MASTER IN COMPUTER SCIENCE

Decentralized repository for data backup management

Derck, Didier

Award date:
1996

Awarding institution:
University of Namur

FACULTÉS UNIVERSITAIRES NOTRE-DAME DE LA PAIX, NAMUR
INSTITUT D'INFORMATIQUE
RUE GRANDGAGNAGE, 21, B-5000 NAMUR (BELGIUM)

DECENTRALIZED REPOSITORY FOR DATA BACKUP MANAGEMENT

Didier Derck

Promoteur : Jean Ramaekers

Mémoire présenté en vue de l'obtention
du grade de Licencié et Maître en Informatique

Année Académique 1995-1996

LBS 6843632

307325

CONDENSÉ

Les environnements distribués ont les avantages que la charge de travail est distribuée sur plusieurs de sites sur le réseau. Cependant, avec l'émergence des Interfaces Graphiques pour les utilisateur, les temps de réponse doivent devenir très court. Cela peut conduire à des problèmes dans les situations où plusieurs stations de travail doivent accéder à un site pour obtenir leurs informations. Dans ces cas spéciaux, les développeurs d'applications peuvent être amenés à changer leur vision pour la mise en oeuvre de leurs applications.
Il peut être obligatoire de faire un tel changement pour répondre aux problèmes spécifiques de performance.
Ce travail présente une étude concernant la décentralisation d'un répertoire dans un environnement distribué particulier : un système distribué de stockage hiérarchique. Cette décentralisation est désirable suite aux mauvaises performances de l'application décentralisée lorsqu'elle doit accéder au répertoire central.

ABSTRACT

The distributed environments have the advantages that they distribute the work load on several sites on the network. However, with the emergence of the Graphical User Interfaces, the response time must become very short. It can lead to problems in the situations where several stations must access one site to get their information. In these special cases, the applications developers can be brought to change their mind about the implementation of their applications. It can be mandatory to do such a change to answer to specific performance problems.
This work presents a study about the decentralization of a repository in a particular distributed environment : a distributed hierarchical storage system. This decentralization is desirable due to the bad performance of the decentralized application when accessing the central repository.

ACKNOWLEDGMENTS

I wish to thank Professor Jean Ramaekers for its precious advices.
I would like to thank Benoit Hucq from Siemens-Nixdorf Software for providing the subject of this thesis. I want to thank Fabian Libion for his help, advices and corrections and also all the people from the RD34-RD35 teams from Siemens-Nixdorf Software for their help, explanations and advices about HSMS and ARCHIVE products.

Finally, I would like to thank Luc Halbardier for his help concerning the English.

TABLE OF CONTENTS

1. INTRODUCTION

The object of this thesis is to study the decentralization possibilities of a repository in a particular distributed environment and to propose an implementation model that solves the problem of the present location of the repository.

This work is divided as follow : in the two first chapters, the distributed environment is presented and the location problem is explained.
In the third chapter, the different theoretical decentralization models found in the literature are presented and briefly evaluated in a general situation.
The fourth chapter contains the evaluation of these models for the concerned distributed environment. At the end of this chapter, three possible solutions are kept for further evaluation. The next chapter contains the study of all the required procedures for the three kept decentralization possibilities.
The sixth chapter contains a solution proposal and the details of the distributed environment architecture together with the required changes associated with the solution.
The last chapter gathers the possible evolution that can be made after the implementation of the decentralization.

2. PRESENTATION OF HSMS

2.1 INTRODUCTION

HSMS is a BS2000 software product -Siemens mainframe Operating Systems- which supports data management on external storage devices in a BS2000 system. HSMS is an acronym for Hierarchical Storage Management System. It provides several useful functions as backup, archival and migration of files.

To cope with the increasing part of the UNIX systems in the IT organization and to complete the offered services, a complementary software has been developed : it is HSMS-CL. This new product is a client version of the HSMS product and allows the UNIX administrators to backup and archive their files with HSMS (the file migration in UNIX is not yet implemented, but the project is under way).

HSMS uses the ARCHIVE product (in BS2000 environment)for the actual saving of the managed files to or from the storage levels.

2.2 CONCEPTS

As HSMS uses several concepts that don't exist in the other products, a brief description is required for a good understanding of the following chapters. However, the most important concepts will be explained in full details as they appear in the next chapters.

2.2.1 Hierarchical Storage

As HSMS is a hierarchical storage system, it respects the HSM norms for the storage space. Globally, those norms are set to organize the storage space in several levels, each one composed by a certain type of media.

In HSMS, the storage space is divided in three distinct levels (Table 1) :

1. Storage level S0

The storage level S0 is the normal processing level. It is the one the users can access on-line. This level is managed directly by the file system (DMS, Data Management System, for the BS2000 and UFS, UNIX File System, for the UNIX). This level is composed by disks. As it is the online level, it must be the faster one. It is thus composed by magnetic disks with fast access time.

2. Storage level S1

The storage level S1 is the online background level. The data stored on this level are managed by the HSMS program. It is composed by disks too, but these disks are generally slower and bigger than those ones used at the level S0.

3. Storage level S2

The storage level S2 is the off-line background level. It consists of magnetic tapes or magnetic tapes cartridges. It is thus the slower storage level of HSMS.

All the data at this level are managed by HSMS.

Storage level	User access	Media	Access	Access time
S0, processing level	direct	disks	online	very short
S1, background level	via HSMS	disks	online	short
S2 background level	via HSMS	magnetic tape or cartridges	off-line	long

Table 1 : Storage Levels in HSMS

2.2.2 HSMS Archive concepts

The archive is the basic HSMS management unit. HSMS stores and manages all data saved by either backup, archival or migration in archives. HSMS distinguishes five types of archives, one archive for each of the basics functions it offers : backup, archival and migration in BS2000, and backup and archival in UNIX (the migration in UNIX is not yet implemented). All the basic HSMS functions can't be executed until the appropriate archive has been created.

The user creating the archive is its owner. It is important to note this feature because only the owner of an archive can manage it. Normal users can only obtain information about the data they own or if the owner of the archive containing those data has declared this archive public, i.e. accessible by all the users.

Each HSMS archive consists of (see Figure 1):

- the archive definition containing the archive's attributes
- the associated archive repository
- the save files containing the saved data

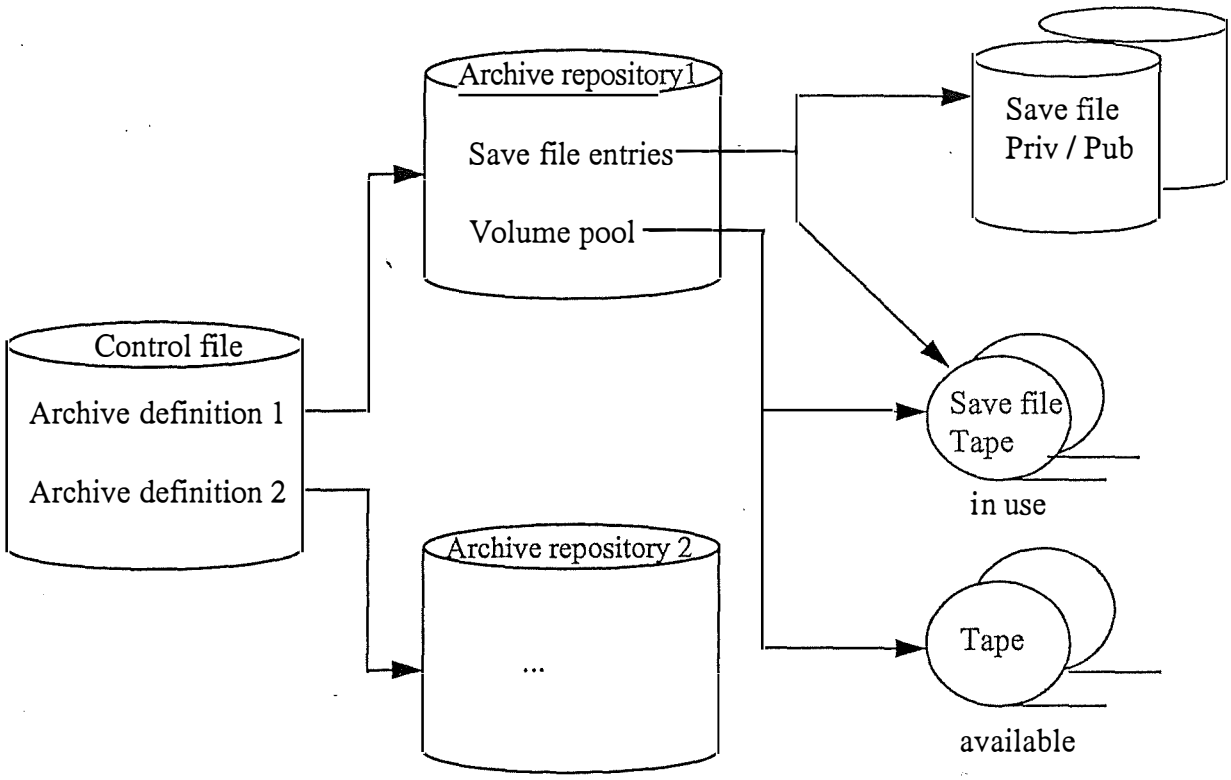


Figure 1 : Structure of an HSMS archive

Archive definition

The archive definition contains the following information :

- the type of the archive
- the owner of the archive
- the attributes of the archive (as the name of the associated repository). Among the attributes of the archive, there is one field that describes if the archive is public or private.

All the archive definitions are stored in the HSMS control file.

Archive repository

The archive repository contains the information about the saved data managed in the archive. These information are :

- the file names and attributes
- the save files data
- the save versions data
- the occupied and free save volumes

The repository is managed by the ARCHIVE product.

Save file

The save file is a cataloged BS2000 disk or tape file. A save file on tape consists of a number of volumes having the same owner and the same Save File Identifier (SFID). A save file can contain one or more save versions(see Figure 2).

Save version

A save version contains all files saved by a request plus the needed metadata to allow a restore or a recall of the files. A save version is identified by its Save Version Identifier (SVID). A file can be saved only once in a save version.

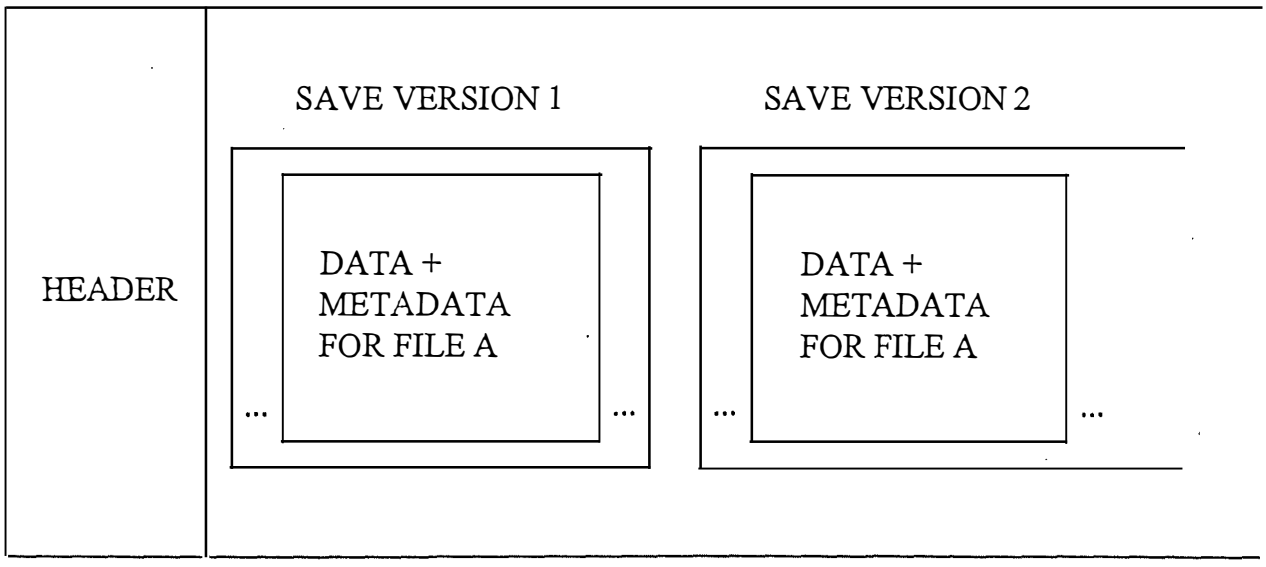


Figure 2 : Structure of a save file with several save versions

A file can appear many times in a save file but only if different save versions were written in the save file.

2.3 HSMS BASIC FUNCTIONS

2.3.1 Common properties to the basic functions

The HSMS basic functions provide the following services :

- Tapes are updated for independent archiving jobs as default. This means that tapes can be used in a more productive manner.
- A logical expiration date (specific to the job) can be allocated for archived, backed up or migrated files, independent of the tape's physical retention period. When changing archive tapes, those tapes that have not yet reached their expiration date can be specifically taken over.
- Archiving and reactivation job are collected in a job file. Tapes can be accessed at times specified by the system administrator. This enables the computer center to better plan tape processing times.
- The data can be compressed when being transferred to the HSMS storage level and are automatically decompressed when accessed.

2.3.2 Migration

This function makes possible to migrate user files from the processing level S0 to S1 or S2 as well as to recall these files to S0 for processing. This effectively reduces the danger of saturating the disk storage. If the disk saturation limit or the user allocations are exceeded, the migration function can be started and the inactive data migrated. The number of days since the last access, the minimum file size and the file fragmentation can be specified as criterion for selecting files candidates for migration.

Migrated files are automatically recalled before processing during file opening or reservation. Migration and recall can also be executed via instructions. The normal user has access to these statements.

Some files are normally excepted from migration (e.g. opened files, temporary files or files that need to be repaired). The user can also set migration lock.

When a file is migrated, the original catalog entry remains on S0 and is given a 'migrated' symbol. This function only exists in BS2000.

2.3.3 Backup

The concept of the backup as a system service must be separated from the concept of 'long-term archiving'. To a great extent, HSMS includes the ARCHIVE functionality, in addition to new functions. The files on the processing level S0 and the migration levels S1 and S2 can be saved.

The system backup is carried out by the system or the HSMS administrator. HSMS manages backup data resources automatically and safely using system backup archives.

The principal functionality of ARCHIVE is, except the complete backup and the archival, the differential saving (incremental backup). For the differential saving, the principle is to compare the catalog entry information and the repository information and to save only files which have been changed since the last saving. Run time and memory space are then economized. Despite of that differential saving the possibility to reconstruct the complete data state as existing at the last saving always exists.

The backup function permits backup in the backup level, either on disk, tape or cartridge. If operation is sometimes unmanned, the backup data can be temporarily stored on disk so that it can be transferred to tape at a later point in time. It is even possible to mix storage forms, i.e. to store the differential backup on disk and store the complete backup on tape.

Through the archive repositories , there are many options for selecting files for the purpose of saving back or reconstruction.

Via HSMS, it is also possible to make complete backup by executing only an incremental backup (full from incremental). This function consists to run an incremental backup, then with the last differential savings and the last full backup, to reconstruct a full backup on tape without needing to access the disks. All the processing is made on the background storage levels.

2.3.4 Long-term archiving (archival)

The HSMS archiving functions , as an end user function, provides optimum support for long-term archiving. It also implements management of all data on the archiving level. HSMS offers the option of adding extra user information with archiving jobs (names/descriptors and text for archive versions), in addition to improved archive information functions. Another option with HSMS is defining archives and setting characteristics such default values for compression, device types, etc.

2.3.5 Copying backup files

HSMS allows save files and save versions they contain to be copied within an archive or from one archive to another.
Copying can be used to :

- swap save files from S1 to S2
- copy save files as a precaution against data loss
- reorganize a migration archive

2.4 HSMS-CL

HSMS-CL is a software running on UNIX systems. It allows to backup and archive UNIX files through HSMS on the BS2000. In order to do such operations, it requires the presence of Network File System (NFS) in server mode on the workstations and in client mode on the BS2000. It also requires the presence of the HSMS-SV product on the server (see Figure 3).

With this requirements fulfilled, and with the presence of the HSMS server software on the BS2000, it is possible to backup and archive the workstations files onto the BS2000. It necessitates too the declaration of dedicated archives for these purposes.

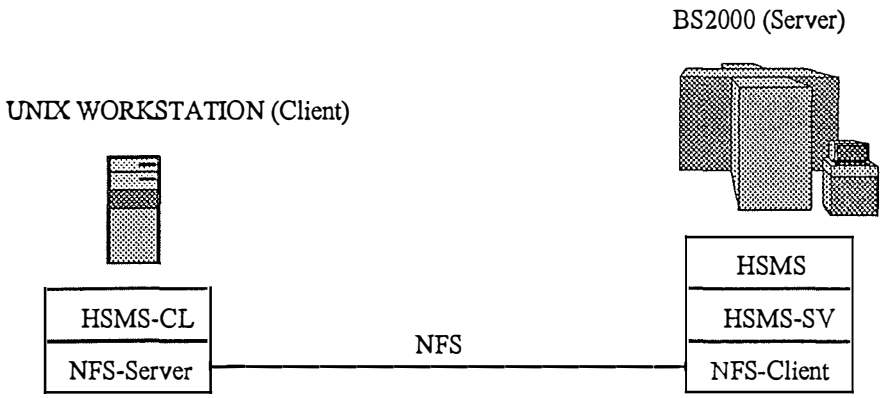


Figure 3: HSMS-CL configuration

The HSMS-CL software comes under three distinct forms : a Graphical User Interface (GUI), a Command Line Interface (CLI) and an Application Interface (API).

It is also possible to backup and archive UNIX files without having the HSMS-CL on the workstations. In this case, only the presence of HSMS-SV and NFS is required. The workstation is called a passive client when it does not have HSMS-CL and active client when it does.

3. PROBLEM DEFINITION

3.1 WHY A DECENTRALIZED REPOSITORY ?

The HSMS architecture is a client-server architecture in which there is only one server. All the clients access to this server where the repository is located (Figure 4).

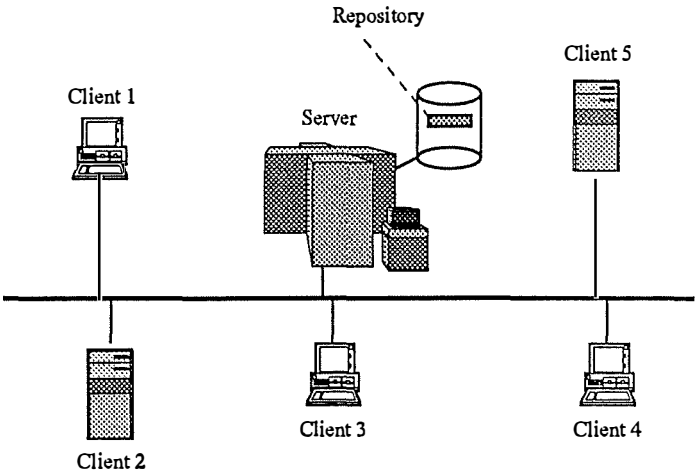


Figure 4 : HSMS Client-server Architecture

The disadvantages of that centralized location are that in case of some operations initiated from the clients (especially query operations), there is a waiting time and a traffic generated on the network. This delay is caused by the sending out of a request for information to the server, the transmission time, the information retrieval on the server and the transmission time for the answer generated by the server. As the repository is a quite big file, the information retrieval on the server can thus take a certain time. The generated traffic is composed of the request from the client and the answer generated by the server. On a heavy loaded network, the transmission times of these datagrams can be quite important. And as the client program can be a Graphical User Interface (GUI), a long waiting time is not acceptable.

The decentralization of the repository can bring a solution to those problems. It can reduce the delay and lower the traffic on the network for query operations asked by the clients. As the information is available locally, the delay can be reduced to the time needed for the retrieval of the information in a local file. The traffic generated on the network can be reduced to the information exchange between the server and the clients that is required to keep the decentralized repositories up to date.

This exchange generates a certain amount of traffic, but it can be made during periods of the day where the overall traffic on the network is reduced (during the night for example). It is important to note that this traffic can be as much important as the traffic generated by the query operations.

Several interrogations arise with the decentralization :

- what kind of decentralization should be chosen ?
- which information should be decentralized ?
- when the updates to the decentralized repositories should be made and how ?

Several problems arise too :

- what does happen when an access is made to a non-updated repository ?
- some information needed to access the repository are located in another file, the control file. Must this second file be decentralized too ?

3.2 COMMANDS USING THE REPOSITORY

Here comes a brief study of the commands using the repository. This study is here to differentiate the accesses to the repository, where they are initiated,...

There are two types of commands : the commands given by the active clients and the commands given by the server. All these commands access the control file which contains the associations between the names of the archives and the names of the repositories, and other information related to the archives.
The client commands are : bsarch, bsback, bsrest, bsshow. The others are the server commands.

The commands are subdivided in two categories : the commands related to the node files and those related to the archive.

3.2.1 Archive related commands

Create-archive

This command is used to create an archive for backup or archival. It is also used to define the name of the repository associated with the archive.
It initializes the repository.

Modify-archive

This command is used to modify an archive repository, i.e. modifying information in a save file, deleting save files and managing a volume pool.

Delete-archive

This command is used to delete an archive definition. It does not access to the repository but only to the control file.

3.2.2 Node files related commands

Archive-node-files, bsarch

These commands are used to archive node files, they access the repository to update it, also to consult it in order to continue a save file,...

Backup-node-files, bsback

These commands are used to backup node files, they access the repository to update it, and also to consult it as the archival commands

Restore-node-files, bsrest

These commands are used to restore node files from an archive (either archival or backup archive). They access the repository to retrieve information concerning the files to be restored.

Show-archive, bsshow

These commands are used to see the contents of an archive. They display information about the save files, the save versions, the node files or the volumes contained in the specified archive. They consult the repository in order to retrieve this information.

Select-node-files

This command is used to select files from an archive. It consults the repository to find which files are corresponding to the selection criterion specified by the user.
The result can be used directly by another command or stored in a file.
This file can later be used with the restore-node-files or the backup-node-files commands in order to specify the files to be restored or backed up without giving all the file names explicitly.

Copy-node-save-files

This command is used to copy a node save file and the save versions of a predefined archive, either within the same archive or into another archive.
It accesses the repository in order to find the information over the save file to copy and in order to update the information about the new save file.

The repository is accessed as much from the workstations as from the server. This implies that the decentralization of the repository must be made so that the performance of the server are kept optimal and the performance of the workstations are increased. It is useless to improve the performance of the workstations if the server's performance is degraded. It is then necessary to find a solution giving the best gains of performance for the workstations when accessing the repository without losing the present performance of the server for backup and restore operations.

4. THEORETICAL DISTRIBUTION MODELS

4.1 INTRODUCTION

There are several possibilities for the decentralization of a repository. Each possibility has its advantages and drawbacks. The repository can be :

- replicated
- partitioned
- partitioned and centralized
- partly partitioned and partly replicated
- any other combinations

Whatever the possibility, it must have the following properties :

- distribution transparency
- site autonomy
- efficiency
- high reliability/availability
- security/access control

4.1.1 Distribution transparency

Regardless of exactly how the repository information is distributed, the interface and overall operations of the repository must provide a single, consistent interface as if the repository resides on the user's local host. There are two aspects to distribution transparency. First, when a user is concerned, the "simple" operations on the repository must appear as if they took place locally, even if remote sites are contacted. The definition of "simple" operations may vary : there are operations which explicitly require the user's awareness of access to remote sites. Second, the consistency of replicated or partitioned data is important; multiple accesses to the repository should not result in inconsistent or unexpected results dependent on the distribution.

4.1.2 Site autonomy

This property is necessary to enable any site to access its local data without being blocked by any other site.

4.1.3 Efficiency

All the information must be stored close to its normal point of use, reducing both response times and communications costs.

4.1.4 High reliability/availability

It is imperative that we can rely on the application, i.e. not easily crashed by the applications or systems failures, and available, i.e. operational at a local site or portion of the network even if other sites crash or the network becomes partitioned.

4.1.5 Security/access control

Although each site has its own security mechanisms, it is necessary to have some security/access controls for the repository itself. The user should not be able to delete the repository's file, modify it directly without using the ad hoc program, consult information to which he shouldn't have access.

4.2 REPLICATED REPOSITORY

In this situation, two or more sites are holding a copy of the repository in its entirety. As illustrated in Figure 5, there is an extreme solution in which the repository is stored in each site of the network.

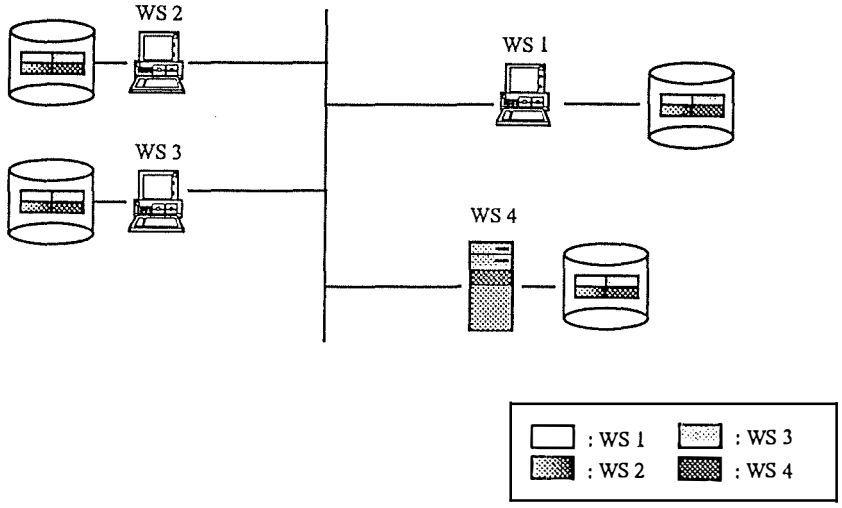


Figure 5 : Replicated repository

This solution (full replication) solves all the problem of access time and direct traffic (by direct traffic, we mean traffic generated by the requests) : the only traffic that is necessary is the one needed to keep the information up to date. It offers a very high reliability : as many sites are holding a copy of the repository, in case of a failure on a site, it is easy to find back the information and there is no lost of information.

In the case of a non-full replication, the load is distributed to several sites in place of one, thus accelerating processing of the requests. But this distribution model implies the propagation to all the copies on the network. It generates thus a lot of traffic. Moreover, special provisions must be made in case some copies are unavailable when the update is made (unavailability caused by a network failure or by a site closure).

This solution requires a lot of disk space as each site holds the total repository. Moreover, it implies to set up mechanisms to avoid data access concurrency.
But it respects all the properties required by the distribution. The distribution transparency is assured as all the operations are processed locally. The site autonomy is full as no other sites must access to this repository. The availability is also assured as no access to the network are required, and the efficiency is maximum as the information cannot be stored closer than its normal point of use.

4.3 PARTITIONED REPOSITORY

In this situation, each site maintains its own repository containing the local information (see Figure 6). There is no global repository containing all the data, but the union of all the disjoint local repositories forms the total repository.

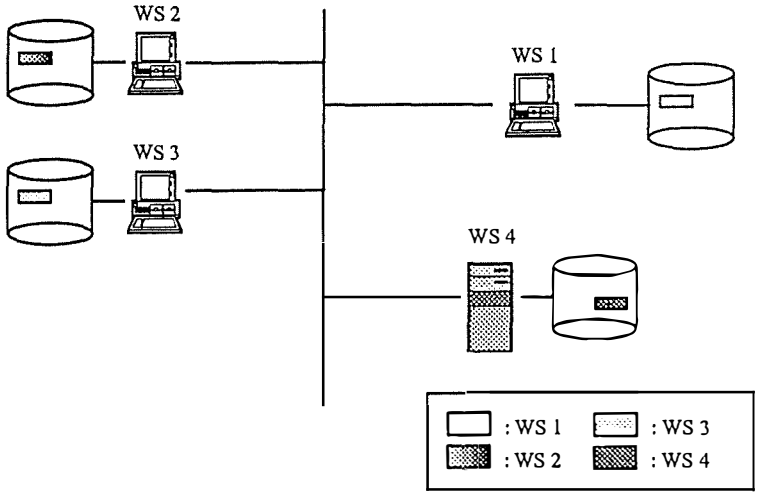


Figure 6 : Partitioned repository

This requires less storage space, accelerates the information retrieval as the file is smaller, but offers less security as there is only one copy of the data and the update procedure is much easier than in any other case as it implies only the local site.

Requests concerning non local information must be broadcasted to all the sites in order to locate the site holding the required repository entries.

All the properties of the distribution are not fulfilled. The distribution transparency is nearly perfect, as long as there is no access to information located on other sites. In this case, there is a delay and it goes against the property of site autonomy. One site can be blocked by the accesses coming from the other. The property of availability is not respected in this case, as a network failure can block access to information. The efficiency is assured as the information is stored close to its normal point of use if the partition has been correctly made.

4.4 PARTITIONED AND CENTRALIZED REPOSITORY

In this situation, as shown in Figure 7, each site is holding its own repository containing the local information and a central site maintains an unified copy of all those local repositories.

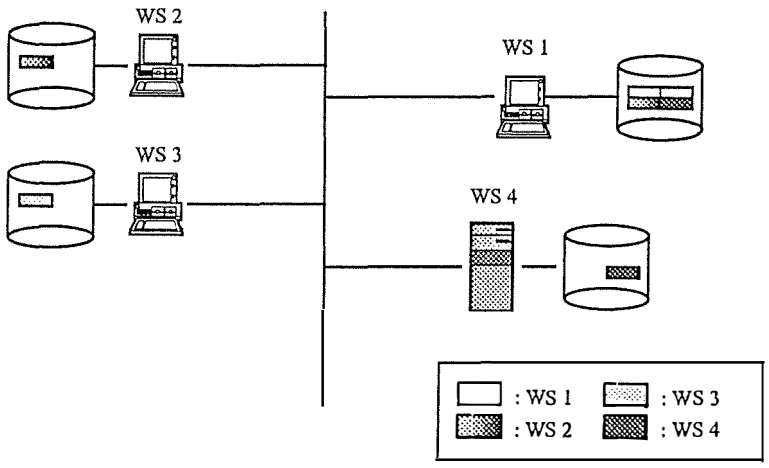


Figure 7 : Partitioned and centralized repository

It has the same advantages as the partitioned case except that this requires more disk space to store the repositories and that requests for non-local information are directed to only one site and not broadcasted on the network. It has the great advantage that as there is more than one copy of the information, there is a greater security.

Updates are directed only to the local and the central site, thus reducing the complexity of this procedure. It also needs a concurrency control between the decentralized site and the central one.

The properties are fulfilled as in the partitioned case, except that the site autonomy is more respected as only one central site is disturbed by the non-local requests. The reliability and availability is respected only if the server remains accessible in case of network failure.

4.5 PARTLY PARTITIONED & REPLICATED REPOSITORY

As shown in Figure 8, with this solution, some information contained in the repository are replicated and some are partitioned.

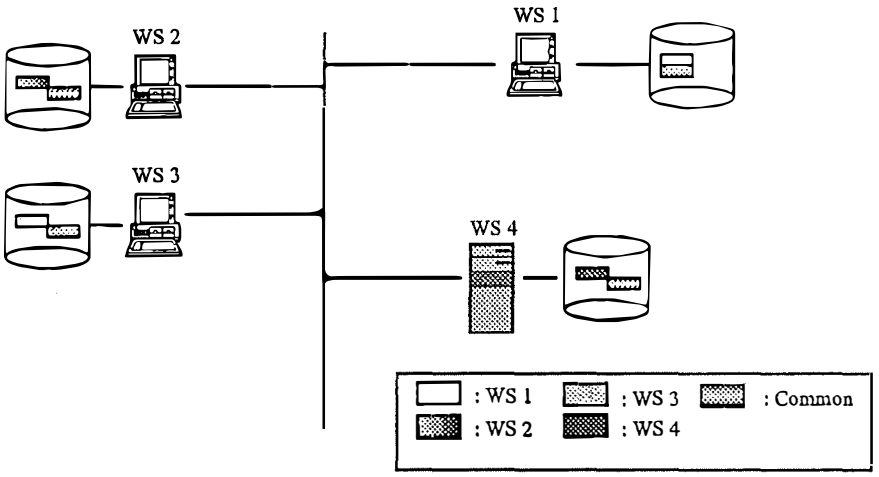


Figure 8 : Partly partitioned and partly replicated repository

The replicated parts generally concern information that is needed by several sites, but it can be information that is critical and that can't be lost. By placing this information on each site, the access time are reduced the update procedure is more complex.

It combines the fast access time to all the information and an average need of disk space but requires, as in the replicated case, a special provision for the sites that are unavailable at the update time of the replicated part. It also requires a concurrency control for these parts.

Concerning the distribution properties, the distribution transparency and the site autonomy are fully satisfied as nearly all the required information are located on the site. It also offers a greater reliability and availability as the bigger part of the requests can be processed locally, without accessing the network. The efficiency is also optimal if the replication and the partition are correctly made.

4.6 OTHER COMBINATIONS

These other combinations imply that each site holds a part of the total repository, generally the local part and the ones of the related sites.(see Figure 9).

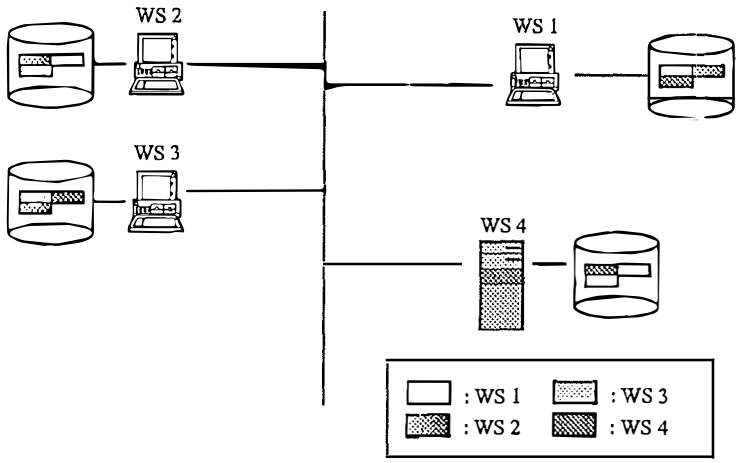


Figure 9 : Other combination

It offers a better security than the other cases as there is more than one copy of the information and easier access to non-local information as there is a copy of those on the site. But this requires more storage space and the update procedure is more complicated.

The distribution properties are more or less respected following the access made to the information. If the accesses are made only to the local information, the properties are satisfied, if it is not the case, they are not.

5. EVALUATION FOR HSMS

5.1 INTRODUCTION

The evaluation for HSMS is divided in three parts : the study of the structure of the present repository, the evaluation of the decentralization possibilities for the repository, the evaluation for the control file.

The structure of the repository is placed here to highlight some important evaluation criterion for the decentralization possibilities. Following these criterion, the theoretical models are evaluated for the repository and the control file. At the end of this chapter, a brief summary is given with the possible solutions for HSMS.

5.2 PRESENT STRUCTURE OF THE REPOSITORY

The repository is an ISAM file based on an access key of 155 characters. It is a BS2000 file located on the server . The repository contains information about the saved files. This information is organized in records. There are five different types of records used to store information. These records are completed by two special records used to mark the begin and the end of the repository. They are also used to verify the state of the repository : at the opening, HSMS verifies if they are present, if not, it stops the access and generates an error. These two records are the Low key record and the High key record.

The five record types used to store information about the saved files are :

- the file record or F-record
- the save file record or R-record
- the save version record or S-record
- the tape record or T-record
- the save version extent record or X-record

5.2.1 The F-record

This type of record is used to store information about the node files that have been archived or backed-up. The identifier of the record is formed by the letter "F" followed by the hostname, the user number and the filename. The filename is composed with the full pathname and the filename itself and is restricted to 99 positions. In the case of a pathname longer than 99 characters, it is truncated and the rest is stored after the identifier in the data part of the record (up to 925 positions, for a total of 1024 characters). The identifier is followed by the information about the file itself (access bits, access dates,...), and information about the save

sessions. For each save session concerning the file, you find the identifier of the save version (SVID) and the identifiers of the volumes (VSN) that contain the file (see Figure 10).

F	Filename	SVID1	VSN1	VSN2	SVID2	VSN5	...

Where filename = Hostname + user number + full pathname

Figure 10 : Structure of the F-record

5.2.2 The R-record

This type of record is used to store information about the save files. The identifier is composed by the letter "R" and by the internal save file identifier (SFID) followed by a filler in order to obtain the 155 positions needed for the access key. The SFID is made of the date and time of the save file creation. The following fields contain information about the save versions stored in the save file(Figure 11). Each save version is referred via its internal identifier (SVID).

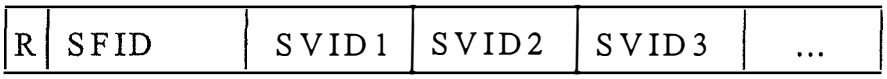


Figure 11 : Structure of the R-record

5.2.3 The S-record

This type of record is used to store information about the save versions. The identifier is formed by the letter "S" followed by the SVID and by a filler. The SVID is formed as the SFID, by the date and time of its creation. The following fields contain various information: among them, there are the identifiers of the volumes on which the save version is located (Figure 12).

Figure 12 : Structure of the S-record

5.2.4 The T-record

This type of record is used to store information about the volumes that are allocated to the archive. Its identifier is made by the letter "T" followed by the highest identifier of the volume contained in the record (Figure 13). The fields that follow are related to the volumes and contain the type, the availability and other data concerning the tape.

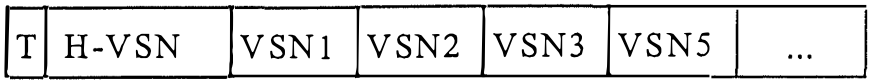


Figure 13 : Structure of the T-record

5.2.5 The X-record

This type of record is used to complete the S-records. It contains complementary information related to the save version.
It has the same identifier than the S-record except that the first letter is a "X".
It contains among other things, the optional name of the save version given by the user and the SFID of the save file that holds the save version (Figure 14). There is one X-record for each S-record.

X	SVID	SFID1	Information (save version name,...)

Figure 14 : Structure of the X-record

Here comes a example of a possible repository

F	Filename1	SVID1	VSN1	SVID2	VSN5	...
F	Filename3	SVID1	VSN2	SVID2	VSN5	...

...

R	SFID1	SVID1	SVID2	SVID3	SVID4	...
S	SVID1	VSN1	VSN2	VSN5	Save type, ...	
S	SVID2	VSN5	Save type, ...			
S	SVID3	VSN3	VSN4	Save type, ...		
S	SVID4	VSN5	Save type, ...			

...

T	VSN5	VSN1	VSN2	VSN3	VSN4	VSN5
X	SVID1	SFID1	Save version name, ...			
X	SVID3	SFID1	Save version name, ...			

...

Figure 15 : Structure of a repository

Remark: the Low key and the High key records are not displayed in the figure due to the lack of interest of those records as part of this study.

There are some types of records shared by several workstations. Those records are the T-records, the R-records and even the S and X-records in some case.
It is possible to have the situation described in Figure 16. In this situation, for a save file, there are save versions related to distinct workstations and save versions shared by several workstations. Thus, there are S and X records shared by some workstations, a R-record and the T-Record shared by all the workstation.

R	SFID1	SVID1	SVID2	SVID3	SVID4	...

Where SVID1 refers to files from Workstation 1
SVID2 refers to files from Workstation 3
SVID3 refers to files from Workstations 2 and 4
SVID4 refers to files from Workstations 1,2,3,4

Figure 16 : Shared records

5.3 DECENTRALIZATION POSSIBILITIES

Each of the theoretical possibilities will be estimated for HSMS.

5.3.1 Replicated repository

In order to obtain gain of performance for each workstation, a full replication is needed. This solution offers one of the best gains of time for the information retrieval as the requests can be fully processed locally. This is not the optimal solution because information retrieval is always made in a big file, and thus increases time loss. Another drawback is that each workstation holds information that it will normally never access thus wasting a lot of disk space. The update procedure for this situation is important as it requires the propagation to all sites.
This solution is fully applicable as there is no problem for the shared records, and does not pose any problem for the performance of the server.

5.3.2 Partitioned repository

This solution, theoretically near to the full optimum in term of access performance, is practically not very optimal for HSMS. The first reason is there is a need to fully reorganize the shared records. In order to be partitioned, those shared records must be splitted and it leads to a lost of directly available information.
Otherwise, this solution offers good performance for the HSMS clients as the local repositories contain only local information. It does not require much disk space, in fact it does not require more disk space than a centralized repository, with the advantage that this total space is shared by all the sites. The second reason is that, with this solution, the server looses its performance for the backup and the restore functionality. This is the bigger drawback of this possibility, making it non interesting for HSMS.

5.3.3 Partitioned and centralized repository

Theoretical proposal

This solution has all the advantages of the previous one but without loosing performance of the server as this one holds a full copy of the repository. It requires more disk space and a slightly more complicated update procedure. It also implies that the shared records must be splitted, leading to a lost of directly available information.

HSMS adapted proposal

By making only a partly partitioned repository (the partitioned records being the non shared ones) combined with the centralization of the full repository, there is no more problem of reorganization of the shared records, thus leading to a simple and quite optimal solution for HSMS. This solution is called *partly partitioned and centralized repository.*

5.3.4 Partly partitioned and partly replicated repository

Theoretical proposal

By replicating the shared records, this solution avoids the problem of their reorganization. This implies that updates made to these records must be propagated to all the workstations. As each workstation holds all its information, this solution obtains the best performance for HSMS client. Unfortunately, the server looses all its performances with this solution as it holds no other information than the local ones.

HSMS adapted proposal

By combining this solution with the centralization, this give a perfect solution in term of performances. However the performances are counterbalanced by the update procedure that requires propagation to all sites for updates concerning the shared records.
This solution is called *partly partitioned, partly replicated and centralized repository.*

5.3.5 Other Combinations

The other combinations don't offer more advantages than disadvantages.
With such solutions, a lot of disk space is needed, the update procedures are complex.
And for HSMS, those solutions are not acceptable.

5.4 THE CONTROL FILE

This file, also located on the server, contains, among other, information about the archives. As part of the decentralization, the only interesting information are the associations between the archives names and the repositories names. By decentralizing these, all the query requests can be fully treated on the workstations (depending on the decentralization schema chosen for the repository). The drawback of this is that it implies another update propagation procedure. If the control file is not decentralized, there is, prior the access to the repository, an access to the central control file to get the name of the repository.

5.5 SUMMARY

There are thus three solutions that are more or less applicable to HSMS. They are the "full replication", the "partly partition with the centralization" and the "partly replication, partly partition with the centralization". Each of these can be combined with the possible decentralization of the control file. This decentralization must be made by a partly partition in order to obtain good performances. The partition concerns only the names associations.

Solution	Advantages	Disadvantages
Replication without control file	Duplication security (with special mechanisms), performance +	Disk space, file size, central access to the control file, update procedure, data coherence
Replication with the control file partly partitioned	Security, performance ++, traffic	Disk space, file size, update procedures, data coherence for the control file
Partly partition & centralization without control file	Performance +, disk space, update procedure	Central access to the control file, traffic
Partly partition & centralization with the control file partly partitioned	Performance ++, disk space, easy update procedure, less traffic	Update procedure, data coherence for the control file
Partly partition & partly replication & centralization without control file	Performance ++, disk space	Update procedures, central access to the control file
Partly partition & partly replication & centralization with the control file partly partitioned	Performance +++, disk space, traffic	Update procedure (3 X), data coherence for the control file

6. INFORMATION EXCHANGE PROTOCOL

6.1 PRESENTATION

The information exchange protocol is divided in two parts : the control file part and the repository part. These two subdivisions are quite similar as they imply information transfer from the server to the clients. They have some differences however : the mechanisms that must be set for each, are different and require a separate study.

The study of the information exchange concerning the repository is itself divided in four : one subdivision for each of the possible repository decentralization schema (full replication; partial partition and centralization; partial partition, partial replication and centralization) plus one part for the mechanisms common to the three decentralization possibilities.

6.2 THE REPOSITORY INFORMATION EXCHANGE

6.2.1 Order

The order of update is the first point to approach before studying the obtaining and the propagation themselves. The order concerns which, from the server or the clients, will get the update information first.

But, before comparing the advantages and disadvantages of the different order possibilities, it is important to note that for security reason, it is necessary that there is quickly an updated repository.

So even if there is a problem with a site or with the network, it can be possible to restore the lost information concerning the updates. Obviously, if it is the site that holds the updated repository that crashes, it does not help to have it quickly updated.

For each of the three possibilities of decentralization, it exists three options for the propagation order : the clients first, the server first or the clients and the server at the same time.

The clients before

The case where the client is updated first has several drawbacks. It is not the quickest solution : it requires the sending of datagrams to the clients, with all that it implies (possible errors, traffic,...). The sending of datagrams generally increases the risk of failure before a repository is updated. As it is necessary to have all the information concerning the update before beginning the update operation itself on the clients, there is a quite long delay during which no repository is updated.

And, for a matter of quickness and availability of the information, it is not the simplest solution.

The server first

All the information concerning the update is located on the server. It is thus more logic and more easy to update the server first and the clients next as all the update information is located there. It also offers the quickest way to obtain an updated repository as soon as possible. Once the update on the server is made, the update can be sent to the clients. It requires for this operation special mechanisms in the case of problems during the transmission of the update itself.

The two together

Updating the server and the clients together is a solution that depends on the sending time. If it is made as soon as the information is available, then it is a good solution. But despite that, it requires the setting of special mechanisms to record which repositories are updated when there is a network or a site failure during the update operations. If the sending is not immediate, this solution is not better than the "client first" solution. There is no quick update repository and it requires mechanisms in case of site failure during the operation.

6.2.2 Replication case

Update obtaining & propagation

In order to propagate the update to the sites, it is necessary to obtain it after each operation. In the replicated case, the update is the same for all the sites on the network.

What must be set in the update and how ?

There are several possibilities for the information that must be set in the update.

First solution

The first solution is the easiest one : it consists of updating the server's repository and then to send it to all the clients. But, unfortunately, this solution cannot be implemented for two majors reasons. First, it is not an economical solution as the repository can be a quite big file, thus involving lot of traffic for the sending to the clients. Second, as the server's repository

is a BS2000 file, it is not directly portable to the UNIX workstations, but it can possibly be converted.

Second solution

Another solution consists of getting only a descriptor of what has changed in the repository and to send this one to the workstations. The descriptor contains only the pathname that was given in the request that initiated the changes in the repository, with the identifiers of the save version (SVID) and the save file (SFID) in which the files have been saved and the volume identifiers (VSN) on which the files are located.

With this kind of solution, the descriptor, when received by the workstations, is stored for further use. The workstations, when necessary, will send a request to the server to get the update associated with the descriptor.
The big drawback of this solution is that the update is send to only one workstation at a time, and therefore this processing must be made for each workstation that has received the descriptor.

Third solution

The last solution is to get the update for the clients during the server's update. Each time something is written or deleted or modified in the server's repository, it must be recorded in a separate file (see Figure 17), together with an indication of the operation. So, once the update is completed on the server, all the necessary information for the decentralized update is contained in this file. This one must be then sent to all the workstations.

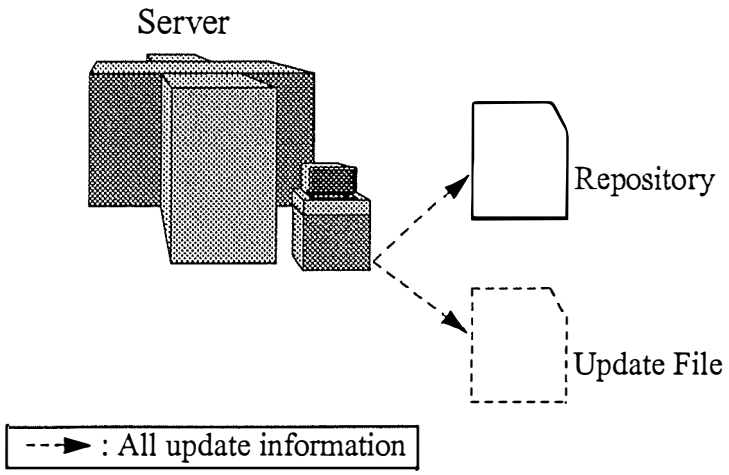


Figure 17 : Update obtaining in a replicated case

Update location

For the update location, there are two possibilities : the update can be located on the BS2000-DMS part or on the BS2000-UFS part. The BS2000-DMS part is the main file system, and the BS2000-UFS is a UNIX file system integrated in the BS2000.

Update propagation

Once the files are ready to be sent, they must be propagated to all the workstations. But there are several possibilities that exists for the sending of these files.
These possibilities are : as soon as the files are ready, at regular intervals or once a day.
In the case the propagation does not take place directly, special mechanisms must possibly be set for the accesses to a non updated repository (this point is studied later in this chapter).

Immediate sending of the files

The immediate sending of the update files offers the advantage that the decentralized repositories are quickly updated. The traffic required can be more important than in the centralized case, especially in case of many short operations concerning the repository. Each of these must be propagated to the decentralized sites, leading thus to a lot of traffic of lesser size. And this traffic is made during period where the network can be congested by the traffic generated by the other applications. It can be slowed down by these applications, or can slow down these ones, reducing the propagation speed.

As the update propagation can concern sites that are not involved in the operation that starts the update (situations that can exist when there is a replication (full or not)), mechanisms must be set to cope with this special situation.

These mechanisms require that, as soon as the unavailable sites are accessible again, the update is sent to them. For this, it is necessary either that the site itself warns that he is accessible again, or that the server continues to try to send the update until it gets an acknowledgment from the site. These sending attempts must be made at regular intervals, as it is useless to try continuously.

Sending at regular intervals

To ensure a sending at regular intervals, a program must regularly check if there are files to be sent. The intervals between two checks can be either predefined or defined by the HSMS server's administrator.

But with this solution, the decentralized repositories are not quickly updated. The traffic on the network can be lesser than in the immediate sending case. It is due to the possible gathering of several updates in one during this interval. This gain of traffic is counterbalanced by a greater load on the workstations' CPU as one update can contain the records resulting from several operations. This load, in place of being spread out in many times is concentrated in one time.

Once a day sending

The "once a day sending" means that the update propagation only occurs once a day. It implies that all the decentralized repositories are rarely up to date and thus that accesses to these are accesses to non-updated data.

This type of propagation can be considered as a particular case of the previous solution with a sending interval set to a full day. This solution limits the most the update traffic since all the modifications resulting from the operations on the repository are gathered in one update. This advantage is counterbalanced by the fact that the workstations CPU's loads are important as all the data must be processed in one time. In the other cases, this load is more or less distributed during the whole day.

But, as the propagation normally takes place during the night or at any other period where the traffic is low, a heavy load of the workstations CPUs is theoretically not embarrassing.

The two last solution requires a study of the accesses to a non-updated repository.

Access to a non updated repository

Accesses to a non-updated repository can be made in either of three ways.

First solution

The first solution consists in allowing the access just as if the repository was up to date, without prior notice. It is the easiest to implement : there is absolutely nothing to change to the access procedure. However it is not acceptable. The displayed information can be wrong and the user is not warned.

Second solution

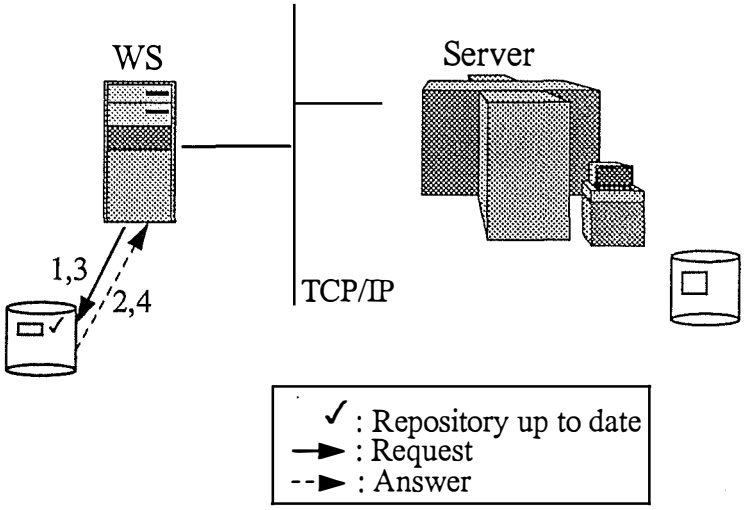
The second solution is very similar to the first one; it consists in allowing the access to the repository but with a signal that warns the user the displayed information is not up to date. It requires a few changes to the access procedure and to the repository's file. In order to be able to warn the user that the repository is not up to date, it is necessary to have a flag positioned with the file. The access procedure must also be modified : a verification of the flag must take place at each read operation of the repository. Following this result, the procedure must display a warning message or not.

This solution is opposed to the property of distribution transparency : the user is warned that remote operations are not yet completed. Despite the warning, this solution is not acceptable. The displayed information can be wrong and if it is the case, the user is warned that there is wrong information but does not know which one.

Third solution

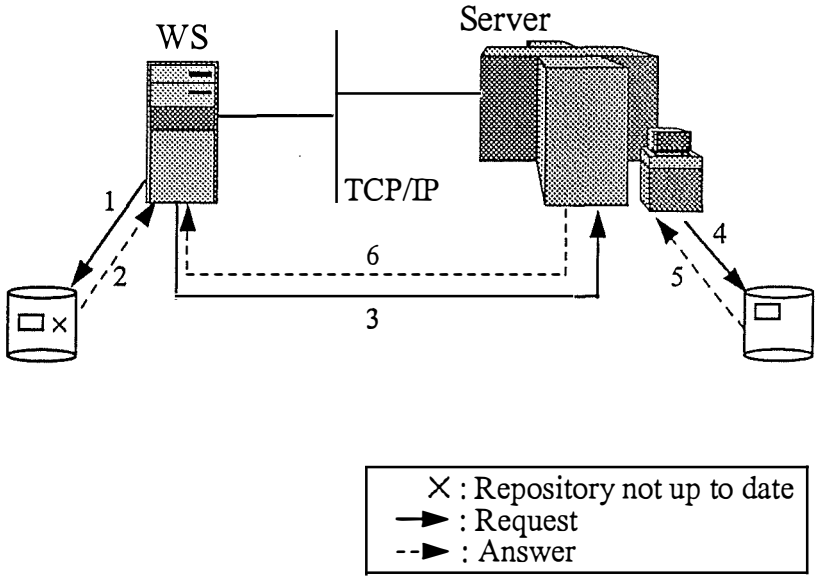
The third solution is more complex than the previous ones : the access to the non-updated repository is redirected to the server's repository. It is the most complex to implement but assures that the displayed information is always the most recent and correct. It requires a modification of the access procedure and a modification to the repository's file. The modification of the file is the same than in the previous solution : a flag is positioned to indicate if the file is up to date or not. With this solution, the access procedure must check the value of this flag. Following the value of this one, the access is made to the local repository or to the server's repository.(see Example 1 & Example 2)

But the gains of time foreseen by the decentralization can be reduced to nothing with this option. If a modification of the repository take place just after a propagation, the related decentralized repositories are no longer up to date and access to them is redirected to the server, leading to the same situation than before the decentralization.
However, the positioning of a flag on the repository requires sending signals from the server to the stations in order to set the right value to it when there is an update that waits to be propagated.



1 The application accesses to the repository in order to know if it is up to date
2 The answer is : the file is up to date
3 Then the application accesses to the file to get the requested information
4 The requested information is given to the application

Example 1



1 The application accesses to the repository in order to know if it is up to date
2 The answer is that the file is not up to date.
3 The request is then transferred to the server.
4 The server accesses its own repository to get the requested information.
5 The requested information is given to the server
6 It is then transferred to the requesting application

Example 2

Client Recovery

There are two possibilities to recover the decentralized repositories in the replication case.

The first solution is to have a special update procedure on the server. This procedure takes all its information directly from the repository. The difference with the normal update procedure is that it does not operate when there is something written in the repository. It is an "update obtaining" procedure based on the supposition that the decentralized repository is empty; it must thus take all the information that is necessary to reconstruct the crashed repository. All the needed information is written in a file that is sent to the station as a normal update file. The starting of this procedure can be made in two ways : the first one, the server's administrator starts it manually; the second one, the recovery request is sent by the workstation itself (by its administrator).

The other solution, consists in exchanging signals between two workstations, the crashed one and another. This solution is applicable whatever the origin of the recovery request. In the two cases, the server must transfer to the crashed workstation the address of another site that can give the information.

After getting the address, the workstation must contact the other and get the repository. With this mechanism, the repository is directly transferred from one site to another without any special "update obtaining" procedure.

The drawbacks of the solution are that it necessitates two exchanges on the network in place of one and that it necessitates authentication mechanisms between the workstations to avoid unauthorized transfers and accesses to the repositories.
It has the advantage that it requires no special update obtaining procedure.

The advantage of the first solution are that the procedure can be used to migrate from a passive client to an active one and from a version of HSMS-Client with a central repository to a version with a decentralized one.

6.2.3 Partial partition & centralization case

Update obtaining & propagation

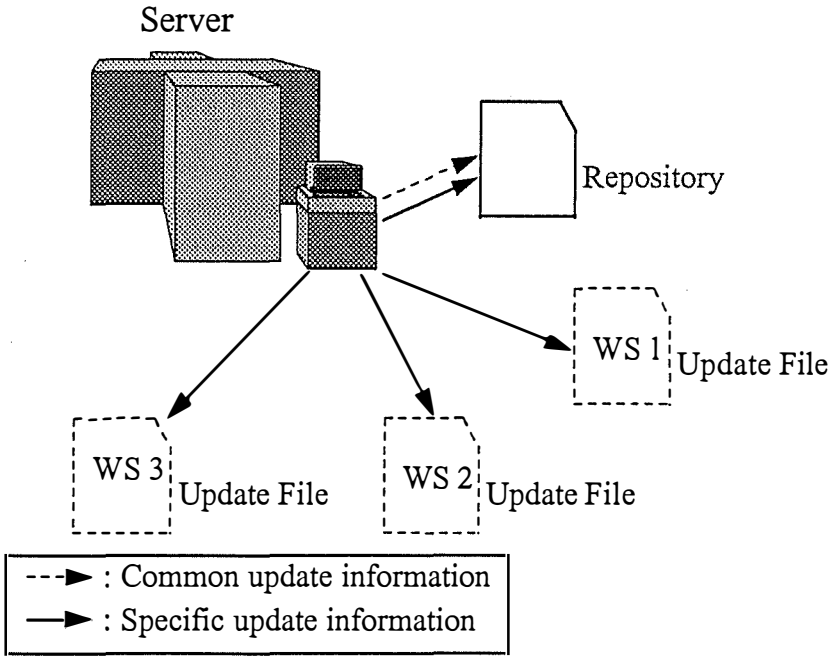
For the partial partition, the update only concerns the server and the workstations targeted by the operation that triggered off the update.

What must be set in the update and how ?

There are two possibilities for the update contents : either the whole update information or a descriptor of what has changed is stored.

First solution

The first solution, getting the whole update information is similar to the obtaining procedure described for the replication case. It is made during the server update but it only concerns the F-records. As the operation that triggered the update can involve several workstations and because the information is not shared, it requires as much files as there are workstations involved. During the recording, the information is directly sorted to each file, so the files contain no useless information (see Figure 18). Once the operation is finished, each file must be sent to its workstation.



During the update of the central repository, if the information to be written is a shared one, it is only written in the central repository. If it is a non shared information, it is normally written in the central file and it is also written in a file related to the concerned workstation

Figure 18 : Update Obtaining in a partitioned case

Second solution

The second solution is to create a descriptor of what has changed during the operation for each workstation involved. This descriptor contains the pathname that was given with the command, and the identifier of the save version (SVID) in which the files have been saved.

Once the descriptors are ready, they must be sent to their related workstation. At the reception on the workstation, the descriptor is stored on disk in a directory accessible by the HSMS-Client program.

For each operation on the client that must access the repository, the pathname of the files for which information is requested must be compared with the pathnames that are received in the descriptors. If they match, it means that the requested records are not up to date. If they don't, the access is allowed to the repository (the term match means checking for an eventual inclusion of the requested pathname in the pathnames contained in the descriptor).

If the requested information are not up to date, there are three possibilities for the processing :

- the access is redirected to the central repository to get the requested information and a partial update is executed with the received information.
- the access is delayed, a full update is executed (concerning all the pathnames received on the workstation that have not yet been updated) and then the access is allowed
- the access is redirected to the central repository and in parallel a full update is executed

Whatever the possibility, a mechanism must be implemented to allow periodic update request based on the received pathnames.

The first solution is simple in its principle but requires a complex mechanism for the periodic update request. For the periodic requests, it is necessary to have parameters that allow to exclude the pathname that have already been updated. It leads to a more complex mechanism as the updated pathname must be recorded for the periodic request. The drawback is that it is possible to have many short updates generated. It also requires mechanisms to avoid duplication : before launching a new update request, it is necessary to verify if the requested information is not being transferred. If it is the case, the second access must be blocked the time the requested information is updated in the repository.

The second solution is a simple one, but the first user that will access the non updated information contained in the repository will have to wait for a long time before receiving it. It has the big drawback that there is a user who is heavily penalized. If a second request for the non updated information is generated before the update completion, it must be blocked until the first one is finished.

The third solution has the advantage that the first user that wants to access to the non updated information is not too much penalized, as he gets the information from the server and has not to wait for a full update of the local repository. The following accesses to the repository will be redirected to the server if the repository is still not updated or processed locally if it is updated.

The three solutions requires that the process that manages the updates on the workstation access the file where the descriptors are saved in order to keep it up to date with the latest processed update. When the information are requested on the server, it is done via the SVID associated with the concerned pathname. The SVID is used to find the information in the central repository.

Update location

For the update files, the same choice as for the replication case is given : place the files on the DMS or on the UFS. For the descriptor, it can be placed on disk as the other update files or can be placed in memory in a buffer as it does not occupy much space.

Update propagation

As in the replication case, there are three solutions for the propagation of the update. The three possibilities are exactly the same (immediate sending, at regular intervals or once a day) and the same mechanisms must be set up in case the sending is not immediate.

Client Recovery

There is only one possibility to recover the decentralized repositories in the partition case.

This solution consists in having on the server a special update procedure. This special procedure works exactly as the special one that is described in the replication case.

6.2.4 Partial partition, replication & centralization case

Update obtaining & propagation

In the partly partitioned and replicated case, the update concerns the server and all the workstations. It is possible to restrict the update propagation of the shared information to the workstations that are involved in the operation that has triggered the update.

What must be set in the update and how ?

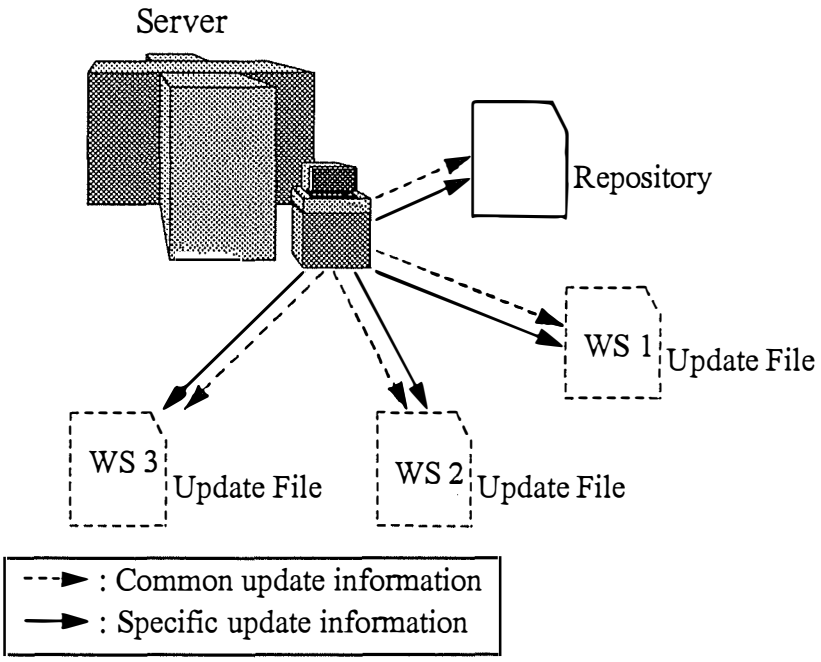
There are two possibilities for the update content : either the whole update information, or a descriptor of what has changed in the repository. Each of these two solutions is itself divided in two parts.

First solution

There are two possibilities for the whole update information :

- One file per workstation containing all the needed information for the update. It means that each file contains a copy of the shared records.
- One file per workstation containing the non shared data and one file containing all the shared data.

The first solution requires more disk space to store all the files on the server, the shared information being written as many time as there are sites containing the repository (see Figure 19). This type of organization requires more CPU and disk accesses as the shared data must be written in each file.

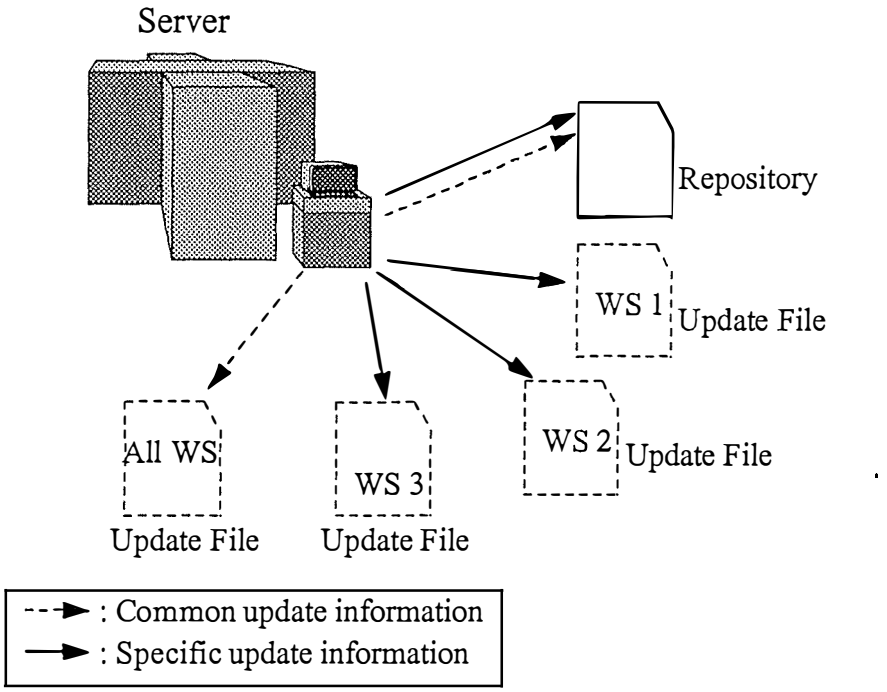


During the update of the central repository, the shared information is written in the central file and in each of the workstation related file. The non shared information is written in the central repository and in the workstation related file.

Figure 19 : Update Obtaining in a partitioned an replicated case (solution 1)

The other solution reduces the CPU load, the disk space and the disk accesses as the shared data are written only once (see Figure 20). For the sending of the files in this case, there are two options again :

- send the two files separately to each workstation, i.e. the file containing the shared data is sent to all the stations through a broadcast on the network.
- send the two merged files to each workstation, i.e. the two files are merged before the sending (this solution is similar to the first organization solution)



During the update, the shared information is written in the central repository and in a file common to all the workstations. The non shared information is written in the repository and in the workstation related file.

Figure 20 : Update Obtaining in a partitioned an replicated case (solution 2)

Second solution

The second solution for the update is to have a descriptor of what has changed in the central repository, i.e. the pathname that was given with the command, the SVID and the SFID. Again, there are two possibilities for the contents of the descriptor : either the SVID and the SFID are given with the necessary information to be updated at the reception, or they are given without any other information and when the update is requested, the supplementary information about the SVID and the SFID is added to the file information. With the first possibility, the SVID and the SFID are updated at the reception. The updates of the other records are made as in the partition case.

Update location

For the update files, the same choice as for the partition case is given : place the files on the DMS or on the UFS. For the descriptor, it can be placed on disk as the other update files or can be placed in memory in a buffer as it does not occupy much space.

Update propagation

As in the replication and the partition cases, there are three solutions for the propagation of the update. The three possibilities are exactly the same (immediate sending, at regular intervals or once a day) and the same mechanisms must be set up for the case where the sending is not immediate.

Client Recovery

There is only one possibility to recover the decentralized repositories in the partition & replication case.

This solution consists to have on the server a special update procedure. This special procedure works exactly as the special one that is described in the replication case.

6.2.5 Common mechanisms

The common mechanisms regroup the mechanisms that are common to all the decentralization schemas. They are the security and integrity, the access control, the transmission problem and the concurrency control.

Security and integrity

This part concerns the security and the integrity of the repository's file, i.e. the access restriction to the file itself, not to the contained information. These restrictions are implemented by forbidding the accesses that are not initiated from the HSMS client program. Thus, direct access to the file is impossible. So the file is protected from intentional or unintentional deletion or modification by a user.
With such restrictions, it is impossible to read, write or modify the repositories without using HSMS.

Access control

The access control concerns the access restrictions to the information contained in the repository. In the present situation, the users can only access to the information related to their files or to the information that are shared. The same situation must be implemented in the decentralized solution. Thus, the program must be modified in such a manner that for each request to read information from the repository, it must check the access permissions to the records and reject the request if read is not allowed.

Transmission problem

As the transmissions are based on TCP/IP, a reliable stream service, the problem are reduced to the network failures. In this case, the mechanism is the same as for the unavailability of a site, the server will retry to send the update at regular intervals until the communication is restored and the transmission correctly executed.

Concurrency Control

The problem of the concurrency control is quite reduced in HSMS as there is only one program that is allowed to write or modify the records contained in the repository. The workstations users can only read the records. It is thus necessary to have a mechanism that manages this problem.

6.3 CONTROL FILE INFORMATION EXCHANGE

You can manage the control files in two ways : keeping the control file on the server, or decentralizing the associations between repositories and archives names.
Whatever the chosen possibility, it requires modifications to the present situation.

The first solution, whatever the repository decentralization, reduces the site autonomy : as a preliminary access to the server must be made, the access to the repository can be blocked by other actions coming from other sites, or by network failures. This access increases the response time too.

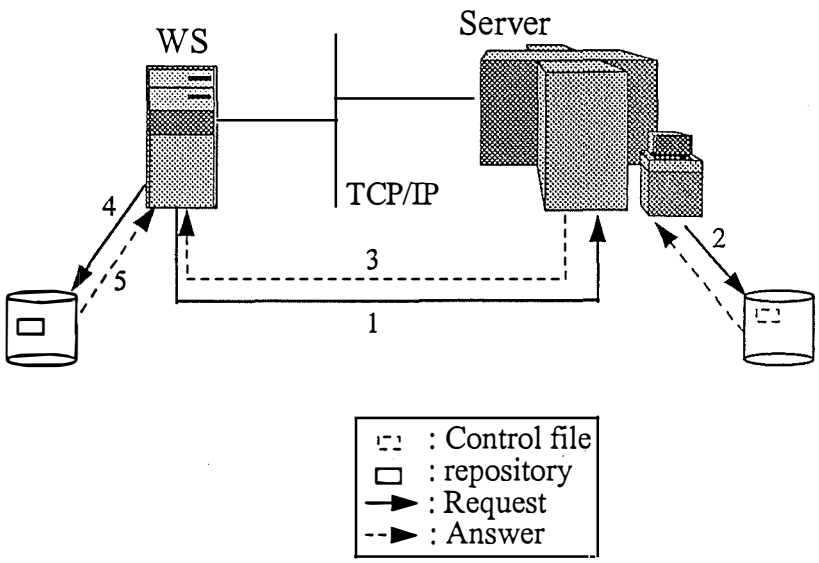
The second solution has the advantage that the site autonomy is increased. If it is combined with a repository decentralization that gives a great site autonomy (partial or full replication), the site autonomy becomes total. The workstations can consult their data without being blocked by another one. It can access the information even if the server is unavailable or if network failures occur.

6.3.1 Centralized control file

With a centralized control file, each access to a repository on a workstation must be preceded by a consultation of the control file on the server. This consultation is mandatory to get the name of the repository's file associated to the requested archive. Without this consultation, as the repository's name is not a default one and can change during his life time, no access is possible. It is also possible to have several archives of the same type in the future (Figure 21).

As this access does not exist in this form (a request to have only the name of repository), it must be implemented. In the present situation, the access form to the control file is the following one : it is combined with the information request transmitted to the server. When the users of a workstation want information on the files contained in an archive, a request is transmitted to the server. This one checks in the control file to obtain the repository's name corresponding to the request and access then to the concerned repository. The request to access the control file only does not exist. It is thus necessary to implement a new client request to get the information from the server. When the server gets this new request, it must just consult the control file and transfer the information to the requesting workstation. It is important to note that this new mechanism can be copied from the first part of the old mechanism.

In the case of a partial partition of the repository is chosen, all the mechanisms must be kept, as the requests for non local information must always be transmitted to the server to get a response.



1 The workstation sends a request to the server to get the repository's name.
2 The server accesses to the control file to get the requested name.
As the contents of the control file is kept in memory, the request is processed without accessing to the file.
3 The name is then transmitted to the requesting workstation.
4 This one accesses the repository.
5 The requested information is given

Rem : the points 4 and 5 may vary according to the repository's access methods.

Figure 21 : Access to the central control file

6.3.2 Decentralized control file

With this solution, all the problems that are present for the repository also exist for the control file. Thus, the same aspects must be approached during the study : update obtaining & propagation, security, recovery,...

Update obtaining and propagation

As the decentralization is a partial partition, the update is different for each workstation. The obtaining mechanism is exactly the same one as for the partly partitioned repository. During the writing in the central control file, all the decentralized information is written in files or in a buffer. There are as much files as workstations, one file per workstation. In each of the files, the information concerning the workstation is written, once the writing finished, all the non empty files can be sent to the stations (see Figure 22).

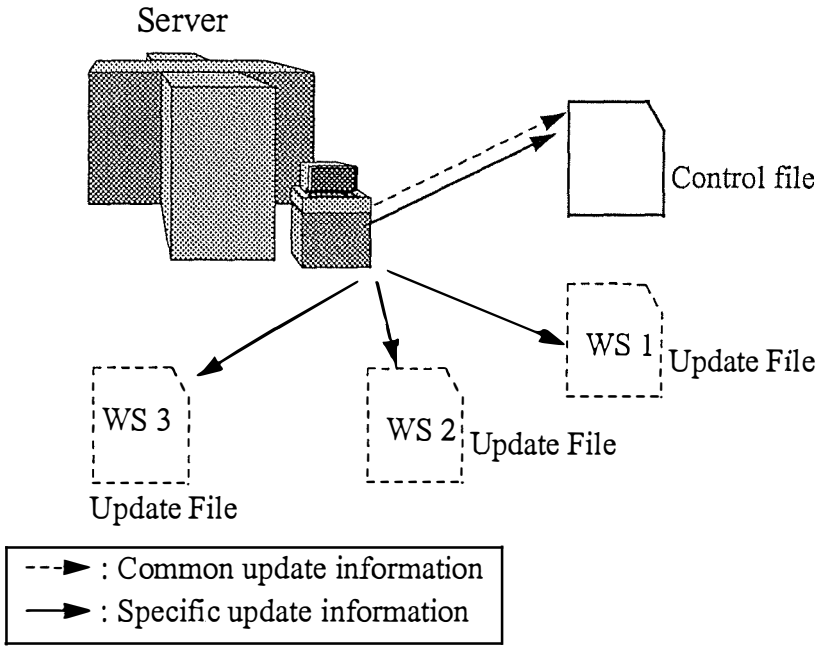


Figure 22 : Control file update obtaining

For the propagation, the same options as for the repository's update exist.
As the update of the control file concerns the archive (creation or deletion) and especially the repositories, it is important that the new information concerning these ones is quickly propagated. If it is not the case, it is possible that accesses are made to a non existent, deleted archive, or that accesses to newly created archive and repository can be blocked. Thus the propagation must be immediate to avoid such problems. With the direct propagation, there is no problem of access to non updated data.

Security and access control

The same security mechanisms as for the repository must be set up so as to avoid direct access to the contents of the control file. Only the HSMS client program can access this information. The access control to the information contained in the file has no reason to exist, as the information is shared by all the users of the station.

For the recovery procedure, it is again the same solution as for the repository.
A special procedure must be implemented to obtain an update that has all the information contained in the file.

And for the concurrency control, it is exactly the same solution, only the update program can access the file to write, all other accesses are read accesses. Thus, a write locking solution solves the problem of concurrency control.

6.4 IMPLEMENTATION CHOICE

To implement the repository on the workstations, there are two majors solutions :

- an ISAM file
- a database

6.4.1 ISAM file

The solution of implementing the repository in an ISAM file is the easiest one. It is also the cheapest solution and is similar to the situation that exists on the server. It has some advantages and some disadvantages.

The first advantage is that it requires no major changes to the records organization. The second one is an economic one : this kind of implementation is portable on each type of workstation and operating systems (HP, Sun,...) and thus does not requires separate implementations.

One of its disadvantages is that all the management of the concurrency control, if not realized by the ISAM file manager must be made in the HSMS-Client program. Another disadvantage is that this implementation must be preceded by a choice of the type of access that will be optimal. The other types of access to the information won't be optimal.

6.4.2 Database

The solution of implementing the repository with a database is more costly as it requires to dispose of a database manager software on each workstation. It also requires a full reorganization of the records structure and requires implementing different solutions for each database software. An advantage of this kind of implementation is that all the management of the concurrency control is made by the database. Another one is that implementing the repository via a database can allow to optimize several distinct types of request without any modification.

7. INTEGRATION IN HSMS

7.1 INTRODUCTION

This chapter is divided in three parts : the present architecture of HSMS, the solution proposal and the modifications to the HSMS architecture.

The presentation of the present architecture is intended to explain the basic functioning of HSMS. The solution proposal gathers the choices that have been made concerning the decentralization model, the implementation, the update obtaining and propagation models and the client recovery procedure. Next to this, the choice for the decentralization of the control file is also explained, together with some complements to the repository's decentralization.

7.2 PRESENT ARCHITECTURE OF HSMS

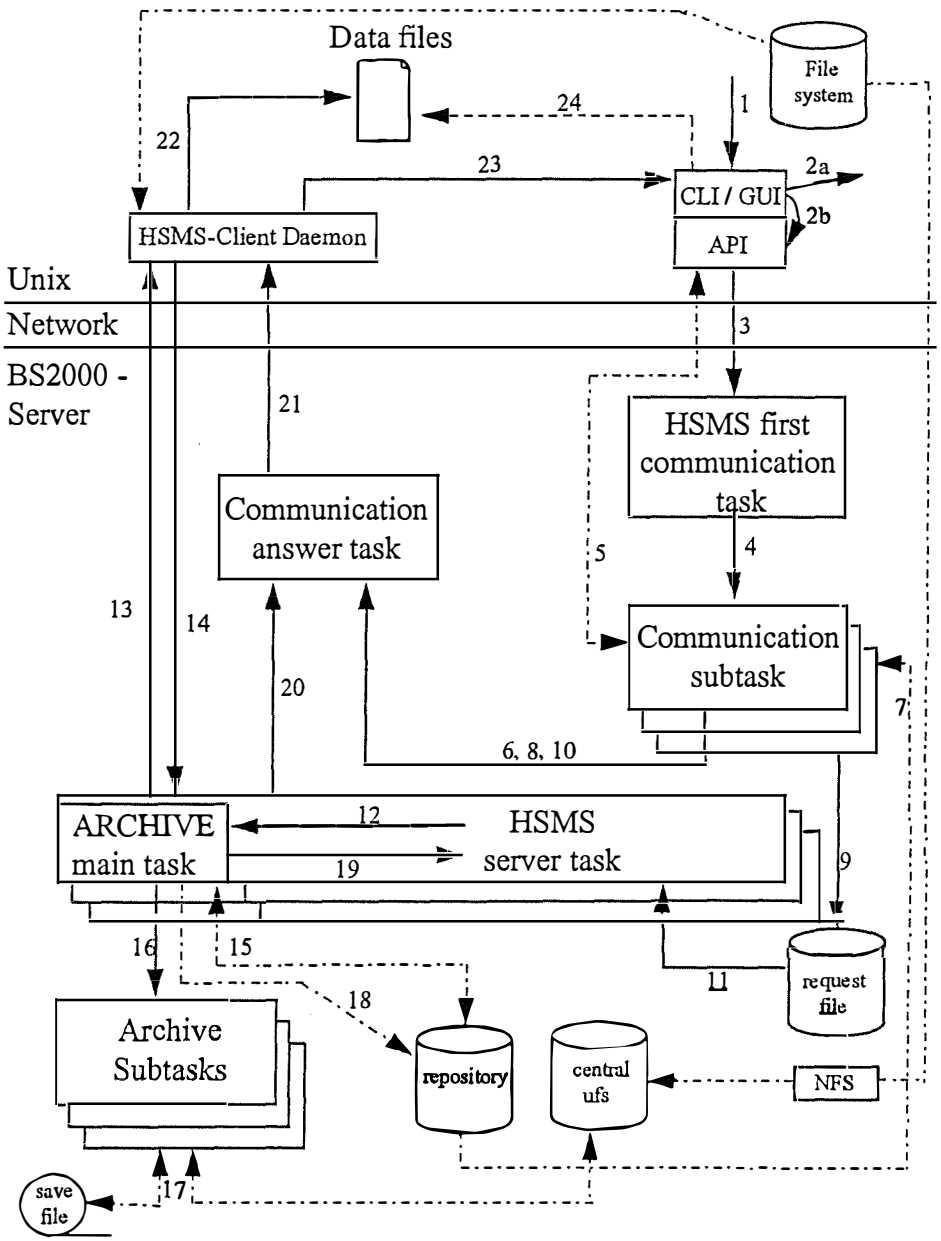


Figure 23 : HSMS architecture

1 A UNIX user issues a HSMS-CL statement on his workstation. A first local syntactic check occurs on this statement ; it is only a formal check, not a detailed one.

2a If the statement contains an error, the process is stopped and an error message is sent to the user.

2b If the statement is correct, the corresponding API function is called.

3 The API function then builds a structure describing the command following the HSMS client-server protocol and sends this data on the network to the BS2000 HSMS server.

4 On the BS2000 side, the connection event is received by the HSMS first communication task, which immediately redirects it to a free communication subtask.

5 The communication subtask reads the data from the network and checks whether it complies with the HSMS client-server protocol.
If the protocol is correct, the subtask builds a control block which contains all the network-related data of the remote command, then translates the structure into a HSMS BS2000-like statement before executing full syntactic and semantic checks.

6 If the statement is refused, the control block is passed to the Communication answer task with a negative return code, and the subtask returns idle.
If the statement is accepted, it is translated into a 'request' which is then processed.

The next steps are different according to the type of the request.

7 Show statements are processed by the communication subtask itself, which opens the corresponding repository to find the requested information.

8 The result and the control block are then passed to the Communication answer task (step 19).

9 Action statement are more complex. The subtask writes the request into the request queue.

10 It then passes the control block to the Communication answer task with a return code 'accepted' and finally return idle.

11 The request waiting in the request queue is eventually taken and processed by a HSMS server task.

12 The server task reads the request and calls the ARCHIVE product via a private interface. The task is then called 'ARCHIVE main task'.

13 The ARCHIVE main task asks the HSMS-Client daemon the complete list of the files to be processed.

14 The HSMS-Client Daemon access the file system to get the requested information and then transfers them to the Archive main task

15 The ARCHIVE main task builds the list of files to be processed by matching the list received from the daemon, and the pathnames already present in the repository.

16 It then creates an ARCHIVE subtask and transmits to it the list of files.
It is possible to have several subtasks created by the main task to work in parallel on a part of the list in order to obtain better performance.

17 The subtasks are responsible for the physical access to the storage media.

18 Once all the subtasks are finished, the ARCHIVE main task update the concerned repository if necessary.

19 The ARCHIVE main task returns to HSMS.

20 The HSMS server task generates a report which is passed with the control block to the Communication answer task

21 When the Communication answer task receives a control block, it opens a network connection to the HSMS-CL daemon of the UNIX workstation and sends the result.

22 On the workstation, the daemon receives the result and stores it in a file.

23 If the user process is still waiting, the result is directly fetched to it, and it is displayed.

24 If no user process is waiting, the result stays available and the user may read it at any time by issuing a specific HSMS-CL command.

7.3 SOLUTION PROPOSAL

7.3.1 Decentralization model

The chosen solution for the decentralization of the repository is the partial partition, partial replication and centralization. From the three remaining possibilities of decentralization, it is the one that gives the best performance without requiring too much disk space. For the implementation of the repository, the ISAM file has been chosen as it is the most economical solution : it requires only one implementation for each type of workstation (the database solution requires several implementations for each type of workstation).

But compared with the solution described in the previous chapters, the implemented solution will have some changes. These changes mainly apply to the shared records (the R, S, T and X records).

The first main modification concern the T records : as they are not used on the UNIX workstations (they are only used for the backup/archive action by the ARCHIVE subtasks), there is absolutely no need to decentralize them.

The second change concern the R, S and X records : they will only be replicated if they are effectively shared by several workstations. If it is not the case, they will only be send to the concerned workstation. And even if they are shared, only the sharing workstations will get the information.

With these modifications, the workstations will have incomplete information about the shared records, but as these ones are useless on some workstations, it does not matter. It has the advantage that it simplifies the update procedure.

7.3.2 Update obtaining and propagation

For practical reasons and disk space usage, the chosen solution for the update is the descriptor option (the full information need too much disk space, especially the full file names). The descriptor is the "short" version, i.e. the version which only contains the pathname given with the command that initiated the update, the SVID and the SFID, and nothing else. It can be placed in a buffer before being sent to the related workstation (the information is quite reduced). By choosing this solution, the central repository is considered to be always the first one to be updated and the reference for the other decentralized ones.

At the receipt of the descriptor on the workstation, it is stored in a file. The name of this file is contained in the configuration file of HSMS-CL. There is one file per repository (thus one descriptor file per archive).

As regards the propagation, the descriptor is sent as soon as it is ready while the update itself is only made periodically. When there are accesses to a non updated repository, they must be redirected to the central repository. It means that it is very important to have a good planning of the periodical update requests to have a gain of time due to the decentralization of the repository. If this planning is not correctly made, the decentralization offers no advantages. It is thus important that the workstation administrators are well informed of this situation in order to be able to synchronize more or less the update request with the backup operations.

The choice of redirecting the accesses to non updated records towards the central repository, ensures the usage of HSMS : generally, the backup/archive operations are processed during the night. By setting the periodical update requests in concordance with the end of these operations, the repositories are normally up to date for the day operations.

With such configuration, the only case where the repository can be no up to date is when a user has made a backup/archive during the day. In this situation, when this user wants to consult the records, his accesses will be redirected to the central repository if he wants to have information about the files he has just archived. The other users of this workstation will have some of their accesses redirected (consultation of the SFID and SVID records)

7.3.3 Client recovery

For the client recovery, there is only one solution : it is the special update procedure. This procedure must be initiated from the workstation, and not from the server. All the problem is to detect the inconsistency of the information contained in the repository. It requires to have a tool to check the different relations that exist between the records (see chapter 7). It is the role of the workstation administrator to initiate the recovery when he notes a problem with the repository. This procedure can also be used to migrate from passive client to active client or from a centralized repository version of HSMS-Client to a decentralized one. Moreover, it can be considered as a particular case of the normal update procedure, the difference residing in the requested information.

7.3.4 Control file

The chosen solution for the possible decentralization of the control file is an intermediate solution between the centralization and the decentralization.
This solution consists in writing in the HSMS-CL configuration file the full name (with the pathname) of the repositories files (one for the archival Archive and one for the backup archive). This writing must be made during the installation of the HSMS-CL program.
As the names of the repositories are not expected to change during their lifetime, it does not require a special update procedure for their management.

7.3.5 Complement to the solution proposal

There are some things that must be added to the solution to avoid problems or unnecessary traffic.

The first thing to add to the solution is critical : it is the conversion from the EBCDIC code (used on the BS2000) to the ASCII code (used on the workstation). The conversion procedure already exists in HSMS, it is used for the conversion of the reports that must be sent to the workstations. Thus, it can be reused for the update.

The second thing to add to the solution is not critical, and if not implemented, leads to useless traffic. It consists a list of the active clients to have on the server. With this list, the HSMS server knows to which workstations it is necessary to send a descriptor after a backup/archive operation that concerns all clients and that has been initiated by the server itself.

7.4 MODIFICATIONS TO THE ARCHITECTURE

For each of the possible operations on the workstations that access to the repository, the modifications involved by the decentralization are studied here.

7.4.1 Backup/archive actions

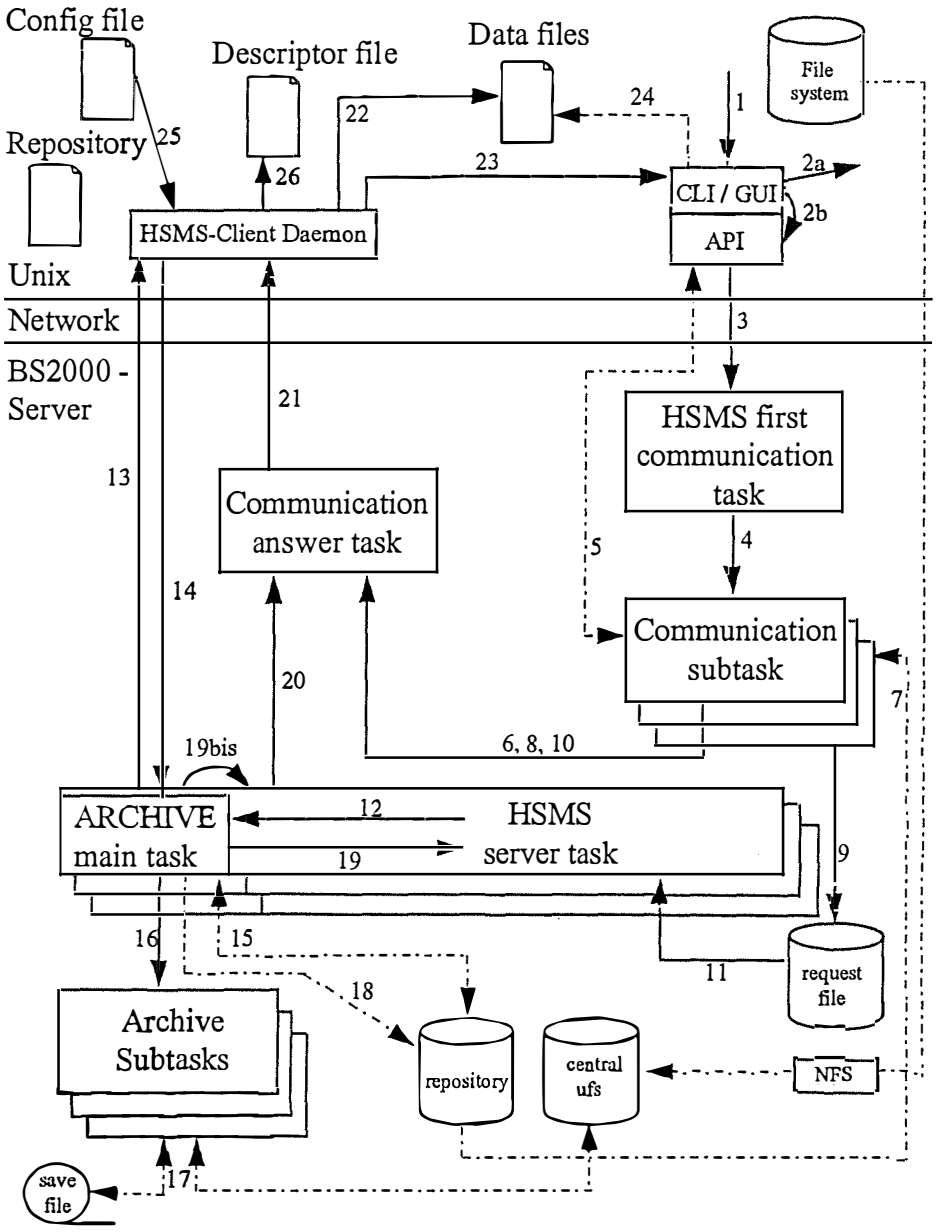


Figure 24 : Modifications due to the decentralization

Until the step 19, there is no change to the architecture.

19bis The descriptor is generated with the SFID, the SVID and the pathname.

20 The HSMS server task generates a report which is passed with the control block and the descriptor to the Communication answer task

21 When the Communication answer task receives a control block, it opens a network connection to the HSMS-CL daemon of the UNIX workstation and sends the result and the descriptor.

22 On the workstation, the daemon receives the result and stores it in a file.

23 If the user process is still waiting, the result is directly fetched to it, and it is displayed.

24 If no user process is waiting, the result stays available and the user may read it at any time by issuing a specific HSMS-CL command.

25 The daemon gets from the configuration file the name of the file where the descriptor must be saved.

26 The daemon stores the descriptor in the corresponding file.

7.4.2 Periodic update obtaining

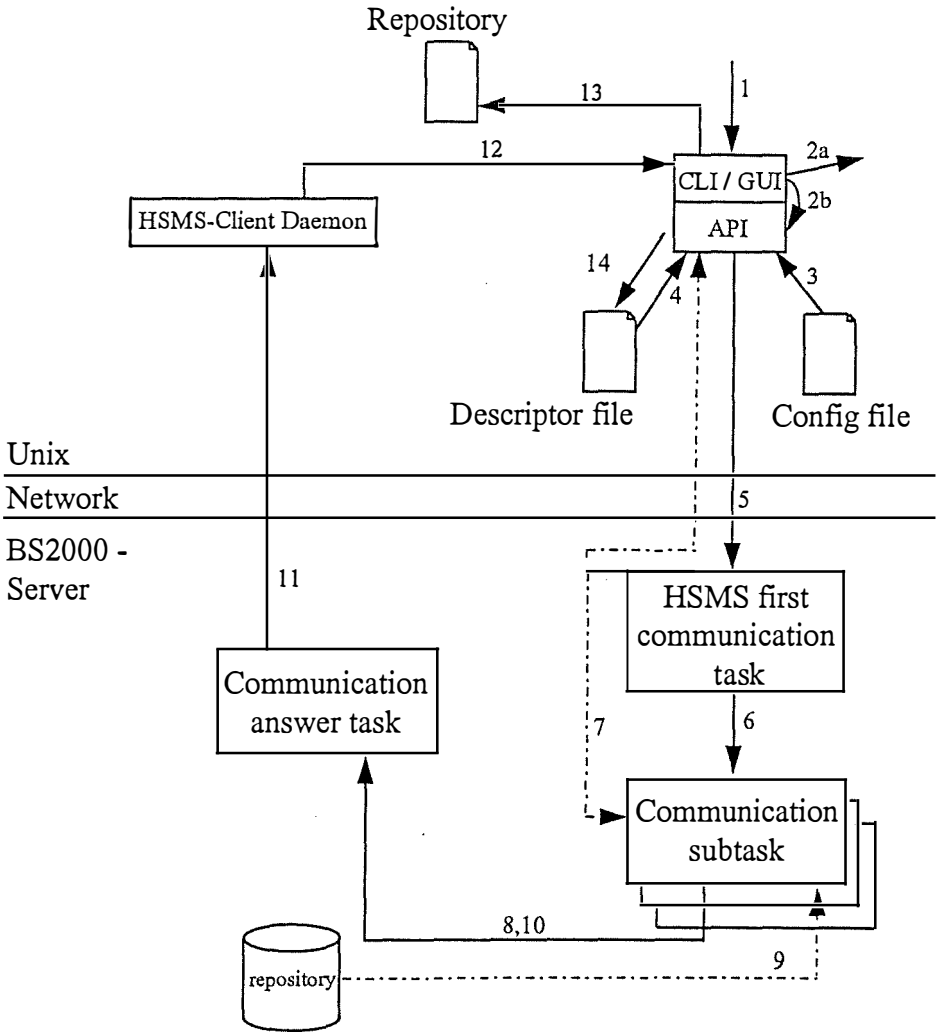


Figure 25 : Periodic update obtaining architecture

1 The command requesting the update is started on the workstation. A first local syntactic check occurs on this statement ; it is only a formal check, not a detailed one.

2a If the statement contains an error, the process is stopped and an error message is sent to the user.

2b If the statement is correct, the corresponding API function is called.

3 The descriptor and the repository file names is extracted from the configuration file.

4 The SVID and the SFID are extracted from the descriptor file.

5 The API function then builds a structure describing the command following the HSMS client-server protocol and sends this data on the network to the BS2000 HSMS server.

6 On the BS2000 side, the connection event is received by the HSMS first communication task, which immediately redirects it to a free communication subtask.

7 The communication subtask reads the data from the network and checks whether it complies with the HSMS client-server protocol.
If the protocol is correct, the subtask builds a control block which contains all the network-related data of the remote command, then translates the structure into a HSMS BS2000-like statement before executing full syntactic and semantic checks.

8 If the statement is refused, the control block is passed to the Communication answer task with a negative return code, and the subtask returns idle.
If the statement is accepted, it is translated into a 'request' which is then processed.

9 The statement is processed by the communication subtask itself, which opens the corresponding repository to find all the information related to the received SVID and SFID (F, R, S, X records).

10 The result and the control block are then passed to the Communication answer task.

11 When the Communication answer task receives a control block, it opens a network connection to the HSMS-CL daemon of the UNIX workstation and sends the result.

12 The HSMS-CL daemon directly transfers the result to the requesting process.

13 The repository is updated with the received information.

14 The contents of the descriptor that has just been updated is deleted.

7.4.3 Restore actions

For the restore operations, the decentralization of the repository leads to no changes to the processing of the requests. The consultation of the repository is made to find where the requested files are stored, and thus is only useful on the server, processing this consultation on the workstation leading to no advantages.

7.4.4 Show actions

For the show actions, the processing of the requests is totally changed. It is important to differentiate the two possible cases that can arise. These two cases are :

- consultation of updated records
- consultation of non-updated records

Updated records

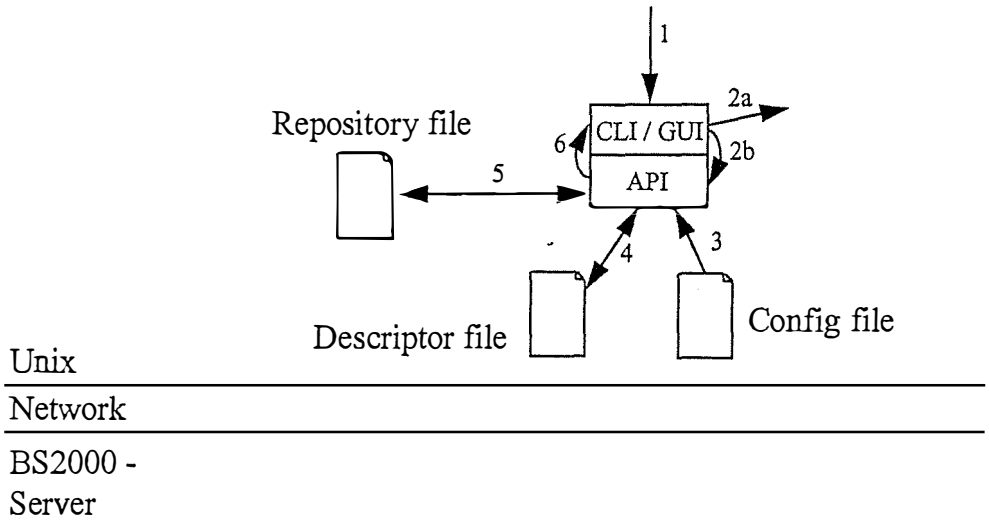


Figure 26 : Bsshow on updated records

1 A UNIX user issues a HSMS-CL statement on his workstation. A first local syntactic check occurs on this statement ; it is only a formal check, not a detailed one.

2a If the statement contains an error, the process is stopped and an error message is sent to the user.

2b If the statement is correct, the corresponding API function is called.

3 The descriptor and the repository file names are extracted from the config file.

4 The information contained in the descriptor file is compared with the requested one.

5 As the comparison has given the result that the requested information is up to date, the local processing of the request can continue. A possible check can occur at this point and next the requested information are extracted from the repository. The access permission to these information is checked.

6 The result, empty if the user has not the rights to access to these records or the requested information, is transferred to the user.

Non updated records

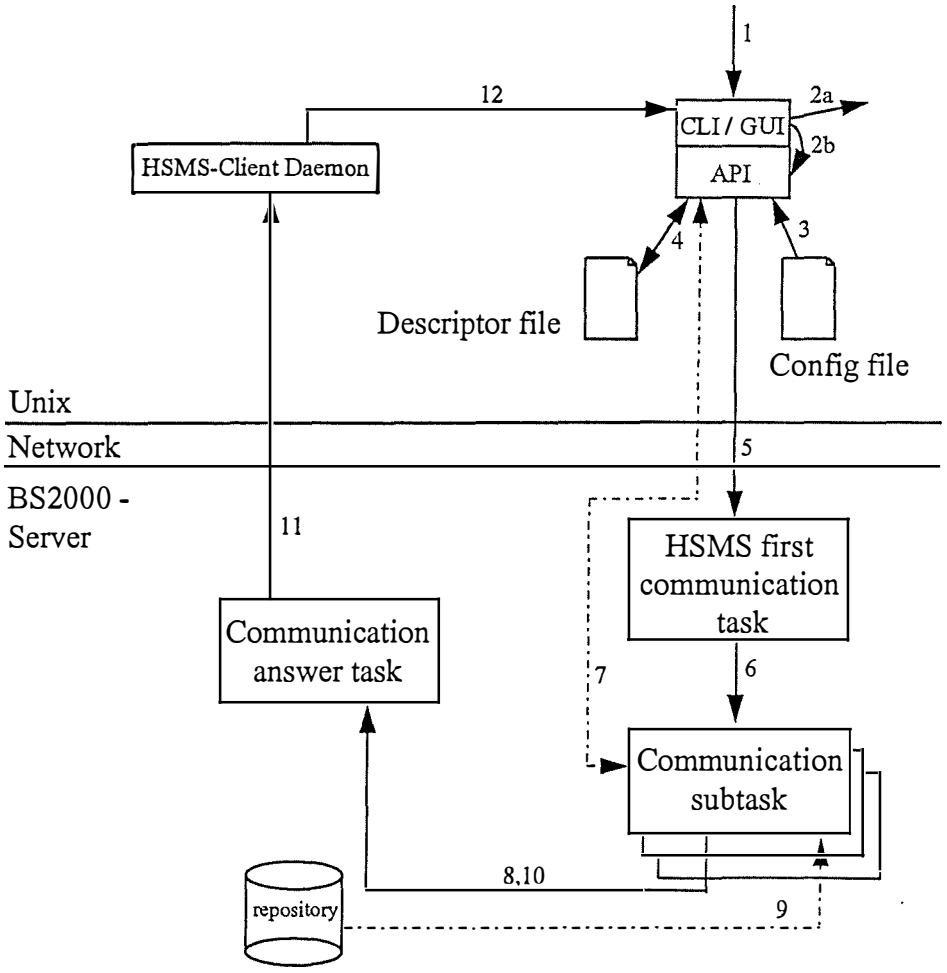


Figure 27 : Bsshow on non updated records

1 A UNIX user issues a HSMS-CL statement on his workstation. A first local syntactic check occurs on this statement ; it is only a formal check, not a detailed one.

2a If the statement contains an error, the process is stopped and an error message is sent to the user.

2b If the statement is correct, the corresponding API function is called.

3 The descriptor and the repository file names are extracted from the config file.

4 The information contained in the descriptor file is compared with the requested one.

5 As the comparison has given the result that the requested information is not up to date, the request is transferred to the BS2000 server.

6 On the BS2000 side, the connection event is received by the HSMS first communication task, which immediately redirects it to a free communication subtask.

7 The communication subtask reads the data from the network and checks whether it complies with the HSMS client-server protocol.
If the protocol is correct, the subtask builds a control block which contains all the network-related data of the remote command, then translates the structure into an HSMS BS2000-like statement before executing full syntactic and semantic checks.

8 If the statement is refused, the control block is passed to the Communication answer task with a negative return code, and the subtask returns idle.
If the statement is accepted, it is translated into a 'request' which is then processed.

9 The statement is processed by the communication subtask itself, which opens the corresponding repository to find the requested information.

10 The result and the control block are then passed to the Communication answer task.

11 When the Communication answer task receives a control block, it opens a network connection to the HSMS-CL daemon of the UNIX workstation and sends the result.

12 The HSMS-CL daemon directly transfers the result to the requesting user.

7.4.5 Changes explanations

Due to the decentralization of the repository, all the software directly involved in the backup processes, i.e. HSMS-Client, HSMS and ARCHIVE, must be changed.

Backup/archive actions

For the backup operations, the changes concern the generation of the descriptor and its recording on the workstations.

For the generation of the descriptor, the question is to know which, from HSMS server task or ARCHIVE main task, will create it. The problem is that both have a part of the information that must be set in the descriptor. If it is the HSMS server task that generates it, the information about the SFID and the SVID must be transmitted from the ARCHIVE main task. In the opposite case, all the information concerning the location of the descriptor (file name or memory location) must be transmitted to the HSMS server task in order to allow this one to send it to the workstation. The two options exist and the solution to this problem can only be found during a deeper analysis of the changes involved by the decentralization.

For the recording of the descriptor, it requires a change to the HSMS-CL daemon. This one must check, at the reception of a descriptor, in the config file what is the name of the file in which the received information must be stored. It involves that the descriptor contains an indication of which archive is concerned by it (i.e. if the descriptor concerns the archival or backup repository).

Another problem arises with the backup/archive operations : it's also possible to generate a command on the BS2000 that concerns the backup of the node-files (backup-node-files, archive-node-files). In this case, the server must first know which are the workstations must receive the descriptor and then must have a new task to send the descriptors to the workstations. This new task is required because there is no report sent to the clients, the descriptor cannot be added to the report. And as there is no control block related to a workstation, there is absolutely no information available to send them.

So, this case requires a new mechanism, but due to lack of time and mainly to the complexity of this mechanism, it will not be studied here.

Update obtaining

For the update obtaining, a new command is required. This new command must get the descriptor file name from the config file, extract the contents of this one and then ask to the server all the needed information. To get the information from the server, the new command can be considered a combination of three successive bsshow :

- one to get the save file information
- one to get the save version information
- one to get the node-files information

It is for these three bsshow that it is necessary to have the SFID and the SVID in the descriptor.
When all the requested information are received on the workstation, the command must execute the update of the repository itself. Once the update is completed, the contents of the descriptor must be deleted to avoid duplication in the updates.

Show operations

The show operations must be changed too. The show command must first consult the config file to get the name of the requested repository and its descriptor.
Then, it must extract the contents of this descriptor and match the obtained information with the one given with the command.

After receiving the result of this comparison, the access is made on the workstation repository (if there is no matching), or is redirected to the server repository.
The local access to the repository requires the implementation of the same security and data privacy mechanisms that exist on the server.

8. POSSIBLE EVOLUTION

8.1 INTRODUCTION

The decentralization of the repository opens several possibilities for the evolution of HSMS. Some of these evolutions are useful to avoid problems, other are useful to accelerate the processing of some commands.

The first possible evolution concerns the processing of the incremental backups : the decentralization of the repository offers the possibility of speeding up the processing of such operations.

The second possible evolution concern the repository itself : it consists in a tool to check the contents of it.

A third possible evolution consist to have several archives of the same type for the workstations (in the present situation, there is only one backup and one archival archive for the workstations).

8.2 INCREMENTAL BACKUPS

8.2.1 Present situation

HSMS and HSMS-Client offer the possibility to make incremental backups in place of full backups, i.e. a backup of only what has changed since the last backup
(full or incremental).
During incremental runs, the files that have not changed are also recorded in the repository. They are recorded with a special mark that indicates that they were present at the time of the backup run but that they have not changed. The mark is 'Cataloged-not-saved' (CNS). It is used during the restore operations.

Following a parameter in the restore command, the files that are marked with CNS are restored or not.

Example

The files have been saved as follows by various backup runs :

Filenames	Save version name			
	BACKUP01	**BACKUP02**	**BACKUP03**	**BACKUP04**
File.1	Full	Full	Full	CNS
File.2	Full	CNS	Full	
File.3	Full			Full
File.4	Full			

Where Full and CNS indicate the save type of the file.

Action statement	Restored Files	From save version
bsrest -sv	File.1 File.3	BACKUP03 BACKUP04
bsrest	File.1 File.2 File.3 File.4	BACKUP03 BACKUP03 BACKUP04 BACKUP01
bsrest -sv BACKUP02	File.1 File.2	BACKUP02 BACKUP01

As shown in the example, the files that are marked CNS are not always restored. During restore, all save versions existing in the referenced archive are selected for processing by default. To select only the latest save version, it is necessary to specify the optional parameter **-sv**.

For restore operations on all save versions, the data are taken, for each file, from the last save version in which they are saved.

For restore operations on the latest save version only, there are only the files that have been actually saved or marked CNS in this save version that are restored. The CNS marked files are restored by taking the data from the last save version where there are fully saved.

And for restore operations on a specified save version, the data are taken for each file either from this save version if the file is fully saved in this one, or from the first previous save version in which the file is not CNS marked if it is CNS marked in the specified one.

Architecture

In the schema describing the architecture, when the ARCHIVE main task builds the list of files to be processed, for an incremental backup, it compares the metadata of the files with those contained in the repository before building the list.

8.2.2 Evolution

As it can be seen, the incremental backups require, in the present situation, the transfer of all the files information from the clients to the server before executing the comparison. With the decentralization of the repository, this comparison can be executed on the workstation, and thus a part of the traffic required by the backup run be avoided, leading thus to greater performance. The direct traffic on the network is reduced for the incremental backup as only the filenames that must be actually processed are transferred, and not all the metadata associated with the files.

But there is a drawback to this : the CNS marking of the non changed files can not be executed on the server, as this one does not know which are the files in this situation.
Thus, these file names must be transferred to the server. For that, they must first be recorded during the comparison on the workstation (Remind the assumption that the central repository must be the first updated). The transfer must take place just before the update of the central repository, so all the information is written in one time.

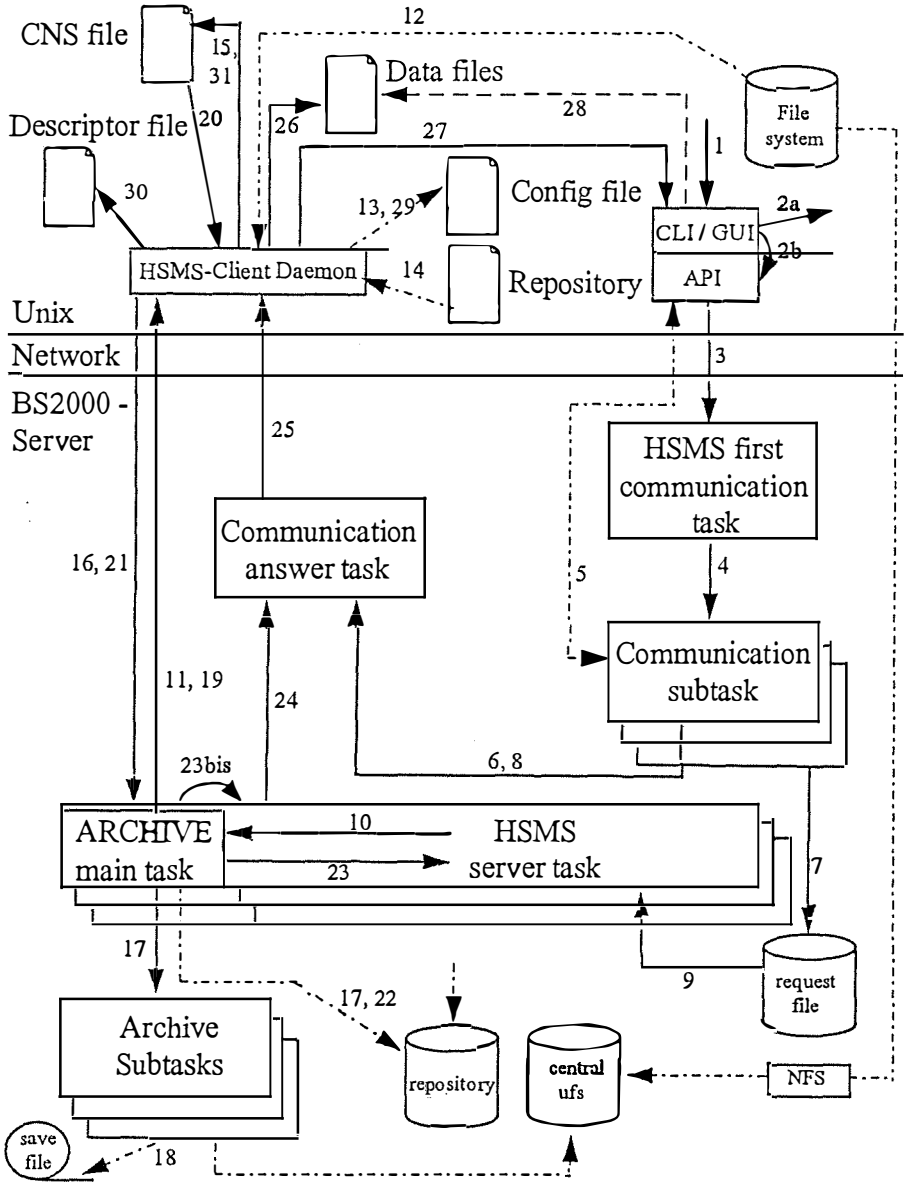


Figure 28 : HSMS architecture with CNS verification on the client

1 A UNIX user issues a HSMS-CL statement on his workstation. A first local syntactic check occurs on this statement.

2a If the statement contains an error, the process is stopped and an error message is sent to the user.

2b If the statement is correct, the corresponding API function is called.

3 The API function then builds a structure describing the command following the HSMS client-server protocol and sends this data on the network to the BS2000 HSMS server.

4 On the BS2000 side, the connection event is received by the HSMS first communication task, which immediately redirects it to a free communication subtask.

5 The communication subtask reads the data from the network and checks whether it complies with the HSMS client-server protocol.
If the protocol is correct, the subtask builds a control block which contains all the network-related data of the remote command, then translates the structure into an HSMS BS2000-like statement before executing full syntactic and semantic checks.

6 If the statement is refused, the control block is passed to the Communication answer task with a negative return code, and the subtask returns idle.
If the statement is accepted, it is translated into a 'request' which is then processed.

7 The subtask writes the request into the request queue.

8 It then passes the control block to the Communication answer task with a return code 'accepted' and finally return idle.

9 The request waiting in the request queue is eventually taken and processed by an HSMS server task.

10 The server task reads the request and calls the ARCHIVE product via a private interface. The task is then called 'ARCHIVE main task'.

11 The ARCHIVE main task asks the HSMS-Client daemon the complete list of the files to be processed.

12 The HSMS-Client daemon gets the information about the concerned files from the workstation's file system.

13 It accesses the HSMS configuration file to get the name of the involved repository.

14 The HSMS-Client Daemon access the repository to get information about the concerned files.

15 The daemon builds the list of the files to be processed by comparing the data from the file system and from the repository. It then stores the names of the other ones in a file, the CNS file.

16 The list of the files that must be processed is transferred to the ARCHIVE main task.

17 The ARCHIVE main task creates an ARCHIVE subtask and transmits to it the list of files. It is possible to have several subtasks created by the main task to work in parallel on a part of the list in order to obtain better performance.

18 The subtasks are responsible for the physical access to the storage media.

19 Once all the subtasks are finished, the ARCHIVE main task asks to the HSMS-Client daemon the names of the files that must be marked CNS.

20 The daemon reads the names from the CNS file.

21 The result is then transferred to the ARCHIVE main task.

22 The ARCHIVE main task updates the concerned repository.

23 The ARCHIVE main task returns to HSMS.

23bis The descriptor is generated with the SFID, the SVID and the pathname.

24 The HSMS server task generates a report which is passed with the control block and the descriptor to the Communication answer task

25 When the Communication answer task receives a control block, it opens a network connection to the HSMS-CL daemon of the UNIX workstation and sends the result and the descriptor.

26 On the workstation, the daemon receives the result and stores it in a file.

27 If the user process is still waiting, the result is directly fetched to it, and it is displayed.

28 If no user process is waiting, the result stays available and the user may read it at any time by issuing a specific HSMS-CL command.

29 From the configuration file the daemon gets the name of the file where the descriptor must be saved.

30 The daemon stores the descriptor in the corresponding file.

31 The daemon deletes the contents of the CNS file.

8.2.3 Changes and choices explanations

Changes

The changes mainly concern the building of the files list : in place of being executed on the server, the comparison and the building are processed on the clients.
It requires thus the adaptation of the HSMS-Client Daemon : new functions must be added to execute this processing. The other change concern the transfer of the CNS file : the ARCHIVE main task must ask them to the workstation before updating the repository. This transfer takes place at this moment to have the correct information as quickly as possible in the repository and a complete report also (containing all the information).

Choices

Concerning the choice that has been made of non updating the workstation's repository, it is for reasons of easiness. As the decentralized repositories are not updated with the CNS information, the update obtaining does not have to be changed for the incremental backups. If the CNS information is recorded in the repository, it requires to have special mechanisms to filter the information during the normal update to avoid duplication, or it requires to have a special update procedure only for this type of backups. It is not very economical in term of traffic on the network, but it has the advantage that it requires few changes to the architecture

8.3 REPOSITORY ANALYZER

In order to help to discover possible errors and inconsistencies in the decentralized repositories, it is necessary to have a special tool to analyze the contents of these ones.

This tool, the repository analyzer, is used to verify the relations that exists between the different fields of the records contained in the repository (see Figure 29).

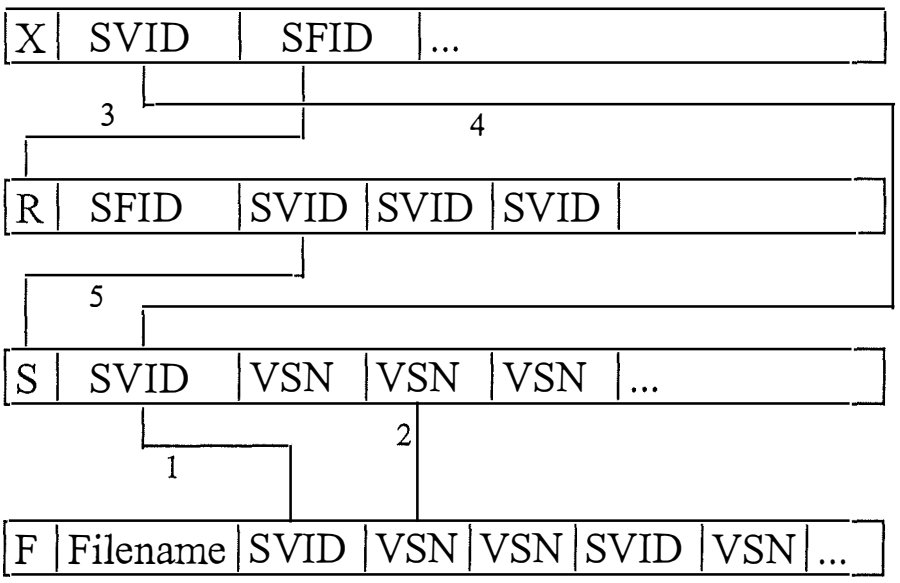


Figure 29 : Records relations

Relations between F and S records

1 For each SVID found in the F record, a S record must exist with the same SVID

2 Each VSN found in the F record must have its counterpart in the S record corresponding to the SVID of the VSN.

Relations between X and R records

3 For each X record, a SVID and a SFID are to be found. It should be verified that the SVID can be found in the R record which has the SFID in its key

Relations between X and S records

4 It should be verified that for each X record, there is a S record with the same SVID

Relations between R and S records

5 Every R record contains a list of SVID. It should be verified that each of them corresponds to a S record.

The repository analyzer only warns that there is inconsistencies in the repository, it does not repair them. When errors are discovered, it is necessary to launch the recovery procedure to solve the problem. The tool helps to discover some of the possible errors that can exist in the repository, but other errors requires the attention of the users or the administrator. They must warn if they get false or incomplete information when they access to the repository.

The recovery procedure, when launched, must delete the contents of the repository if it is not empty and reconstruct it with the information received from the server.

8.4 ARCHIVE SELECTION FROM CLIENT

In the present situation, for a backup, an archival, a restore or a show action issued from a client, it is impossible to select the archive concerned. It is always the default archive that is used (sysnodebackup or sysnodearchive).
When the same command are issued from the BS2000 and concern the node files, the selection of the archive on which to work is possible.

A possible evolution of the present product is thus to allow the selection of the archive from the UNIX workstations as in BS2000. This evolution involves several changes to the HSMS-Client commands and can also lead to the adjunction of new HSMS-Client commands or lead to the modification of existing HSMS commands.

8.4.1 Commands changes

The commands changes consist in adding a field to the commands. This field will contain the name of the archive on which the operation must be processed.
The commands concerned by this modification are the archival, backup, restore and show command (bsarch, bsback, bsrest and bsshow) and their sisters in the graphical user interface (GUI) and the application program interface (API).

Moreover, the archive selection also involves a change to the GUI. It must be possible to select the archive where the backup or archival operation must take place, and also the archive from where the restore must be made. It implies thus changes to the interface itself in addition to the changes to the called functions.

8.4.2 Possible adjunction

The possible adjunction to the already existing commands consists in adding commands concerning the archives manipulations. In the present situation, the creation, the modification and the suppression of the archives can only be made from the BS2000.
As only the owner of an archive can use it, either the workstation users must have an access on the BS2000 to create their own archive, or these commands must be implemented to allow the creation of the archive from the workstations.

9. CONCLUSION

The object of this work was to study the decentralization possibilities of a repository in a particular distributed environment, and to propose a solution to solve the problems of the present implementation of the repository.

Due to the specificity of the distributed environment, none of the theoretical decentralization models found in the literature can be applied, but through some adaptations and modifications of these models, a solution has been found. The repository will be partially replicated, partially partitioned and also be kept centralized. With this implementation of the repository, the performance problems are solved as the client workstations and the server have good performances (the performances were a key argument in the choice of the implementation model). And also important, the required disk space is not too important as only partial information is replicated. Another problem with the decentralization of the repository was the update information that must be propagated from the server to the clients. With the chosen solution for this problem (use of a short descriptor), the traffic required for this is kept at a low level. The chosen implementation has a little disadvantage : there is a loss of information in the decentralized repositories (the decentralized information is not complete). But this loss is not disturbing because the lost information, if required, can be found on the server, and because the lost information on the workstations will never be accessed on these places.

Despite the modifications brought to the theoretical models, the implementation of the repository keeps all the distribution properties :

- **Distribution transparency** : the only change that can appear on the workstations is the response time that will be reduced. The interface is kept consistent and no other change appears.
- **Site autonomy** : as each site holds all the necessary information to access its data, this property is completely fulfilled.
- **Efficiency** : all the information is stored close to its point of use (on the server or on the concerned workstation).
- **High reliability/availability** : The aspect of the reliability has not change from the previous implementation, thus remains at the same level. The availability has been increased as the client application remains operational at a local level if the network is partitioned or other sites crashed.
- **Security/access control** : as the accesses to the file containing the repository will only be allowed through the use of the dedicated application, the access control and the security requirements are also fulfilled.

10. GLOSSARY

active client : workstation with the HSMS-CL software installed.

archival : Long-term saving of files that are no longer required. The files are deleted from the processing level once they have been backed up.

archival archive : HSMS archive used for archival.

ARCHIVE : BS2000 software product which saves files logically. ARCHIVE has an internal interface with HSMS and implements the HSMS action statements.

archive : Management unit for files under HSMS management, consisting of the archive definition and the associated repository. HSMS makes a distinction between five archive types. There are archives concerning DMS files : backup archives, long-term archives, migration archives. The other archives are used to save node-files : node backup archives, node long-term archives. Furthermore, HSMS distinguishes between private archives, which may be accessed by the archive owner only, and public archives, which are available to all users.

archive directory : File used for managing the objects saved in an archive, i.e. files, save files, save versions and the volume pool, and implemented as an ARCHIVE directory file (cf. ARCHIVE).

archive owner : Users have the right to create archives by means of the CREATE-ARCHIVE statement. The creator of an archive is the archive owner. The option of making the archive created available to other users is restricted to the HSMS administrator.

archive type : Determines the basic HSMS function for which an archive is to be used.

backup : The periodic creation of copies of the data inventory to permit the restoration of data lost due to hardware errors or inadvertent deletion, etc. Can also be used to reorganize disk storage.

backup archive : HSMS archive used for backup.

BS2000-UFS : UNIX file system mounted on a BS2000 server.

Cataloged-Not-Saved : Indicates that a file was not saved either because an incremental backup was performed and the file had not been changed since the previous backup, or because an error (e.g. open error) prevented it from being saved.

CLI : Command Line Interface

CNS : Cataloged-Not-Saved

control file : File that contains the HSMS control parameters and the archive definitions.

default system archive : Archive assigned globally to the entire system or to disks and accessed unless another archive is specifically specified. There is a separate default system archive for each of the basic HSMS functions migration (SYSMIGRATE), backup (SYSBACKUP) and archival (SYSARCHIVE), node backup (SYSNODEBACKUP) and node archival (SYSNODEARCHIVE).

GUI : Graphical User Interface.

HSMS : Hierarchical Storage Management System: BS2000 software product offering such functions as migration, backup, archival, and data transfer, implemented in a storage hierarchy and in archives.

HSMS administrator :User enjoying the HSMS administrator privilege. The HSMS administrator can use all HSMS functions without restrictions. The following are typical HSMS administrator tasks: managing the storage hierarchy, creating the default system archives, system backup and control of tape processing.

HSMS-CL : Client version of the HSMS software, running on UNIX workstations.

HSMS-SV : Server version of the HSMS software, needed to backup and archive the UNIX workstations files.

H-VSN : Identifier of a Tape-record. Equals to the highest volume identifier contained in the record.

implicit recall : Automatic recall of migrated files as a result of attempts, on the part of DMS, to access these files, as opposed to recall requested via a statement.

ISAM : Indexed Sequential Access Method.

level : cf. storage hierarchy

long-term archive : HSMS archive used for archival.

migrated file : A migrated file is a file whose data has been deleted from the processing level but whose catalog entry remains on this level. The catalog entry indicates the background level to which the data was migrated.

migration : Moving inactive files from the processing level to a background level without deleting the catalog entry.

migration archive : HSMS archive used for migration.

NFS : Network file system. Access method to the file systems and their contents.

node : Instance (workstation, PC, ...) connected to a network, and for which the file systems can be processed by HSMS when HSMS-SV is present. node archival archive
HSMS archive used for the archival of node files.

node backup archive : HSMS archive used for the backup of node files.

node file : file located on a node

passive client : workstation without HSMS-CL installed.

pool : cf. volume pool

recall : To move migrated files back to the processing level S0.

reorganization : HSMS function used primarily to reorganize the migration archive, i.e. to reshuffle save files without transferring invalid files.

restore : To move data from an HSMS archive back to the processing level S0.

retention period : Period of time during which data modification or deletion is prohibited. The (physical) retention period prevents save files and save volumes from being overwritten during this time, while the (logical) retention period defined by the file expiration date prevents files from being modified or deleted.

S0 : Normal online processing level, implemented by (high-speed) disk storage.

S1 : Online background level, implemented by disk storage (possibly with more capacity and longer access times than S0).

S2 : Off-line background level implemented by archives on magnetic tape or tape cartridge.

save : Used as a synonym for backup.

save file : "Receptacle" for saved files. The save file contains one or more save versions and consists of a set of volumes which all have the same owner and retention period. Each save file is identified by a save file ID (SFID) formed by the date and time of its creation.

save version : Result of a backup or archival request. The save version is internally identified by a save version ID(SVID). The user can refer to it via its creation date or the name assigned to it at creation.

SFID : Identifies a save file; the save file ID has the following format: S.yymmdd.hhmmss

SVID : Identifies a save version; the save version ID has the following format: S.yymmdd.hhmmss

storage hierarchy : Assignment of storage units to different storage levels, depending on their availability, access time and storage costs (cf. S0,S1,S2).

storage level : cf. storage hierarchy

SYSARCHIVE : Default system archive for archival.

SYSBACKUP : Default system archive for backup.

SYSMIGRATE : Default system archive for migration.

SYSNODEARCHIVE : Default system archive for archival of node files.

SYSNODEBACKUP : Default system archive for backup of node files.

system archive : cf. default system archive

TAPE : Volumes of the class "TAPE" are assigned to storage level S2, these are magnetic tapes as well as magnetic tape cartridges.

UFS : UNIX File System.

volume pool : Set of volumes managed by an archive and registered in the directory. Volumes required for save requests are normally fetched from the free volume pool of the archive.

VSN : Identifier of a tape volume.

11. BIBLIOGRAPHY

Dr. Michael A. BAUER, Dr. J. Michael BENNETT, Dr. Jacob SLONIM. *a Conceptual Framework for Distributed Directories.* Technical report 240, The University of Western Ontario, June 1989. Department of Computer Science, Distributed Directories Laboratory.

Dr. Jacob SLONIM. *The Role and Use of Data Dictionaries.* Technical report 244, The University of Western Ontario, June 1989. Department of Computer Science, Distributed Directories Laboratory.

Dr. Michael A. BAUER. *Distributed Directories : a Conceptual Framework (a Presentation).* Technical report 245, The University of Western Ontario, June 1989. Department of Computer Science, Distributed Directories Laboratory.

Dr. Jacob SLONIM. *Distributed Database Management Systems (a Presentation).* Technical report 247, The University of Western Ontario, June 1989. Department of Computer Science, Distributed Directories Laboratory.

C.J.DATE. *An Introduction to Database Systems Volume II*, chapter 7 : Distributed Databases pp 291-332. Addison-Wesley Publishing Company 1983

Sam COLEMAN, Steve MILLER. *Mass Storage System Reference Model : version 4.* IEEE Technical Committee on Mass Storage Systems and Technology. May 1990

Souleymane Bah. *File migration in a distributed environment.* Facultés Notre-Dame de la Paix Namur, June 1996. Institut d'Informatique.

External Interface Specifications for HSMS-CL

HSMS / HSMS-SV V2.0B. Hierarchical Storage Management System : User Guide. July 1995.

HSMS Solution Studies for Backup Services V1.0, September 1993.

Minutes from Internal Formation about the HSMS and ARCHIVE repositories.
HSMS-Hierarchical Storage Management System : Brief Description. March 1994

BS2000 as Backup Server in an Open Universe. Company-wide backup with HSMS V2.0 Brief description. July 1994.

Technical Description : Archive

Hierarchical Storage Management System - HSMS.