and checks whether the request is valid or not. If it is invalid, it will be rejected.
3. The replica verifies if all replicas are alive. When one of them is dead, the replica which accepts the request, wait until all replicas become alive. In this case, the request does not reflect immediately to the database.
4. When all of replicas are alive, the replica tells them the new information, and every entry in copies is changed.
5. If two requests are given at the same time and they conflict with each other, the request will be rejected.

Creation and deletion of an entry is achieved with the same manner as above.

Only an owner (or an administrator) is permitted to update a record of the database. A user identified by the user name in the record is defined to be the owner. No one is permitted to change other's records. But he can refer a record of others, if the owner permits.

5.1.4 Extensions to YP

The information server is implemented as extensions of SUN's Yellow Pages (YP)[SUN86b]. SUN defined several maps for YP protocol to share database for network administration. We define new maps for the information server of *Apostle*. These maps provide the translation of the name of user to other contents of the database for *Apostle* described in 5.1.2.

Since YP protocols itself has no needs to be modified, the YP facility offered by SUN is preserved. We modify *ypserv(8)*, which is YP database server, and *ypbind(8)*, which is YP subsystem to find YP database server.

For the procedure to update a record of the database, we implement a new YP subsystem, *ypupdate*. This new subsystem accepts the request to modify a record of the database, and modifies all of copies of the database on each component computer. After modification, it tells the YP database server, *ypserv*, that the database has been changed.

The relationship among these processes is shown in Fig. 6.

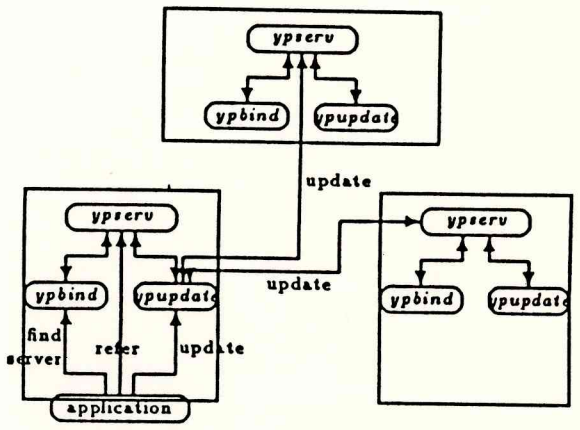

Figure 6: The structure of the information server

5.2 The User Interface

5.2.1 Classifications of Files

The user interface assumes the division of the whole file system into the following categories:

1. public files:
   Common commands and data files for these commands belong to this category. Also directories to create temporary files are in this category.
2. private files:
   This category includes all files located under the home directory of a user.
3. group files:
   These files are shared by several persons. Files for some project should be shared by members of the project. These files are in this category.
4. system files:
   These files depend on the individual operating system of each component computer of *Apostle*.

5.2.2 Allocation of Each File Class

Private/group files are located on one component computer on the network. When a user wants to refer his private files on a different component, copies of files are transferred from his home to a temporal directory on the component he loges in by the function of file cacher. Since these files are cached, the next reference to the same file does not need the traffic between components.

Each component has its own public files. *Apostle* itself does not provide the mechanism to keep the consistency among public files on each component.

System files are managed in each component. System files on a component are independent from those on other components. They are used for a local operating system on the component. Every users, except for an administrator of a component computer, is not permitted to access these files.

5.2.3 Access Rights

Four types of users are defined; 1) casual users, 2) administrators for one administrative domain, 3) system administrators, and 4) directors of *Apostle*.

Casual users use public files and private/group files to work. They can create temporary files under a public directory on a component computer he loges in. But they are not permitted to modify public files prepared by the system. The access to private/group files of others are prohibited.

Whole computing environment is divided into some administrative domain. Usual administration, such as the registration of new users, is done by an administrator of the administrative domain. This type of administrator is not permitted to modify files for *Apostle* or files which depend on the local operating system of a component computer.

System administrators manage files for a local operating system. They maintain system files to make a component computer or a network between components to work well.

Directors of *Apostle* control the whole system. They are permitted to access every files on components. Commands or subsystems for *Apostle* are maintained by these directors.

UI recognizes the type of a user, and judges the access right of the user. The classification of the types of users makes the access more secure than traditional UNIX, and tasks for maintenance of the system are arranged into three types of administrators.

5.2.4 Mechanisms

UI will be implemented through two steps.

The first step is the extension of shell program of UNIX. On this stage, all available commands are registered into the command table. This table tells the name of a command, the path name for the command, and its attribute. The attribute is a general description for the command.

By using this table, UI determines which file is needed by a command, or whether the command modifies the file. When a command line is given to UI by a user, firstly, UI checks the access right of the user for the file. When the user is permitted to access the file and the file is a private file, UI communicates with file cacher on the same component. File cacher creates the copy of the private file and tells UI the path name of the shadow file. Then, UI converts the name of file on the command line into the path name of the shadow file. Finally, UI executes the command by using the command line converted.

In this scheme, the shell program evaluates only a command line and the name of a file should be described explicitly on it. Therefore the access security is not enough, because the name of a file is not converted and judgment for an access right is not done inside processes. The second step cooperates this program.

The second step is the modification of system calls on which path name is given as a parameter, such as *open(2)*, *stat(2)*. UI checks an access right through the communication with the IS, then converts the file name (if necessary), and executes the command. In the primitive of a system call, the access right is checked, and file cacher is called. Dynamic link editing for shared libraries supported by SunOS 4.0 or later is a candidate of the substrate mechanism.

In this scheme, many faculties of the previous version of UI will move into primitives of system calls. Every accesses become more secure, and file cache will be available even if the path name of a file is not declared explicitly.

5.3 The File Cacher

5.3.1 Key Techniques

When a user loges in a component computer different from the component on which his private files are located, the mechanism of file cacher is used for a rapid response.

The access over inter-campus link makes the traffic too heavy. The file cacher transfers the copy of a private file from his home directory to a temporary directory the component he loges in. This transfer is done upon the first reference from the user interface, and the next reference, even it might be a destructive update, to the same file does not cause the traffic over inter-campus link. When the user loges out, these copies are removed or some of them are wrote back into the original

files, if they are modified.

The user interface or some primitives for file handling call the file cacher. Or each file cacher communicates with each other, if necessary.

5.3.2 Copies Created by the File Cacher

There are two types of copies created by the file cacher. One is the contents of a file, and another is the attribute of a file. Usually, the size of the contents of a file is much lager than the size of its attribute. Therefore, if a command needs only the attribute of a file, the contents is not transferred.

5.3.3 File Locking

Even if a private file is cached and accessed to modify, there is no need to lock the original file, because of the assumption that a user is not permitted to log-in from different place at the same time. But group files should be locked to avoid the confliction on the modification. When a group file is transferred for modification, a modification-inhibit mark is flagged on the original files. The UI sends a request to clear cache when the user loges out or he types a special command to clear cache. After the file cacher receives these request, it transfers the copy into the original file and clears the modification-inhibit mark. Otherwise, if the file is not modified, the file cacher omits the transfer.

6 Final Remarks

Since the project started in April 1988, we had not yet finished the whole design and implementation, and we cannot have a review and an evaluation.

Table 2 shows an estimation of the comparison of five types of connections concerning the cost assessment. As shown in the table, we expect the cost merit over tight connection though it can provide a resemble environment as tight connection.

The Trinity Initiative is a three year plan and The first version of *Apostle* is scheduled to finish at the end of March 1989. The details of *Apostle* will be published as [CSRL88a,CSRL88b]. We will have a schedule to write papers about the result in the next year.

The harmonic connection itself is a quite general enough to apply other cases. So, we would like to find any other applicable cases.

Acknowledgment

The authors would like to express their gratitude to Prof. Hiroshi Oyachi (director, ISRC of AGU), Prof. Hisao Nishioka (president, AGU), and Prof. Kinjiro Ohki (chancellor, Aoyama Gakuin) for their continuous encouragements. Also thanks are due to JUNET/WIDE research group members and Prof. Haruhisa Ishida of University of Tokyo for their technical communications.

References

[Accetta86] M.Accetta, R.Baron, W.Bolosky, D.Golub, R.Rashid, A.Tevanian, and M. Younge, *'Mach: A New Kernel Foundation for UNIX Development,'* in Proc. of USENIX 1986 summer conf. pp93-112, 1986.

[BBN87] BBN Advanced Computers Inc. *'Butterfly Product Overview,'* Oct.14 1987.

[CSRL88a] Masayuki Ida and Keisuke Tanaka, *'The* Apostle *System Overview,'* CSRL Technical Report #88-001, Aug. 1988.

[CSRL88b] Keisuke Tanaka and Masayuki Ida *'The* Apostle *System Referrence Manual,'* CSRL Technical Report #88-002, Aug. 1988.

[Lyon84] B. Lyon, *'Sun Remote Procedure Call Specification,'* Technical Report, 1984, Sun Microsystems, Inc.

[Postel81a] J. Postel, *'Transmission Control Protocol,'* RFC793, Sep. 1981.

[Postel81b] J. Postel, *'Internet Protocol,'* RFC791, Sep. 1981.

[SUN86a] Sun Microsystems Inc. *'Network File System Protocol Specification,'* Feb. 1986

[SUN86b] Sun Microsystems Inc. *'The Yellow Pages Protocol Specification,'* Feb. 1986

[SUN87] Sun Microsystems Inc. *'SunLink Internetwork Router System Administration Guide,'* Jul. 1987