The Information Server

–A User Information Database System for Apostle–

Technical Report No. 88-002

Keisuke Tanaka* and Masayuki Ida†
Computer Science Research Laboratory
Information Science Research Center

Aoyama Gakuin University
Address: 4-4-25 Shibuya, Shibuya-ku, Tokyo Japan 150

Feburary 1989

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

Abstract

The Trinity Initiative is a project of Information Science Research Center of Aoyama Gakuin University. A distributed system called '*Apostle*' is the kernel of the Trinity Initiative. *Apostle* provides an integrated computing environment to users among loosely coupled component computers of three campuses of Aoyama Gakuin University. All the component computers are connected each other via synchronous lines. The Information Server (IS) is one of the subsystems of *Apostle*. This paper describes a design and an implementation of IS.

IS is upward compatible with YP (Yellow Pages). IS is a distributed database system to manage information on users. *Apostle* uses this information to provide a computing environment to a user. IS has the following features. 1) IS provides the same user information to all the software subsystems and all the users on all the component computers. 2) Each component computer has the replica of the same user information database. 3) References to the user information database of IS are applied to the replica of the local component computer, so as not to cause a traffic on serial lines upon every reference. 4) Mechanisms for simultaneous update of all the replicas are designed to have the shortest critical region to avoid the possibility of inconsistency as possible. 5) IS divides the roles of management of the database into several group managers.

1 Introduction

1.1 *Apostle* System

The Trinity Initiative is a project developed by Computer Science Research Laboratory (CSRL) of Information Science Research Center (ISRC) at Aoyama Gakuin University (AGU). AGU has three campuses; Aoyama, Setagaya and Atsugi. The Trinity Initiative provides a network over these three campuses. It provides an integrated computing environment on this network. The development of this project has the following two phases:
1) construction of a distributed operating system called '*Apostle*',
2) construction of an integrated computing environment over whole network of ISRC using *Apostle* as the core.

Figure 1 shows the network for the Trinity Initiative.

Apostle works on the three gateway computers of campuses. Each gateway is called 'component computer'. Operating systems on these three component computers are virtually integrated into one distributed operating system. For this purpose, we designed three software subsystems for *Apostle*. One is the user information database subsystem called Information Server (IS). Another is the User Interface subsystem (UI). And the other is the File Cacher subsystem (FC).

IS manages information on users of *Apostle*. Other two subsystems use the information served by IS. UI provides a uniform computing environment to users. User can log in to the same environment which is independent from the component computer he uses. FC is a support mechanism to access files on remote machines from the local machine the user loges in. We choose UNIX (SunOS) for a base Operating System for '*Apostle*'. The overview of *Apostle* is described in [CSRL88].

This paper describes the details of the Information Server of *Apostle*.

1.2 User Management with User Information Database

In general, most computer systems have databases holding information on user computing environments. Information stored in a database is used to set up user computing environment, to check file access right, and to keep the user's behavior under control. In other words, all activities of users in a computer system are authorized using user information stored in a database. We think that to manage users is to manage user information database.

Since *Apostle* is to provide a same computing environment on all the component computers, all the component computers must share this user information database. IS provides a user information database for this purpose.

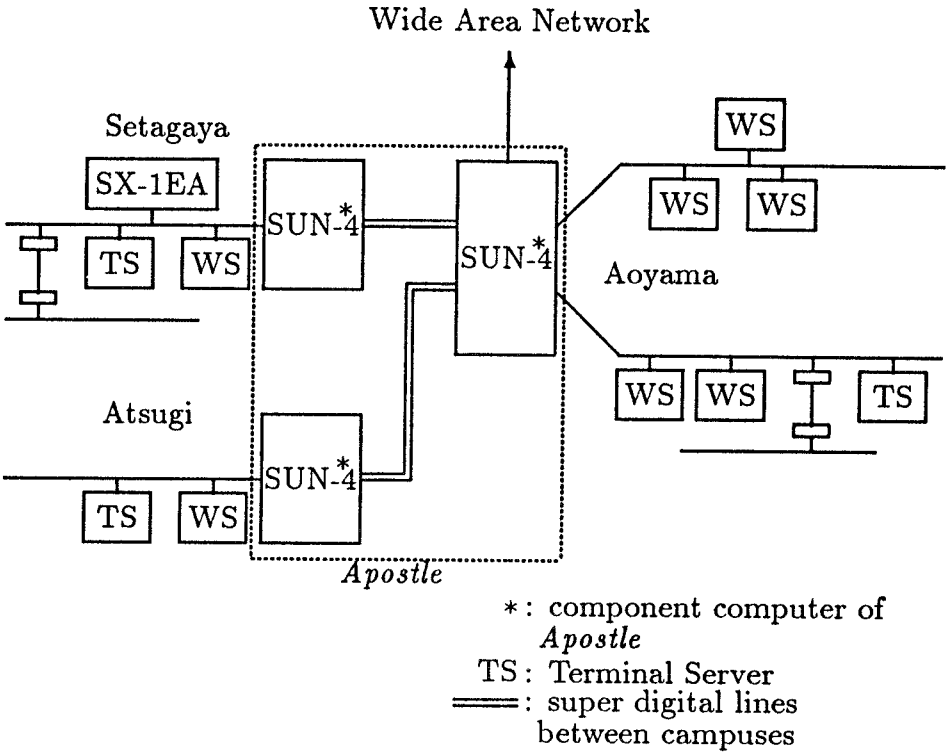


Figure 1: A Network for the Trinity Initiative

1.3 YP: A Base for IS

This section introduces YP (Yellow Pages) to make this paper self contained.

YP[SUN86] is composed of two subsystems; one is *ypserv*, and the other is *ypbind*. *Ypserv* serves the data reference service and is on every network server. The primary role is the handling around the DBM[1] database reference function. *Ypbind* is a server for any clients on the network which request to use YP functions. *Ypserv* resides on one server of the network. *Ypbind* is on each workstation. The following steps are needed to access data;

1) A client requests the *ypbind* on the same local host to search the host where *ypserv* is running.

2) The client accesses the *ypserv* using the information obtained from the *ypbind*.

There are two concepts for database management of YP. One is 'YP map', and the other is 'YP domain'. YP map is implemented as DBM file. This DBM file contains a set of key-value pairs, like user-name as a key and personal information as a value. YP domain means a named set of YP maps.

[1]DBM is a database tool of UNIX.

2 The Concept for User Information Database Management in *Apostle*

2.1 Integration of Database about User Information

To use a computer system, every user is required to type his login name and password. After his successful login, his behavior is authorized against information given from database with his login name as the key. Usually, for cases of network environments, the key for database of user information is not login name but the pair of [login name, host name]. Namely, the same login name is possible among hosts connected to the same network. But IS, the Information Server of *Apostle*, unifies the login name through all the component computers and makes database independent from the hosts.

To achieve so, IS manages login names over all the component computers of *Apostle*. And it provides the same user information with login name as the key. The user information is composed of name information such as user name, group name, the path name of a home directory, and so on. For further discussion, see Chapter 3.

2.2 Hierarchical Distribution of the Roles to Manage Database

In *Apostle*, user information database is not managed by one supervisor, but by group managers. 'Group' is the same concept as defined in UNIX. This structure prevents *Apostle* from the trouble of too much concentration of control.

In UNIX, only one super user, which is a supervisor of UNIX, has the right to modify administrative data of system. This scheme is basically the feature for centralized management. But according as network is developed, the following weaknesses of this scheme become clear.

- the lack of quickness of administrative works
  As network grows larger, the amount of works of the supervisor becomes larger and he becomes overloaded. This results in the lack of quickness of administrative works.
- destabilization caused by centralized management
  Since various kinds of control judgment rush into one supervisor, his mistake may cause a serious trouble on the whole network.

To minimize the occurrence of these two troubles, we divide job roles of one supervisor into those of several group managers.

We think 'group' is supposed to be something like laboratory or project group.

A organization using IS is assumed to have the following two types of administration roles:

- the system manager, who manages the whole system:
  This manager directly manages the whole activity of *Apostle* system. He is responsible to keep *Apostle* work well without any trouble. He registers/modifies information on groups or group managers. He has no role for the details inside groups.

  To avoid the concentration of the roles for administration to one person, this system manager is defined to be separated from a super user of each component computer.

- group managers:
  A group manager registers/modifies information on members of his group. Group itself is not subject to him.

To make *Apostle* more secure, administrative works of a group manager is automatically traced by the system manager and audited.

The role of each group manager is strictly restricted to the inside of his group. For example, the maximum number of members of a group is a subject of the system manager. A group manager cannot change this number. For another example, the file creation role of a group manager is restricted by the system manager. A group manager can create a home directory for his members only under the directory fixed by the system manager for his group. A group manager cannot create files/directories under directories of other groups.

This division of labors and the restriction of works of a group manager inhibit him from unprivileged work.

3 Design of the User Information Database

3.1 The Contents of Database

There are two kinds of databases: one is a database on groups, and another is a database on users. Figure 2 shows the relationship between records of the user database and records of the group database. Figure 2 also shows that no user can belong to multiple groups.

Items of each record of databases are as follows:

- Each record of the database on groups has the following items;
  - group name: This name is used to identify a group. Information on groups is obtained with this name as the key.
  - GID: GID is the same as GID of UNIX.
  - number of allowed members: This number describes how many members the group can have.
  - information on group manager: This is a user name for the group manager of this group.
  - the list of members belonging to the group
  - miscellaneous information on group: This information includes the real name of the group and other various attributes about the group.

  These items are registered and modified by the system manager described in 2.2. A group manager cannot modify the contents of these items except for the list of members and miscellaneous information on the group.

- Each record of the database on users has the following items:
  - user name: This user name is the same as a login name of UNIX. Other information on the user is obtained with this name as the key.
  - UID: UID is the same as UID of UNIX.
  - group name for his group
  - GID for his group
  - ISRC accounting system handle: *Apostle* accounting system must be compatible with the super computer accounting system of ISRC. User's accounting information is obtained with this key.

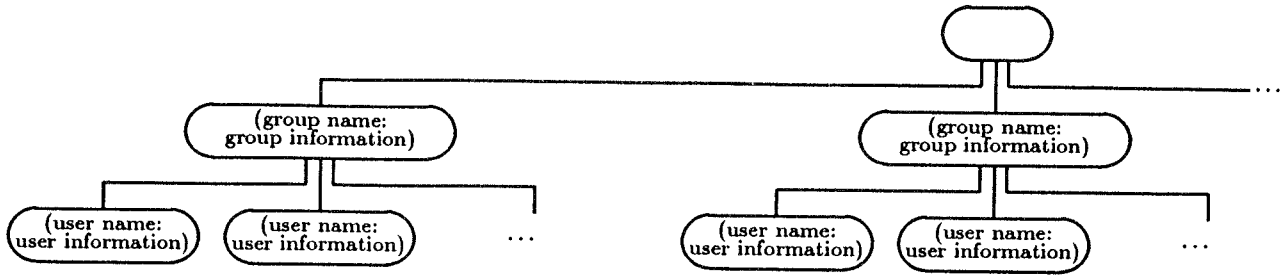


Figure 2: The Relationship between Groups and Users

- personal profile: This information includes real name, organization name, post office address, and so on.
- login password: *Apostle*, or UNIX, uses this password for authentication for users.
- home directory path name
- login shell name
- information for mail handling system: This information includes the mailbox name of the user and the location of the mailbox. The mailbox name is the one used for the sender/recipient name of E-mail. The location of mailbox is the description which component computer has his mailbox file. Further explanations are described in Appendix.

These items are registered and modified by a group manager. A user has the right to modify the contents of his items about personal profile and password.

3.2 Placing of the Replicas of the Database on each Component Computer

In *Apostle* system, the links between component computers are 64kbps synchronous lines. It is obvious that this line speed is much slower than other LAN media, such as Ethernet. Furthermore, these links have the role as the backbone of the whole network described in Figure 1. With these facts, we conclude that the frequency of data transfer between component computers should be reduced as possible.

We concluded that the frequency of references of data of a user information database is more often than that of modifications. The reason of the conclusion is as follows: The reference is occurred for the following cases; on the login of a user, on the checking of a

user's password, on the starting of his shell, and so on. On the other hand, the modification is occurred for the following cases; on the new registration of a user, on changing password, on changing information of a user's computing environment. As a result, the number of references increases in proportion to the number of login and elapsed time of user. While, the number of modifications increases in proportion to the change of user's status. Usually, the former situation is much more frequent than the latter. Therefore, the traffic between component computers for data references should be considered first.

As a result, we decide to give the replica of one same database to each component computer. In other words, this means each component computer has the same database. Since all component computers have database locally, database reference is local and traffic between component computers is not needed to refer data. On the other hand, it is assumed that appropriate mechanism is necessary upon updating data among replicas.

We employ a server-client model to manage these replicas. Each component computer has two servers which manages the replica in it; One is for data references, and the other is for database update. Since only the servers can access the replica directly, for data access, a user is necessary to start an appropriate client process. The client process requests the servers to access database.

3.3 Integrity Consideration

To keep the integrity of the user information database, several provisions like lock mechanism, priority dispatching, automatic repairing should be considered in general. These provisions have relations to so called secure update scheme.

We must take care of the locking mechanism to prevent replicas from the inconsistency due to multiple update requests at the same time. A process which wants to modify information should make a lock on the whole IS system. In our design, a client process requests all the update servers to make a lock.

We must also take care of the facilities to determine priorities of multiple update requests. Each updating server on a component computer must have its own priority different from others. A client asks the highest priority server to supply the privilege for updating. Only the highest priority server can supply the privilege for updating, and the client which gets the privilege from the highest server can make a lock on IS. Because of this lock and priority system, the inconsistency due to simultaneous multiple update requests is avoided.

To implement the procedures for lock system and priority dispatch system, *Apostle* has a secure updating subsystem described in 4.3.

To cope with the worst case like a communication failure causing an inconsistency among

replicas, we provide a mechanism to check the consistency among replicas. This check is done periodically and if an inconsistency is found, this fact is reported to the system manager. Since an inconsistency is occurred only in very complicated situations, we don't expect automatic repairing is effective. With this expectation, we don't support automatic repairing system for replicas. However, we provide several commands to aid the system manager to keep the integrity of the database.

3.4 Data References

Any client, which wants to refer data, communicates a server for data references in the local component computer. As a result of our design described in 3.2, traffic between component computer is not necessary for data references.

3.5 Data Update Algorithm

To design a distributed database system with multiple copies of the same contents, we can take the following two scheme for database update algorithm into account.

- define one of the copies as the master database: With this scheme, the target of update procedure is the master database. After the master is updated, the whole contents of a new master database or a partial information to update other copies is sent to other component computers.

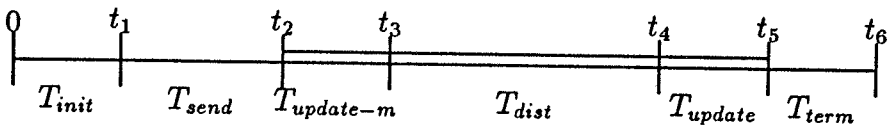
  The mechanism for this scheme is very simple. But during the time period from finishing updating of the master till finishing updating all the others, the temporary inconsistency among copies exists. And a failure during the sending update data from the master database results in the inconsistency among copies.

  Figure 3 shows a time table for this scheme. The temporary inconsistency among copies exists during the period from t_2 to t_5 for $T_{update-m}$, T_{dist} and T_{update}.

- define all replicas as symmetric components: With this scheme, a update request is sent to all replicas on component computers at once.

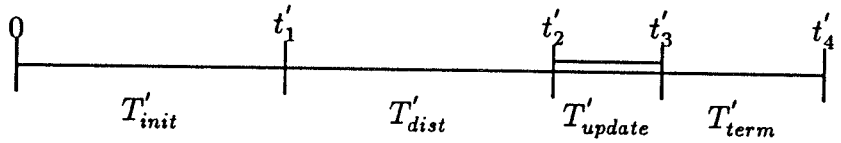
  Figure 4 shows a time table for this scheme. The temporary inconsistency among copies exists during the period from t_2' to t_3' for T'_{update}. The chance of inconsistency for this scheme is less than the former, because the temporary inconsistent period doesn't include the data transfer phase, like T_{dist} of Figure 3. Meanwhile, since it is necessary to keep status of all replicas same, the mechanism is more complex than the former.

There are the following relations between Figure 3 timings and Figure 4 timings:



T_{init}:	initial set for updating procedure
T_{send}:	sending update data to master
$T_{update-m}$:	updating the master database
T_{dist}:	distribution of update data from the master
T_{update}:	updating all the replicas
T_{term}:	termination of updating procedure
▭	critical region

Figure 3: A Time Table for Updating with the One Master Database Scheme



T'_{init}:	initial set for updating procedure
T'_{dist}:	distribution of update data
T'_{update}:	updating all the replicas
T'_{term}:	termination of updating procedure
▭	critical region

Figure 4: A Time Table for Updating with the Same Replicas Scheme

T_{init}	$<$	T'_{init}	; Since T'_{init} needs the communication with all the component computers.
T_{send}			; T_{send} is only required for Figure 3 case, since the scheme in Figure 4 has no master.
$T_{update-m}$			; $T_{update-m}$ is only required for Figure 3 case, since the scheme in Figure 4 has no master.
T_{dist}	$< \text{ or } \simeq$	T'_{dist}	; Since T'_{dist} needs the distribution to all the component computers and T_{dist} needs distribution to all except for the master.
T_{update}	$<$	T'_{update}	; Since all copies are updated for Figure 4 case and all but the master are updated for Figure 3 case.
T_{term}	$<$	T'_{term}	; since T'_{term} needs the communication with all the component computers.

We define the critical regions for updating are t_2 to t_5 in Figure 3 and t'_2 to t'_3 in Figure 4. We want to know which procedure takes more time than the other. We must compare the critical region time lengths of the two. It means whether $T_{update-m}+T_{dist}+T_{update} > T'_{update}$ or no.

In general, we know that T_{dist} is longer than $T_{update-m}$, T_{update} and T'_{update}. Since T_{dist} needs the communication between processes in different component computers, the time for T_{dist} depends on the status of other component computers and the status of links between component computers. For example, when one of remote component computers is down, a process which wants to communicate with another process in the remote component computer, must wait until the remote component come back. Since we are not sure that all the component computers work well all the time, we cannot expect the length of T_{dist}. On the other hand, since $T_{update-m}$, T_{update} and T'_{update} don't need the interaction among component computers, we can expect the length of these time period from the load of each component computer. To reduce the affection to data references from the temporary inconsistency in the critical region, the critical region time length should be shorter as possible and the critical region should not include the unexpected time like T_{dist}. For this reason, we choose the latter algorithm for updating mechanism of IS, though the total time length might be not so different for both scheme.

Table 1: Files for Implementing Databases of IS

file	YP map name	contents
mailbox description file	aliases	mailbox name
group definition file	group	group name and its GID
password definition file	passwd	user name and '/etc/passwd' information
account description file	account	usee name and his account code
profile file	private	user name and his personal profile
group management file	apostle_group	group name and administrative information for group management

4 The Implementation of IS

4.1 Database Files Managed in IS

The user database and the group database of IS described in 3.1 are logical concepts. They are implemented as the following 6 files: mailbox description file, group definition file, password definition file, account description file, profile file, group management file. These files are YP maps. In other words, they are DBM files made by DBM library of UNIX. These files are shown in Table 1.

Aliases, Group, and Passwd in Table 1 are the YP maps originally produced by SUN Microsystems. Other three YP maps are newly introduced for *Apostle*. We don't modify the form of the former three YP maps which the current YP uses. Therefore, IS is upward compatible with YP.

4.2 The Reference of Data Using *Ypserv*

The mechanism to refer data is implemented using a server process called *ypserv* which is provided by the current YP[2]. In IS, each component computer of *Apostle* has its own *ypserv*, and each *ypserv* manages the same copy of DBM file, a replica.

The steps for data reference procedure are as follows:

1. A client, a process which wants to refer any data, asks the local *ypbind* where *ypserv* is.

2. *Ypbind* returns information on *ypserv*.

3. The client communicates with *ypserv* using information from *ypbind*, and refers data.

Figure 5 shows this reference scheme.

[2]A summary of YP is introduced in 1.3.

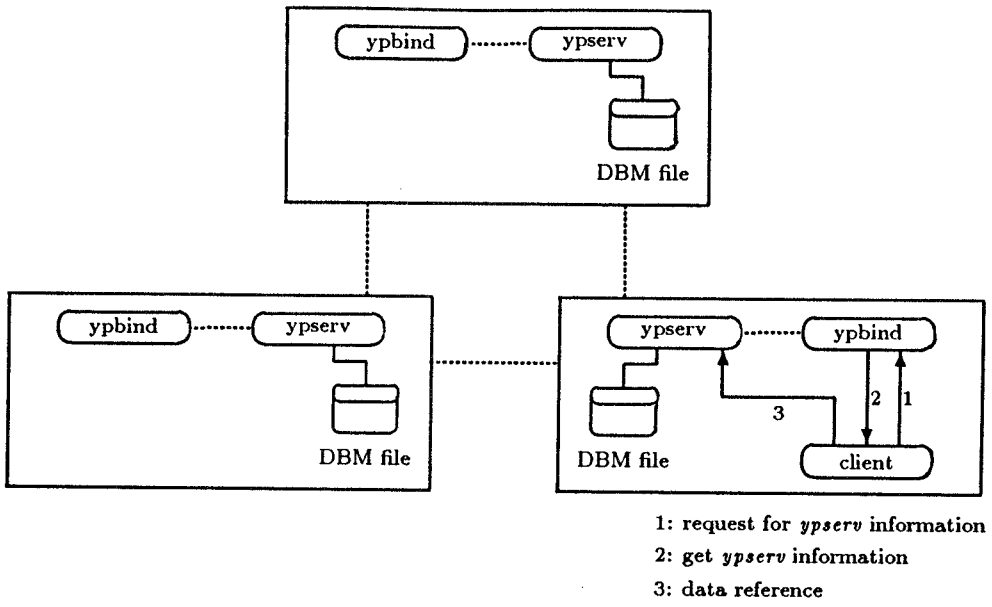


Figure 5: Data Reference Steps

In the current YP, *ypbind* uses the broadcast function to search an active *ypserv*. In IS, because each component computer has *ypserv* on it, it may be possible to eliminate *ypbind*. However, removing *ypbind* means modification of many existing application programs which refer databases managed by the current YP. For this reason, we keep *ypbind* in IS.

4.3 Data Updating Procedure

The mechanism to update data is separated from the mechanism to refer data, according to the analysis described in 3.5.

We introduce a new server process for this mechanism. It is called *ypupdated*. *Ypupdated* resides in each component computer and manages DBM files on each component computer. This mechanism assumes priority system. One of the *ypupdateds* has the highest priority among them over all the component computers. All *ypupdateds* but the highest priority one, know the way to communicate with the highest *ypupdated*. The steps for data updating procedure with lock and priority dispatch scheme of 3.3 are as follows:

1. A client process which wants to update data starts. A user who invokes this client is checked. If he has no privilege to modify data (he is not a group manager or the data is not his own), his request is rejected.

2. The client asks the local *ypupdated* process to supply where is the highest priority *ypupdated*. The local *ypupdated* returns it.

3. The client asks the highest priority *ypupdated* to supply the privilege to send update data. The *ypupdated* acknowledges it, unless the *ypupdated* has already received another request. As an acknowledge tag, the *ypupdated* sends the identifier of the on-going updating procedure to the client.

   The *ypupdated* checks whether the request is valid. A request from a user who is not permitted to update the data is rejected.

4. The client sends the identifier to all other *ypupdateds*. All the *ypupdateds* are locked for this updating procedure until the procedure finishes.

5. The client sends a transaction for updating to all the *ypupdateds*.

6. After all the data transmission are done successfully, the client requests the invocation to modify database replicas to all the *ypupdateds*. To avoid inconsistency among replicas, the procedure aborts at this step when a data transmission failure is occurred (each replica on component computer is not modified).

7. Each DBM file on component computer is modified.

8. The client asks all the *ypupdateds* to release the lock.

Figure 6 and Figure 7 show this scheme. The numbers appeared in Figure 6 and 7 correspond to the above step numbers.

4.4 Clients to Update Data

The clients to update data are implemented as several commands. Table 2 shows the repertory of commands for a requester to update data. There are 6 commands; 3 for groups and 3 for users.

Newgroup creates a new record for a group. *Newgroup* invokes *newuser* to register a group manager. *Modgroup* modifies a record for a group. *Delgroup* deletes a record for a group. *Delgroup* invokes *deluser* to delete records for the group manager and members of the group. Only the system manager is allowed to use *newgroup* and *delgroup*. The system manager and a group manager are allowed to use *modgroup*. The system manager can modify all the items of a group record. A group manager is allowed to use *modgroup* only to modify his group record. Items he can modify are described in 3.1.

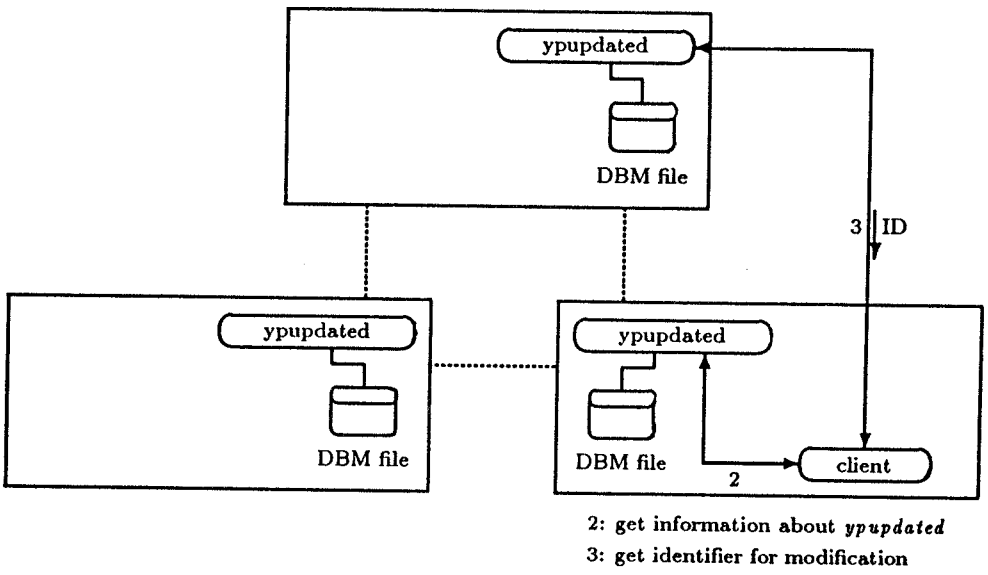


Figure 6: Data Updating Steps (1)

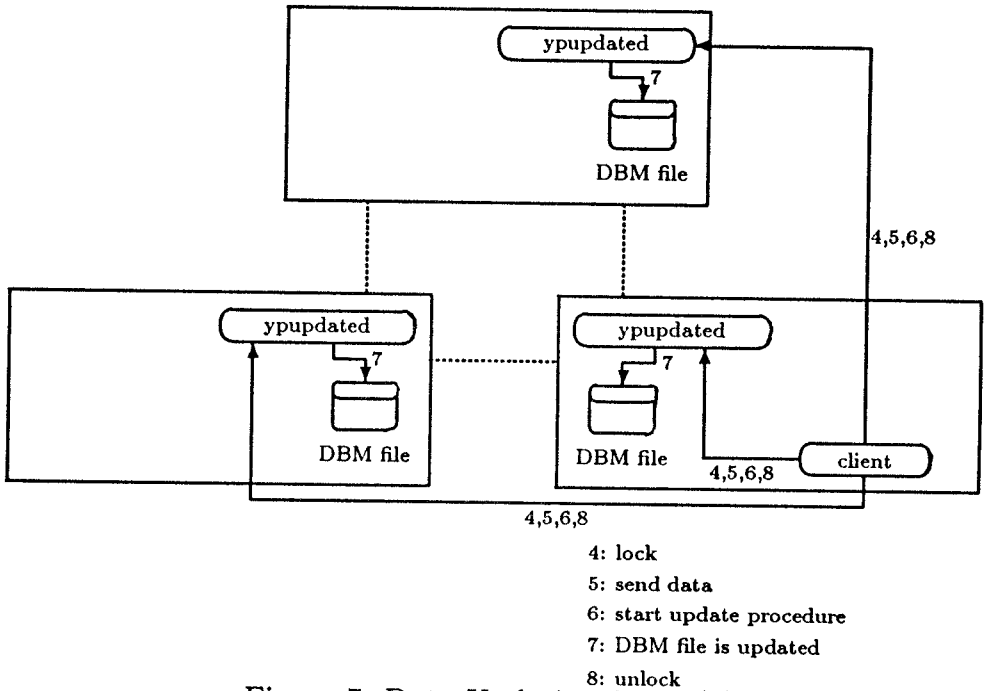


Figure 7: Data Updating Steps (2)

Table 2: Client Programs to Update Data

command name	explanation
newgroup	a client to register a new group
modgroup	a client to modify information on a group
delgroup	a client to delete a group information
newuser	a client to register a new user
moduser	a client to modify information on a user
deluser	a client to delete a user information

Newuser creates a new record for a user. *Moduser* modifies a record for a user. *Deluser* deletes a record for a user. A group manager is allowed to use *newuser*, *moduser* and *deluser*. A group manager can modify all the items of a user record using *moduser*. A user is allowed to use *moduser* only to modify his own record. Items which a user can modify, are described in 3.1.

For registrations using *newgroup* or *newuser*, a requester must give a list of items described in 3.1 to the command. All the commands confirm the registration, modification or deletion by prompting a requester to acknowledge.

4.5 Provisions to Support Group Manager for Setting Up A User Environment

To register a new user, a group manager will use *newuser* command. His home directory should be created at the same time of this registration. Since a group manager is not a super user though he has the role for this registration, he has no rights to create a new directory and to change the ownership of the directory with ordinary mechanisms of UNIX. For the distribution of the roles of super user's administrative works, a mechanism to give a group manager a part of rights of the super user is necessary. For this reason, we support this mechanism on *Apostle*.

To support a group manager to set up environment for a new user, we introduce a server process, called '*rsetupd*.' The *rsetupd* is running on each component computer.

A client which wants to create a new home directory, asks *rsetupd* on the target component computer to set up a new user's environment. The target component computer on which the home directory is to be created, is determined by the path name information in the user information database. (Home directory path name information is brought from the home directory field of the group manager's record of user database at first.) Setting up a user environment has the following steps:

1. The target *rsetupd* checks the privilege of the request.

2. The client sends *rsetupd* the three types of information on a new user to be created, namely the path name of home directory, UID, and GID.

3. *Rsetupd* creates the new home directory.

4. *Rsetupd* puts several startup-files, such as '.cshrc' or '.login', under the home directory.

5. *Rsetupd* gives the ownership of files/directories to the new user.

5 Conclusions

5.1 Summary

As a summary, IS has the following features:

- The uniform database is managed on a network.
- On the reference of data, data transfer is not necessary between component computers. Each component computer has a replica of one master database. Therefore, any reference of data is processed locally inside the component computer. This feature is important for the case of *Apostle*, because the links between component computers in *Apostle* are not assumed to be high bandwidth.
- The inconsistency among replicas of database cannot occur with usual situation. Database is managed in the distributed scheme. Since each replica of database is updated synchronously, the inconsistency among replica must not be occurred. However, on the modification of data, all component computers with a server to manage database must be alive.
- The essential mechanisms of IS and the structure of database don't depend on the operating system or the administrative concepts on each component computer.
  The operating system of each component computer can interpret and use the contents of IS database. It means that several different machines can compose a distributed system with IS described in this paper.

  Under the current implementation for *Apostle*, each component computer is happened to be the same. As the consequence, all the mechanisms on each component computer are happened to be the same.
- The concept of groups in UNIX is re-defined as groups for the administration.
  Because of this extended concept of groups, users can be managed under groups. Roles of administration of super-user is distributed into several group managers. This feature is important for UNIX system with many users.

5.2 Security Considerations

Security problems are important parts of distributed systems. Security problems in IS are classified into several topics. We try to solve these problems, but the current implementation is not enough secure yet.

1) Security on Database Files of IS

Database files of IS are implemented as YP maps. These maps are the DBM files of UNIX. Access rights for these DBM files are set by the super user. Usually, only the super user can read/write these DBM files, and others cannot read/write them. Since access rights for the DBM files are subjects to a super user's administrative works not IS but the super user of a component computer should take care of the DBM files security.

2) Security on References for Database

On UNIX, any user can read a user information file, like '/etc/passwd'. This means any user can refer information on other users. In YP provided by SUN, this scheme is kept. Any user can get information on others in database managed by YP. Since the mechanisms for references of data of IS are the same as the current YP, any user can get information of others from database of IS.

This is one of the weaknesses of UNIX. We can choose to design a more secure operating system or to implement functional interface to read/write such important data. The current *Apostle* uses the traditional UNIX interface or YP interface. A more secure mechanism for references is future problem.

3) Security on Modifications for Database

On modifications of data, both a server and a clients for updating procedure check the rights for a requester to modify data. A client communicates with a server using Interprocess Communication (IPC) mechanism of UNIX. Since the current implementation of IPC mechanism of UNIX is not so secure (see next paragraph), a false statement from vicious users cannot be rejected. This problem will be solved by prompting a user to type password or implementing the mechanism of call-back from the client.

4) Other topics on Security problems

Many of common security problems on network come from the design of the current implementation of IPC mechanism of UNIX. For example, since any user can use IPC mechanism of UNIX, he can call any server. However, the server cannot know who really calls the server.

All the IPC mechanisms must be newly implemented to resolve this problem. However, it is out of scope of *Apostle*.

5.3 Further Consideration

The IS is designed for *Apostle*. Since the IS is implemented as portable as possible, it may be workable on a different distributed system.

One of our goals is the integration with the whole system of the Trinity Initiative. An integrated administration system in the ISRC network is one of the goals of the Trinity Initiative. For this purpose, database management system for user information not only for *Apostle* but also for whole the network will be necessary. We are now designing this total database management system. IS should cooperate with this total database system. The affects on IS to do so is a future problem.

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) and the office staffs of ISRC for their continuous encouragements.

References

[CSRL88] Masayuki Ida and Keisuke Tanaka, *'The Apostle System Overview'*, CSRL Technical Report #88-001, Aug. 1988.

[SUN86] Sun Microsystems Inc. *'The Yellow Pages Protocol Specification,'* Feb. 1986.

[Allman83] E. Allman, *Sendmail – An Interconnecting Mail Rerouter, Version 4.2*, UNIX Programmer's Mannual, 4.2 Berkeley Software Distribution, 2c, Virtual VAX-11 Version, Univ. of California, Berkeley, August 1983.

Appendix: Considerations about E-mail addresses

We employ '*sendmail*[Allman83]' as our E-mail system for *Apostle*. E-mail system needs the two types of information; one is '*name*' and the other is '*address*'.

Name means string used to identify the mailbox. *Name* appears in a message header of E-mail. And we can use the *name* as the recipient/sender name of message, too.

On the other hand, *address* describes where the mailbox is located. In other words, the *address* describes the host name to which a message is delivered and the user name of the host. Using these two types of information, the host name and the user name, the mailbox is identified.

In Apostle, a user can use the name of mailbox which is independent from the host name of the component computer he use. For example, on one component computer called 'aoyama', the name of mailbox for 'user' is 'user@aoyama.cc.aoyama.junet'. In this mailbox name, 'aoyama' means the host name, and 'cc.aoyama.junet' for the domain name in which the host name is defined. Since this name depends on the host name on which the mailbox is located, we defined this type name as the *address*.

In *Apostle*, a user can have the name of mailbox which is different from the *address* of the mailbox. For example, 'user' can have 'user@cc.aoyama.junet' for his mailbox name. This name is independent from the host on which his mailbox is located. This type of name is used the recipient name and for the sender name too.

A) The Recipient Name:

Messages to 'user@cc.aoyama.junet' are sent to an appropriate component computer on which his mailbox is located. This feature is implemented using an aliases database of sendmail. This aliases database is the mailbox description file described in 4.1. IS has 'aliases' YP map as mailbox description file. This YP map includes the following list.

```
keisuke: keisuke@aoyama.cc.aoyama.junet
user: user@aoyama.cc.aoyama.junet
```

This map is shared by component computers using IS. Using this shared map, E-mail system in *Apostle* converts the name of mailbox, like 'user@cc.aoyama.junet', into the address of mailbox, like 'user@aoyama.cc.aoyama.junet'.

B) The Sender Name:

The sender name is the name of user who writes the message. The name is placed on 'From:' field in the message header. This name is generated by the configuration rules of

sendmail. Configuration rules are written in the file called 'sendmail.cf'. We implemented a tool to aid to make this rules file, sendmail.cf. This tool is called '*mailconf*. *Mailconf* creates the configuration file for sendmail.

We added the following declaration on a database for mailconf.

```
$control: genericfrom
```

With this declaration, *mailconf* creates the appropriate configuration file for sendmail. This file has the functionality to control the headers of messages. Since the configuration file contains the direction of `genericfrom`, 'From:' header of a message has no host name on which the message is being created.