MASTER IN COMPUTER SCIENCE

Management classes in heterogeneous distributed storage management

Donkels, Daniel

Award date:
1997

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

MANAGEMENT CLASSES IN HETEROGENEOUS DISTRIBUTED STORAGE MANAGEMENT

Daniel M. Donkels

Advisor : Professor Jean Ramaekers

Dissertation submitted in conformity
with the requirements for the degree of
'Licencié et Maître en Informatique'

June 1997

Abstract

In the last years, computer systems have become more and more decentralized and heterogeneous. A big disadvantage of these heterogeneous distributed systems is that storage is very difficult to manage. In this study, we're going to develop the theoretical concepts and mechanisms of a storage management system in a heterogeneous environment. An aspect that will particularly be considered is the data backup.

To do this, we're going to develop the central concept of management classes, a collection of attributes, that will be used to configure one for all the processing policies for the objects to which these management classes are linked.

Condensé

Au cours des années, les systèmes informatiques sont devenus des plus en plus décentralisés et hétérogènes. De tels systèmes ont pourtant le grand désavantage que l'espace de stockage y est difficile à gérer. Dans le présent travail, nous allons développer les concepts théoriques et les mécanismes de base pour un système de gestion d'espace de stockage dans un environnement hétérogène distribué. Un aspect que nous allons particulièrement traiter est le backup.

Pour cela, nous allons développer le concept de 'management classes', un ensemble d'attributs, qui va être utilisé pour spécifier une fois pour tout la manière dont les objets assignés à ces classes seront traités.

Acknowledgments

First of all, I want to thank Professor Jean Ramaekers, promoter of this thesis, for his beneficial comments on this document and the complete freedom of action he gave me.

I thank Benoît Hucq from Siemens-Nixdorf Software S.A. Namur for providing the subject of this thesis and Fabian Libion for his advice, corrections and helpful guidance during my study. I also want to thank all the people from Siemens-Nixdorf Software that by their explanations gave me help and assistance.

Finally, I would like to thank all the persons that showed interest in this thesis and particularly Pascal Goossens for the comments and constructive critics he gave me.

Table of contents

0. Introduction

System administrators have the task to manage the computer system and to assure that everything works in accordance with the users' objectives. How these objectives can be attained is generally expressed under the form of policies which are interpreted by the administrators.

Humans are very strong at management because they are not only able to interpret informal policies but they can also solve conflicts and react whenever something unforeseen happens. One could think that these qualities make the human the perfect system manager, but unfortunately, computer systems have become so large and so complex, that it's impossible to manage them completely by hand. Thus the idea of automating parts of the management, which requires however formalized policies. A possible formalism would be the management class concept.

A computer system can be managed according to different points of view. There is for instance the security management which specifies which users have access to which resources whereas the performance management assures that the system's computing power is used in the best possible way. One aspect of computers systems that grows very fast is the storage space and it becomes thus more and more difficult to manage. This is mainly due to the fact that the mainframe computers with their centralized storage are more and more replaced or supported by other types of machines. Nowadays, most companies have a heterogeneous distributed system where there is coexistence of different kind of platforms such as mainframes, UNIX machines and personal computers. In such an environment, storage management is even more difficult. In this study, will be discussed how the management class concept can be used to simplify this task.

The work is divided as follows : in the first chapter, storage management is presented and in the second, we present three existing products in which the management class concept is more or less developed.

In the third chapter, we define our view of management classes as well as the potential profits of the concept. In chapter four will be developed an additional concept, the entity structure, on which the management classes can be applied. The chapter also contains a rather complete example of such a structure. Chapter five will explain the attributes of backup management classes and contains an example of how management classes should be used by the system. The sixth chapter contains some practical considerations of how such a management class system could be realized.

Our personal part begins with the third chapter and this does also explain the rather limited bibliography. Most of the listed titles concern in fact only the first two chapters. This study doesn't base on any particular research project and that's why there was no further documentation available.

1. Storage management

The main purpose of this study is to discuss the use of management classes for storage management and so it might be useful to recall in what storage management actually consists. In this chapter we're thus going to introduce the three aspects of storage management, i.e. backup, archival and migration.

1.1 Backup

Backup consists in making an insurance copy of data in case the original medium should fail. The copy is put on a different medium than the original data. Very often, it's a removable medium such as a tape or an optical disk and the copy is stored in a secure place that is a different one than the location of the original data. By storing several copies of the same data in different places, data security can considerably be increase. Backup does not change the original data, nor does it free up any storage space but it even makes more data [Mango, 94].

These insurance copies are used to recover files that have accidentally been deleted or that have become corrupt. Whenever there has been a disk crash, a virus or any other major problem, the backup copies can be used to recover the system. As backup creates more data and makes a lot of work, it's an expense but it's also an important part of disaster recovery plan and that's why people accept to do it.

With backups, it's very important that backup copies are as up to date and as complete as possible to minimize the work it takes to restore the system as it was at the moment of failure. If recovery takes too long, this can become very expensive. Not only because of the recovery work but also because of all the users that can't work during that time.

Computer systems are changing continuously and nowadays, they're becoming more and more decentralized. This means that the workstations have their own disk with their own applications and their own data. In centralized systems, data storage and data backup were centralized but in decentralized system, data backup is done by the users who are responsible of backing up the data on their workstation. This decentralized backup has some inconvenients:

First of all, backup takes a lot of time. Users loose their time doing work that has nothing to do with their real task. Because backing up is not their main task, lots of users consider that it's not so very important and that's why very often, backups are not done regularly, but only when there is some time left.

Another inconvenient is that very often, users have no tool that is powerful enough to make good backups and they don't also have the same knowledge than a system administrator. In fact, making good backups is not as easy as it seems. Backup

must be done at the right time, versions and mediums have to be managed, copies have to be stored in a save place and so on.

There are enough opportunities to make errors and backup errors can be fatal. Nowadays, information has become a major economic factor. If now a company looses its vital data and there's no possibility to recover it, the consequences can range from major financial losses to having to give up business.

Everybody sees that letting backup under the only responsibility of the users is much too risky. In fact, it has to be done by some professional, the system administrator for instance, or by some automated backup tool.

1.2 Archival

Archival is used for data that is normally not needed anymore or not needed for a long time. This means that the data will not stay in the on-line storage space but will be put on some special off-line storage. This liberates the on-line storage space the data occupied before and reduces the storage cost.

Rather than deleting the old data that isn't needed anymore, most firms prefer to archive it in order not completely to loose the history of their activities. It's always possible that someday, the old data could be needed again. On the other hand, the storage medium used for archival, magnetic tapes or cartridges, are not very expensive.

For most companies, there are even legal obligations to retain certain data, such as the accounting information, up to ten years or longer. Whenever the data is needed for some reason or the other, it can be recalled onto the on-line level.

1.3 Migration

Nowadays, the data volume grows from day to day. It's especially since there are graphical user interfaces, that the size of applications have exploded. Pictures are used on a large scale, and this needs a lot of space. A 50 pages Microsoft Word document with graphs and pictures can easily have up to 5 megabytes and more. This was completely unimaginable several years ago and this development hasn't yet reached it's climax. In the time of multimedia, sounds and animations have again considerably increased the need of storage space. Storage space is thus always a rare resource, especially the fast on-line storage.

Migration brings a solution to this storage space scarcity by moving parts of the data to slower storage medium [Mango, 94], [Intel, 96]. For this, we can categorize storage into different levels : The highest level consists of expensive disks with very fast access time and important transmission rates whereas the lower levels consist of slower, but also less expensive mediums such as slower disks or tape. Data that's accessed frequently stays on the top level and data that hasn't been accessed since a certain time, is moved to a lower level.

This gives people the impression of unlimited storage space. Migration is completely transparent and users don't know where their data is actually located. Data is automatically migrated and when it's accessed, it's also automatically recalled. For files with which users normally work, and which are thus on the top

level, there is no deterioration of access time. Whenever some migrated data is accessed, recall can however take a certain time depending on the level it's on. This is the price to pay for nearly unlimited storage space.

2. Management classes in existing products

Management classes are not a new concept but they have been implemented in backup products for several years. There is of course Siemens-Nixdorf's HSMS from which we've taken the name of management classes but there are also other products in which this concept or a similar one exists under the one or the other name. We will now have a close look at several of these products to see how they have implemented the concept and how it's used.

2.1 HSMS

2.1.1 Brief product description

HSMS is an acronym for Hierarchical Storage Management System. It's a BS2000 (Siemens-Nixdorf's mainframe operating system) software product which supports data management on external storage devices in a BS2000 environment [Sieme, 96a], [Sieme, 96b]. It's basic functions are backup, migration and archival.

2.1.1.1 Hierarchical Storage

At the name of the product we can see that hierarchical storage is one of the most important concepts of HSMS. This means that HSMS implements an hierarchy of storage levels according to storage costs and access time.

HSMS's different storage levels are:

1. S0: The processing level

This is the normal storage level which users can access on-line. The data on this level is directly accessible via the file system which in BS2000 is called DMS (Data Management System). Because this is the on-line level, it has to be implemented by disks with very short access time.

2. S1: The background level

This level can be used for storing inactive on-line data, i.e. data that is only required from time to time. This level is implemented by disks that don't have to be as fast as those of the on-line level and are thus less expensive, but that are still much faster than certain storage mediums such as tapes or cartridges. The data on this level is managed by HSMS and the user is unaware of the migration.

3. S2: The archive level

This is the off-line background level that is used for inactive and seldom required data. It is implemented by low cost storage mediums such as magnetic tape or cartridge and the S2 level is thus the slowest storage level of the HSMS hierarchy.

2.1.1.2 Fundamental Functions

2.1.1.2.1 Backup

The HSMS backup function allows it to save files of the processing level S0 and the migration levels S1 and S2. System backup can be carried out by the system or by the HSMS administrator.

HSMS manages backup data resources automatically and puts them in so called system backup archives. HSMS saves files and job variables logically and it offers the options of full backup, differential backup and partial backup.

In full backup, all selected files are saved completely, whereas in differential saving (incremental backup), the catalogue entries are compared to the meta-information of the previous savings and only the files that are new or have been changed since the last saving are backed up. This represents an important economy of storage space and processing time and it is always possible to reconstruct the complete data as it was at the last saving.

In partial backup, if a file has been modified, HSMS checks which parts of the file have been changed since the last full backup. Only these parts are saved. To restore such a partially backed up file, one needs the last partial backup and the last full backup.

A particularity of HSMS is its backup with concurrent copy. This function allows it to make consistent backups of files without rejecting the file modifications concurrently.

Another feature is that HSMS can make a complete backup by doing only an incremental one (full from incremental). This function consists in doing an incremental backup, and then to reconstruct from the last full backup and the last differential savings a full backup on tape without having to access the disks.

HSMS permits to do backups either on disk, tape or cartridge. For unmanned operation, the backup data can be temporarily stored on disk so that it can be transferred to tape at a latter point of time. It's even possible to mix storage forms, i.e. to store the differential backup on disk and the complete backup on tape.

2.1.1.2.2 Archival

The HSMS archival function provides saving of files that are no longer required on the on-line processing level to off-line archives residing on magnetic tape or tape cartridge at level S2 of the storage hierarchy.

HSMS also offers the option of defining archives, and setting characteristics such as maximum archiving period, default values for compression, device type, an so on.

2.1.1.2.3 Migration

Migration is the process of moving inactive data from the processing level to a background level. This is done in order to reduce the danger of saturating the on-line disk storage. Criteria for migrating files can be the number of days since the last access, the minimum file size and the file fragmentation. HSMS automatically recalls the migrated files during file opening or reservation. Migration and recall can also be executed via instructions that the normal user can access. When a file is migrated, the catalogue entry however remains on the processing level.

Some files are normally excepted from migration (temporary files, open files or files that need to be repaired) and the user can also set migration locks on files.

2.1.1.3 HSMS-CL

HSMS-CL is a software product running on UNIX systems. It allows to backup and archive UNIX files through HSMS on a BS2000 server. HSMS-CL is not only a graphical interface for the decentralized platforms but it also helps optimizing the performance by doing a part of the work already on the client platform.

Backup and archival of UNIX clients is also possible without HSMS-CL but it's less performant.

2.1.2 Management class structure

In HSMS, a management class is a named collection of management attributes that are defined by the administrator and used by HSMS to control backup and migration. A set of management attributes is attached to an object by assigning to the object the name of a management class, whose attributes specify policies for backup and migration.

Every management class contains all the attributes for backup and migration but they differentiate by the attribute's values. There are two types of attributes: the first one specifies the prerequisites for an object to become eligible for migration or backup whereas the second one defines the desired result of the backup or migration.

For migration, there are several attributes specifying under what conditions, objects move to which storage level whereas for backup, there is only one object giving the retention time of the backup copy.

Another concept that needs to be mentioned here are the guards. Guards are a kind of protection mechanism that is used to control users' access to management classes. The system administrator uses a special tool to define guards and assigns them to a management class. They specify which user can use this management class for assigning his objects to it. This is very useful if the administrator has defined special management classes to be used exclusively by a particular group of users, such as the management. In this case, the guard that goes with these classes assures that only the management staff and no-one else can use them.

2.1.3 The use of management classes

Depending on the different objects that exist in the system environment, the system administrator defines the management classes and the guards. Then, users can assign their objects to a class of their choice. Users can take no action on management classes but they can only look at them and make assignments.

When the system administrator wants to do a backup or a migration, he must give a management class name within the command and only the objects assigned to this class will be considered. The values of the management class attributes will be used by HSMS to control the processing of the selected objects.

2.2 ADSM

2.2.1 Brief product description

IBM's ADSM (ADSTAR Distributed Storage Manager) is a client-server product that provides storage management and data access services to customers in a multivendor computer environment [IBM, 93]. The different components of this product are the administrative client, the backup-archive client and the server program.

2.2.1.1 Administrative client

The administrative client allows the administrator to control and monitor server activities, define storage management policies (the management classes), set up schedules for regular backup and archival services. It's possible for several persons to share the administration work. The administrator with system privilege can perform any function whereas the other administrators can only perform the subset of administration functions they have been granted.

The module also provides the possibility for the administrator to do backup and archival in a central way. It's possible to specify priorities among the files. This means that the ones that have been declared as more important, are processed first.

2.2.1.2 Backup-archive client

The backup-archive client allows the user to do backup and archival. For backup, there are two possibilities: incremental backup, where only the new and the modified objects are saved, and selective backup, where only the files that have been selected by the user are backed up. During backup, include/exclude lists are used to decide which files have to be saved and to which management class they belong. It's the management classes that specify how the files are backed up.

Users can also archive files that they do not currently need on their workstation, and retrieve the archived files when necessary. Backup and archival can either be triggered by the users or scheduled. It's the backup-archive client that allows the users to define the include/exclude list and to assign their files to the different management classes.

2.2.1.3 Server program

The server program allows a mainframe host to act as the backup and archive server for workstations. It makes it possible to automate backup and archival and to manage the data. The server manages a hierarchy of storage pools in which the files are saved as it's specified in their management class. If the upper watermark for a storage level is attained, i.e. if this storage level is almost saturated, then the saved files are migrated to another level in order to free space for new data.

The server has several databases that contain the following information:

- the user registration
- the administrator registration
- the location of the client's files in the server's storage pools
- the management class definitions
- the schedule definition for automatic backup and archive operations

These information are particularly important for the correct functioning of the system and every modification is put into a recovery log before it's done in the base. This is done to assure consistency whenever there is a transmission problem.

To bring even more security, databases and recovery logs can be mirrored up to three times to assure against problems that could destroy the original bases and the logs.

2.2.2 Management class structure

ADSM's storage management classes specify when files are eligible for backup or archival, how many backup versions to keep, how long to retain backup versions or archive copies and where to put the backup versions and archive copies in data storage.

The elements of storage policy management are policy domains, policy sets, management classes and copy groups.

Policy domains contain policy sets, management classes and copy groups that can be used by a particular group of users. They are a kind of access rights and specify to which management classes a certain user can bind his files. Each policy domain can contain several policy sets but only one of them can be active at any point of time.

Each policy set contains one default management class and any number of additional management classes. The default class should represent the most common storage management requirements.

A management class can contain an archive copy group, a backup copy group, both of them or no copy group at all. The backup copy group specifies under what conditions and how the backup is done, whereas the archive copy group specifies archival. A file that is bound to a management class without backup copy group is not backed up and if there is no archive copy group, it's not archived.

A particularity of ADSM is that attributes are not only in the copy groups and thus also in the management classes but there are also attributes at a higher level. A policy domain contains backup and archive retention grace period attributes. These

attributes specify the number of days to retain the backup versions and the archive copy when the server is unable to rebind the file to an appropriate management class.

2.2.3 The use of management classes

The definition of the storage management policies falls under the administrator's responsibility. To do this, he can copy and modify the standard storage management policies that come with the product to meet his needs.

When client nodes are registered to the server, the administrator assigns them to a policy domain. If users register themselves they belong to the default policy domain and they can ask the administrator to reassign them to another one.

Include/exclude lists are used to identify what files are eligible for backup services and they specify how the backed up and archived files will be managed. If users do not create an include/exclude list, then all the files will be saved and backup and archival will be done according to the default management class. Included files that are bound to no management class are processed according to the default management class. The include/exclude list can be used to specify the files that should be backed up and to bind the files to a specific management class, so that the file will be managed by the backup and archive copy groups belonging to the management class.

Files remain bound to the same management name, even if the attributes change. If the management class to which a file belongs no longer exists, ADSM uses the default management class attributes to manage the backup versions. If this default management class does not contain a backup copy group or an archive copy group, then backup and archive retention grace periods are used. The grace periods are used to protect backup versions and archive copies from being immediately deleted when the default management class does not contain a backup or archive copy group. The backup retention grace period is also used if a user rebinds a file to a management class that does not contain a backup copy group.

Backup and archival can either be triggered by the users or it can be automated by the central scheduling.

As the administrator with system privileges can grant rights to other persons, he can specify that a certain additional administrator has only management rights on a certain policy domain. This allows to well define for which users the administrators are responsible.

2.3 NetWorker

HSMS and ADSM are two excellent examples of products implementing the management class concept. Not only the implementation is quite the same, they also use the same name for the concept.

Now we're going to see Legato's NetWorker. The product has no management class concept but we try however to see if there cannot be found parallels between NetWorker's architecture and HSMS and ADSM's management class concept.

2.3.1 Brief product description

Legato's NetWorker is a backup solution that was originally written for UNIX but it's also available for Windows NT [Sieme, 94], [Hixon, 94], [Intel, 96]. NetWorker is a good example of a client/server application. The server is responsible for backup mediums, backup devices and management of backup copies whereas the client is responsible for the data on its platform. At the scheduled moment, the server notifies the clients that it is ready for backup. The clients then uses directives, some kind of include/exclude list, to select the files that meet the backup criteria, the files are compressed and sent over to the server.

During recovery, the client requests a list of the backup copies from the server and the user selects the files to recover. The server then transmits the requested copies to the client who decompresses them and writes them to the disk.

NetWorker also implements hierarchical storage management. The administrator can set up migration policies for the client's file systems. When the high watermark is reached, migration starts and continues until all eligible files are migrated or until the low watermark is reached. NetWorker allows pre-migration sweep: candidates for migration are migrated but the original file remains on the disk. Then, when the watermark is reached, and if the migrated files have not changed since migration, the files on the client disk are deleted and migration is complete.

2.3.2 Structure

One of NetWorker's basic concepts are the clients. Clients do represent the file system of their node. A client can appear several times in the server's list, representing each time a different part of the node's file system.

Every client is bound to a schedule which fixes the days for full and incremental backup, the retention period, the number of retained versions and so on. The client also gets its directive for backup. Apart this, there are settings specifying how long backup copies stay in the backup copy index and the access right to the copies.

Clients can be grouped and they can belong to one or several groups. For every group, a start time is set which fixes at what time of the day the backup of that group is started. Every group has also a 'client retries' attribute which fixes how many attempts will be made to backup a client of that group. Another of NetWorker's concepts are the pools. They group the storage devices and they have a number of attributes defining how the stored data will be organized on these devices. Groups, clients, file systems and even the different levels of differential backup can be bound to the pool in which they will be saved. There can be a pool for archive, full backups, incremental backups, off-site storage, confidential information and so on.

2.3.3 Parallels to management classes

NetWorker has no management classes and it has of course a different architecture and uses a different terminology, but there are however some similitudes to products having the management class concept. In HSMS and ADSM, files are assigned to management classes whereas in NetWorker, it's the clients that are

bound to groups. Clients are not only files but they can also be directories or even complete file systems.

Clients are bound to groups that have attributes specifying at which time of the day the backup will be made. This can be used to optimize network traffic.

The clients schedule attribute could also be seen as the name of a management class. Schedules are defined independently from clients or groups and they contain a quite important number of settings specifying the different backup strategies. It's only afterwards that clients are bound to one of these schedules.

The client's directives which specify which kind of files of the assigned file system will be retained for backup are quite similar to ADSM's include/exclude lists.

NetWorker's directives are defined one for all and they are then linked to the clients by the directive attribute. It's the same for browse policy and retention policy. They too are defined by their attributes and then bound to the clients.

In HSMS and ADSM, files were linked to only one management class but in NetWorker, clients can be bound to several management class like groups of attributes. It's interesting to see that attributes are not put all together but grouped according to the nature their function.

If the administrator does not specify a group, schedule, directive or pool for a client, the correspondent attributes will be set to default. In the hierarchy of groups, clients and file systems, each of the levels can be bound to one or several pools. This allows to specify where the backup data will be saved.

We think that NetWorker, even if it doesn't have management classes, has an architecture that has quite a number of similitudes with the management class concept and it might bring some good ideas for extending the concept of the existing products.

In this chapter, the analysis of the three existing products has allowed us to get an idea of what features could be useful for a storage management tool. It has become clear that there is not only one way to implement the management class concept. It also appeared that the management class concept alone is not enough to make a storage management system but that there are a lot of additional concepts that go with it and that need to be discussed. That's why in the next chapter, we're going to develop our own view of the management class concept.

3. Management class concept

In a computer system, there are a large number of different files, directories, workstations, ... These objects are very different one from the other but when looking at them from a backup, migration or archival point of view, they very often have very similar or even identical needs. This means that it is not necessary to define a strategy for each specific object but only one strategy for every group of objects having identical needs. That's the idea on which is based the management class concept presented in this chapter.

3.1 Definition

We will begin to give our personal view of a management class concept. Even if in what follows, some technical terms will be used, that have already be seen with existing products, they do not necessary have the same meaning but they refer, unless specified otherwise, to our personal model.

3.1.1 Management class

In the context of this study, a management class will be seen as a formalism for defining backup, migration and archival strategies. The management class is defined once for all and then, objects are assigned to it.

It's this assignment that links objects to the strategy chosen by the user. Definition and assignment are two completely different actions that can be executed by different people in different places and on a different time. Definition must however be done before assignment.

A **management class** is materialized by the values of it's attributes specifying how the backup, archival and migration is going to happen. This means that the processing of all the objects linked to this management class will be influenced by these values. When doing an action, such as backup, on a management class, this means that it will be done to all the objects belonging to that class. In fact, they are treated together just as if they were one unique object.

3.1.2 Additional concepts

Apart the central concept of management class, there are also some additional aspects that might be useful in setting up a good system.

3.1.2.1 Default management class

When an object that is assigned to a management class is processed, this is done according to the strategy specified in the management class. When there is automated backup, archival and migration, this can only be done for those objects

for which a strategy has been defined, i.e. that have been assigned to a management class. Assignment however has to be done by someone and this would mean that as long as nobody has assigned an object, it would not be eligible for automated backup, archival and migration. Assignment is easily forgotten or people might have no time to do it. It would thus be very useful to have a default management class to which all the objects, that have not yet been explicitly assigned, belong to.

This means that there has to be defined a **default management class** to which objects belong on their creation and to which they stay linked until their explicit assignment to another class. By this, objects can benefit from backup, archival or migration, even if they have not yet been explicitly assigned.

If an administrator chooses to offer the default management class service, the definition of this class depends of course on his own preferences. There can be no universal definition that's ideal for all objects but it seems logical to define the default management class so that it meets most objects needs. This would mean that these objects belong to the right class, right from the start and that there is no need to change their assignment afterwards.

It can also be very useful to specify the default management class so that it assures a minimum service. This is especially useful for backup where it is of greatest importance to have a backup service since the objects creation because it might happen that a problem appears before people had time to assign it to a class of their choice.

On the other hand, there might be objects, such as temporary files for instance, that are not interesting to be processed. It should thus always be possible to specify that a certain type of objects will not be linked to the default management class when they are created.

3.1.2.2 Management class specification

The specification of the management classes is of course one of the most important things to do because everything depends on it. If management classes are well defined, we can have sure backup, fast migration and comfortable archival, but if they are not well defined, this can be as worse as having nothing at all. That's why the specification should not be done by the users. As they have not enough knowledge and not the global view of the system and they would probably end up with a big number of management classes full of contradictions and inconsistencies.

System administrators however have a more complete view of the system. Their professional knowledge and their practice give them the right skillfulness to detect the needs of the different objects and to set up the right management classes. This will result in a realistic number of well-defined classes. By the way, it is better to let the management classes be defined by one person than by several persons working independently one from the other.

The system itself does not guarantee that the management class definitions are good and that there are no inconsistencies, conflicts or omissions. The administrator is free to define what he wants and it's under his full responsibility to define good classes.

3.1.2.3 Assignment

For assignment, there is the same question to know who is going to do it. As there is a default management class, every object belongs to this class right from the start and thus is eligible for processing.

Users know their objects and approximately know the objects needs. They compare these needs with the specification of the available management classes and then they assign the object to the class that best meets the needs. By this simple assignment action, the object gets a complete backup, archival or migration strategy. In place of having to define a complete strategy, with all the risks of errors that are linked to such a task, the user only has to choose the right class. This means that the user has all the power of good strategies without the work.

Assignment can also be done by the administrator. First, he needs the classes for managing the objects on the server that don't belong to any user and thus, stay under his exclusive responsibility. Apart assigning his own objects, he is also allowed to assign the other user's objects. This fits in the philosophy generally admitted, according to which the administrator has superior rights on everything in the system. He has thus the right to change the assignment of any object if, for any reason, this choice doesn't seem good to him.

3.1.2.4 Policy domain

It's clear that when there are management classes, people are going to use them. The administrator might however have created one or several special classes at the intention of some particular user and he doesn't want anybody else to use them. That's why there has to be some mechanism controlling the users' access to the management classes. This kind of authorization mechanism will be called the policy domain.

A **policy domain** is assigned to a user or a group of users and it groups the management classes that these users are allowed to use. In other words, it is a domain of management classes a user can use to fix his backup, archival or migration policy. If the system allows the definition of a default management class, such a class is contained in every policy domain. If the domain contains several classes, we could imagine to let the user choose which one he wants to be his default class.

These policy domains are set up by the administrator and assigned to the users. A user can be assigned several policy domains. This may happen if, in addition to his personal policy domain, he gets a second one that has been assigned to a team he belongs to or if he plays several roles in the organization. In this case, he can use the management classes of all these domains.

We could of course imagine to let him use the one or the other domain depending on the task he is doing or the objects he is working with but such a mechanism would probably be much too difficult to implement and of too little use. In fact, it would be necessary to detect to which of his roles an object belongs and to allow only the use of the corresponding domain. It might be more realistic to let this under the responsibility of the user.

If we say that users can only make assignments to management classes of their policy domain or the policy domain of a group they belong to, this presupposes that the administrator first has to create an organizational structure representing an hierarchy of groups and users. When this structure is represented, a simple hierarchy going from the smallest, the users, to the biggest, the company, passing by workgroups, teams, departments and so on. The structure could be similar to Figure 3-1.

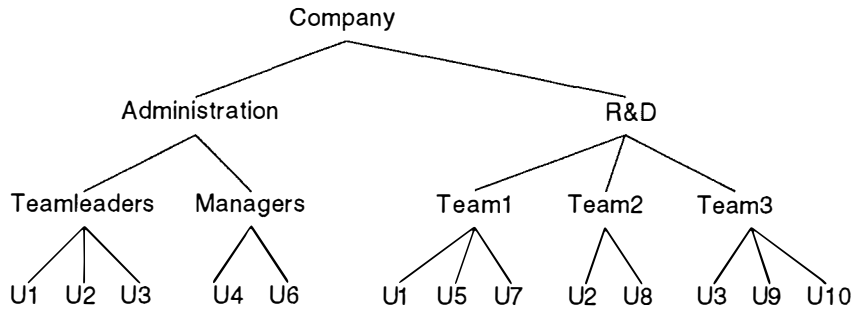

Figure 3-1: Example of an organizational structure

The administrator then defines the different policy domains and assigns them to the different groups and users.

Policy **domain inheritance** is a recursive process. Policy domains of the higher levels propagate through the structure until they arrive at the user's level. Users can dispose of their personal management classes and of all the domains that belong to groups that are above them in the hierarchy.

The main use of inheritance is, that if there is a team where all the users should get the same policy domain, the administrator can simply give the domain to the team and automatically, all team members inherit it. A user can belong to several groups and inherit policy domains from each of them. The main advantage of inheritance is thus a simplification for the administrator. As already said, it can happen that a certain user plays a special role in the organization, either within a group or not and that's why he can get, in addition to the inherited policy domains, personal policy domains that are assigned directly to him. All these policy domains, the inherited and the personal ones, form the user's policy domain.

3.1.2.5 Management class management

A computer system is continually changing and this means that the management classes must be adjusted to the new situation. The administrator is the only one to have the right to modify the classes. To do this, he can edit an existing management class definition and change the values of the different attributes. The strategy formalized by the management class definition changes but not the class itself. There is no change in the assignments and this means that changing the attributes does not affect the objects that are linked to the class.

If the administrator thinks that the modification he made is not susceptible to interest the users, he does not even have to tell them about the change. If however there is a major modification that might influence the users' choice in assignment,

the administrator should send them a message and then, each user is free to change or not his assignments.

The administrator has always the right to create a new management class, which has of course not yet any linked objects.

It may also happen that a certain class is of no use anymore and it must be possible to delete it. There is however the problem that the linked objects will have no class anymore. To assure them at least a minimum service, they fall back into the default management class and their owners are informed, so that they can reassign the objects. It is however possible that the default class offers a much less performant service than the original class and that's why the administrator should inform the users some time before the deletion in order to let them enough time to assign their objects to another class.

3.1.2.6 Delegation of the administrator function

Until now, we always assumed that everything linked to the management class definition has to be done by the system administrator. This seems plausible but when thinking about everything else he has to do, it's easily understandable that he might want to delegate this work to someone else.

The system administrator can thus delegate the management class management but he does not loose his rights. He has always the right to withdraw delegations or to make new ones and to act on the management class definitions.

The management rights can be delegated to one or even several persons, where the last possibility is, even if it's technically possible, not to be used extensively because of the danger of inconsistency in the management class definitions. The delegation goes as far that the administrator has even the right to delegate his delegation and withdrawal rights.

Conflicts are not an issue, even if several people can modify the management classes. The class definitions are unique and it's thus always the latest modification that must be considered. Different levels of rights onto the classes, apart of course the difference between the main system administrator and his deputies, are not necessary because these people should be responsible enough not to sabotage one other's work.

3.2 Potential profits

The value of a concept mainly depends on it's potential use. A well-defined concept that is of no practical use, is worthless. That's why we will now put in evidence the potential profits of the management class concept.

3.2.1 Automation

As said above, a management class is a formalism for defining backup, archival and migration strategies. This means that for the strategies, there exists an unambiguous formal definition, the major prerequisite for automated interpretation.

Automation has an important positive effect on accuracy and performance. Computer systems grow every day and there are so much objects that backing up,

archiving or migrating everything by hand has become impossible. Automated execution of these tasks is not only much faster than manual interpretation but it's also done with much more accuracy. If a task is done by a human, there is always a danger that he makes an error or forgets about something. This can not happen if the task is automated. If it's correctly formalized, it will also be correctly executed.

Automation can also have a positive effect on network traffic. When people trigger backup, archival or migration, they do it without thinking about the traffic they are creating on the net. Backup for example is mostly done at the end of the day and if everybody is doing it at the same time, this has disastrous effects on network traffic. The use of management classes to automate such tasks can thus be of big use for solving this network traffic problem. The module that is interpreting the management class strategy can measure the traffic on the net and distribute its tasks in time in order to optimize the traffic.

3.2.2 Advantages of a good default management class

As said above, a good default management class should be defined to meet the needs of as much as possible objects so that the owners of these objects don't have to reassign them. This means that, if there is no manual assignment and if processing is automated, users don't have to matter about these objects at all. In fact, they get the service for free and there is a considerable economy of work.

For backup, the default class is an important security improvement. The objects belong to a management class right from the start and that's why they are backed up even if their owner has not yet had the time to assign them explicitly.

3.2.3 Dependency management

It happens sometimes that different objects of a computer system are interdependent. An example of such objects are the table files of a database. If one of these objects becomes corrupt we could of course recover only that object but this would lead to inconsistencies because the recovered object would be as it was at the moment of backup whereas the other objects would represent the situation of a later moment. To avoid such problems, the management class concept could be of greatest use. We could imagine assigning the interdependent objects to one specific management class and this would allow to backup and recover them as if they were one object. We might also consider prohibiting to backup or recover them individually.

3.2.4 Economy of work

The use of management classes saves a lot of work because they do only have to be defined one time and are then assigned to several objects. Without this concept, the users would have to define a strategy for each object and this would of course make much more work.

It's the same if a strategy has to be changed. It's not necessary to make the modification for every object but it's enough to modify the management class definition.

If users want another strategy for one of their objects, they cannot modify the management class attributes but they can only reassign their object to another management class. This makes much less work than having to specify a new strategy.

3.2.5 Work-sharing

The separation of management class specification and assignment allows people to do what they know the best. As said above, the person who sets up the classes has the required professional knowledge whereas the one who make the assignments know very well their objects. This work-sharing results in a quality improvement of backup, archival and migration for the different objects.

3.2.6 Processing a complete management class

Every object belongs to a management class and it's possible for the administrator to start backup, archival or migration of a particular class. Objects have been assigned to the same management class because they are similar in one way or the other. It's thus possible to process these objects just as if they were one.

3.2.7 Simplification of selective processing

From time to time users and even the administrator can want to back up, archive or migrate the one or other of their objects. Such a selective processing can be considerably simplified. Without the management class concept, users would have to select the objects and specify afterwards by hand how the processing should be done.

If however the objects have been assigned to a management class, no manual specification is necessary because the strategy is already described in the management class. This simplification is not only an economy of work but it's clear that, the simpler backing up is, the more probable it is that users will do it.

3.2.8 Fix boundaries to the users actions

If backup is completely under the user's control, it may happen that he does bad backups, not enough backup, or even no backups at all. But the opposite is also possible. A user might tend to make backups of his objects, even of the unimportant and the redundant ones, very often and very good. He might make backup copies of every object several times a day, and hold them for a couple of years. Everything's possible. This behavior is of course not dangerous from a security point of view, but it takes too much storage space.

The management class concept would allow to build a backup tool where the users can only assign their objects to the management classes of their policy domain and the tool verifies that these backup strategies are respected.

In a computer system where migration has to be done manually by the users, it will probably not give the desired results. Users tend to not putting their objects back to the slower medium, not only because they have no time, but also because everyone tends to considering his objects as more important than the other people's and thinks that he might probably need them soon again. People are much too

subjective. If migration is automated and uses the management class concept, it will bring the desired results because it will be done according to objective rules.

It's the same for archival. Here too the user isn't free to do what he wants but his possibilities are limited to the archival strategies defined in his policy domain.

3.3 Potential objects

Until now we've only talked about objects without saying in what these objects consist but now we'll discuss what objects of the system environment could be interesting to be managed by management classes.

Files : Files are of course the first object people think about when talking about backup, archival and migration. They are in fact the most common objects and exist on every computer system. Everything people do on their machines is saved in files and that's why files are of extraordinary importance for people's conception of their computer environment.

Directories : A file system without directories is flat and files are all at the same level. Directories are used to make hierarchies of files. We'll not discuss here how it works physically because it changes from one operating system to the other. Logically however it's always the same: a directory can contain files and other directories. Directories are of big use for structuring files and very often they group the files belonging together. This means that it's often much simpler to select a directory for backup or archival than to select the files it contains individually.

Named pipes : Pipes are used in UNIX to exchange data between processes. One process writes data in the pipe and another process reads it in a FIFO way without that processes must know each other. Normally, the pipe data is stored in the file system and a process can work with a pipe as if it was an ordinary file. There are two different kinds of pipes: anonymous and named pipes. The first are temporary and the second are permanent. Named pipes belong to a directory and they have an access path. They are thus potential objects for storage management.

Raw partitions : We have now seen a certain number of objects as they exist in a computer environment. But we can also find objects at a lower, more physical level: the different parts of storage space can be seen as objects. Raw partitions are one of these objects. They contain higher level objects such as files or directories but at this low level, these objects disappear. At the raw partition level there are only bits and it doesn't matter what they actually mean. Doing backups at this physical level has the advantage of being very performant.

Configuration files : When installing an operating system or any other applications, system administrators often take days, or even longer, before everything works as it should. Most of this time is spent in trying to find a configuration that works correctly and brings the desired performance. After the software is correctly installed and configured, the administrators must create the logins, the passwords and the rights. If there are a lot of users, this can take a lot of time.

Whenever there has been a problem and the system has to be reinstalled, this is usually a high stress situation where it is very useful to have backups of configuration files.

Tables : Databases applications save their data in tables. Normally, databases are based on the entity-relation model and the entities can exist on the disk under the form of different files, such as in Borland DBase, or they can be grouped in one big file containing all the information of the database, such as in Microsoft Access.

Log files : Sometimes it might be necessary to make hot backups. This means backup without shutting down the database. During hot backup, each file is backed up at a different time and has of course a different timestamp. There must thus be a method of synchronizing all those files to the same point during restore and that's what redo logs are for. In this log are recorded all the changes that are done to the database. At the moment of restoring from an hot backup, the damaged database files are first replaced with their respective copy and they are then synchronized by redoing all the transactions that affected these files, as they are recorded in the redo logs. Without redo logs covering the time of backup, it's impossible to recover. That's why logs are so important and should be treated very carefully.

Records : Traditionally, tables were the basic objects to be manipulated by backup but nowadays, databases have become so big that it's very often not possible to treat them as a whole. It seems thus interesting to consider doing some actions, not on the whole table but on only a part of it i.e. a record. It might be interesting not to have to make the backup of a 60.000 record database but only to back up the 100 records that have effectively changed.

Links : Links are access paths for files. In UNIX, every file has an element in a directory that points to it's inode. It's however possible to create additional links and this means that it can be accessed from different directories.

Network and network sections : Modern computer systems generally consist of a great number of different devices that are linked together. When it's seen in the light of it's physical architecture, a system administrator might want to do certain operations to parts of this network. It might be useful to make the backup of all the machines in a UNIX pool before updating the kernel. Such network sections might be one kind of objects on which to apply management classes.

Workstation : It's clear that different workstations may have different needs. It's thus normal to imagine that workstations might be seen as objects that can be assigned to management classes. It would thus be possible to assign a workstation to a special management class to do a good backup before the hardware is changed.

User : In a computer system there are a lot of users with different tasks, rights and needs. To serve them better, we can consider a user as an object that can be assigned to a management class. This means that all his objects, such as files and directories, will be processed according to the strategy of that management class.

Team, Workgroup, Department : If a single user can have special needs, it's of course also possible that a group of people has the same special needs. If such a group is considered as an object that can be assigned to a management class, this management class will apply to all the objects belonging to the members of this group. When talking about groups of people that are assigned to management classes, it must however be fixed if this will only affect the personal objects of the members, only the objects they have in common or both.

3.4 Potential attributes

We'll now see the potential attributes that could be used to form a management class definition. It's discussed later which one will effectively be used to form the management classes.

3.4.1 Backup attributes

Retention period : This attribute would be used to specify retention time of the backup copy. When the specified number of days have elapsed, the copies are deleted from data storage.

Browse policy : This attribute fixes the time during which the information about the backed up data stays in the on-line index. This on-line index is much faster and much easier to access than the index on the backup medium. The on-line index allows the users and the administrator to easily search for the desired backup copy. But the on-line index may become very large and that's why it might be a good idea not to let all the backup copies in the index, but only those that are the most susceptible to be needed for a possible recovery. It's of course also possible to access backup copies that are not in the on-line index anymore but it's a little bit more complicated because the index of these older savings has first to be reloaded into the on-line index before browsing.

Scheduling : This allows to choose the backup strategy. The schedule specifies at what time interval backup will be done and it also defines of what kind, full or incremental, the savings will be. It should be possible to specify plans such as: incremental every day and full backup every Friday.

Daytime : This attribute allows to fix the moments during the day when the backup is going to happen. It might be possible to use this attribute to distribute backup activities in time, in order to optimize network traffic.

Number of backup copies : To reduce the risk that restoration could become impossible because the only backup copy has become corrupt, it would be of big use to make not only one, but several backup copies. The number of backup copies is fixed by this attribute.

Multiple backup method : To make multiple backup copies, there are different methods that have their advantages and their disadvantages :

- parallel: the backup copies are done in parallel from the original data. This method is very fast.
- grooming: the first copy is made from the original data, the second one is made from the first copy, the third one is done from the second one and so

on. This method is of course not as fast than the parallel one but it allows to control that the copies can be read.

- comparison: after the backup, the copy is compared with the original data. This takes of course some time but it allows to see if the copy can be read and if it is correct.

It's always possible that a storage medium gets corrupt and that's why the copies must not be stored on the same medium.

Versions data exist : When a new backup is done, the old one is either deleted or it stays on the server. This attribute specifies the number of different old backup versions that stay on the server if the original data still exists. When the maximum number of backup versions is exceeded, the oldest version is deleted.

Versions data deleted : If the original data is erased from a client node one must know what to do with the backup copies. This attribute fixes how many backup versions are retained if the original data doesn't exist any more.

Retain extra versions : The most current backup copy is called the active version and all other versions are called inactive versions. The retain extra versions attribute specifies how many days the server retains inactive backup versions when the original file no longer exists on the client's workstation.

Retain only versions : This attribute is used to specify how many days to retain the only backup version of a file when the original data has been deleted from the workstation.

Serialization : It may happen that an object gets modified during backup process and there are several possibilities for what the backup tool should do in this case. These possible reactions are set by the following values:

- static: the object is not backed up
- shared static: the tool tries several times to back up. If during the last attempt, the object is still used, it's not backed up.
- shared static file: If during the last attempt the object is still used, it's not backed up and it's name is put in a file
- shared dynamic: the tool tries several times to back up and if during the last attempt, the object is still used, it's however saved, even if there's a danger of inconsistency.
- dynamic: the object is backed up, even if it's used.

Encoding : It's always possible that there can be the one or the other problem during transmission. It could thus be very useful to use some error correction encoding based on redundant bits, or any other mechanism, to avoid these problems.

Encryption : Sometimes, sensible data has to be backed up and it must be avoided that anybody can read it. A special room for medium storage is not always sure enough because if someone really wants to steal the medium, he gets in.

The use of a powerful encryption algorithm however can make it almost completely impossible for non authorized persons to read the copies.

Compression : Depending on the kind of data to back up, compression could be very useful to save storage space. This could be the case for images that take a lot of place but generally allow good compression rates.

Minimum storage level (Maximum recall time) : There might be data for which people don't want recall to take too much time. The different existing storage mediums such as optical disk or tape have of course different access times and transfer rates and the maximum recall time could thus be specified by the storage medium that can be used for backup.

Medium : This attribute would allow to choose the medium for backup. This depends of course on a lot of elements: does the backup copy have an infinite or a very short lifetime, probability that it will be needed,....

Storage pool : This attribute allows to specify the storage pool where the server stores the backup versions.

Interdependency : This attribute allows to manage dependencies among the objects belonging to a certain management class. When it is set, the assigned objects can only be backed up or recovered together.

Access rights : Normally only the owner of the original data and the administrator have access to the backup copies. This attribute can however be used to grant certain rights on the backup versions to other people.

3.4.2 Archival attributes

Multiple archive method : It's always possible that archived data gets corrupt and in this case, it's of course very useful to have not only one, but several archived copies. These multiple copies can be done according to the same methods that have already been seen for backup and they are: parallel, grooming and comparison with the original data.

Delete original data : When data is archived, it moves to some off-line support. This attribute specifies what's going to happen to the original data. It's possible to remove it immediately after archiving or to let it remain on-line for a certain time.

Retention period : The retention period attribute fixes how long the archived data remains on the storage medium. After this period has elapsed, the data is deleted.

Index : To enable the recall of data that is archived on some off-line storage, there must of course be some index. This index can be put on the storage medium but it can also be held on-line. It's very good to have an index on the archive medium because it is always with the data and so the archived medium can be easily read on another server.

But it's also useful to have an on-line index. This one can be very easily accessed and it allows the administrator and the users to browse and to find the data they look for.

It's of course possible to have both, on-line and off-line index. It would be useful to have a browse policy fixing how long archived data stays in the on-line index. This can of course be different from the retention time.

Medium : The medium attribute allows to choose to what kind of medium the data will be backed up: Tape, CD-ROM, Optical Disk,... An important argument will be of course the retention time. Data with infinite lifetime could be stored on a non rewritable medium such as CD-ROM whereas data with a shorter retention period could be put on a rewritable medium.

Storage pool : This attribute specifies the name of the storage pool where the archived data will be stored.

Encoding : As for backup, there can be some transmission problem during archiving and thus, error correction encoding could be very useful.

Encryption : The fact of archiving data doesn't mean that it is not important anymore. Very often data stays confidential, even if it's several years old. This means that it could be very important to encrypt archived data because it's always possible that the archive medium gets into false hands.

Compression : As already said above, some data takes a lot of place, but has very high compression rates too. In this case, compression would of course make sense to save storage space.

Access rights : This attribute too has already been seen for backup. It allows to specify which persons, apart the owner, have rights on the archived data.

3.4.3 Migration attributes

Migratable : This attribute serves to specify if an object can, or cannot be migrated.

Unused days : This attribute fixes the minimum number of days an object must stay unused before it becomes eligible for background migration.

Background level : This attribute fixes the background level to which an object will be migrated after leaving the on-line level.

Minimum size : This attribute specifies the minimum size an object must have to be eligible for migration to a background level.

Maximum size : This attribute gives the maximum size an object can have to stay eligible for background migration. Some objects are so big that migration would take too much time, especially during the recall to the on-line level.

Minimum extents : This attribute fixes the minimum number of extents a file must have to be eligible for background migration.

Minimum days : This attribute specifies the minimum number of days an object must be on a certain background level before it can be moved to the next one.

Maximum days : This attribute fixes the maximum number of days an object can stay on a certain background level before it must move to the next one. It can also serve to give the lowest level on which an object can be migrated. For this, the lowest level only has to get an infinite value.

Minimum size for background migration : This attribute specifies the minimum size an object must have to be eligible to move from a background level to another.

Maximum size for background migration : This attribute would fix an upper limit for the objects that can be migrated within the different background levels.

Copy mandatory : This attribute is used to tell the migration tool if yes or no, an object has to be backed up before migration and how many copies have to be done. It would be possible that in this case, backup is completely done according to the backup management class.

In this chapter, we've presented our personal view of management classes and a certain number of concepts that go with it. It's however not enough only to define a concept but we must also define how it shall be used. This will be done in the following chapter where we will develop a so-called management class system.

We have also presented potential attributes for management classes but we have not yet selected the ones that will actually be used to form the management classes. Such a selection of attributes for backup management classes will be done in the fifth chapter.

4. Conception of a management class system

In this chapter we'll see what are the objects to be assigned to the management classes, how they will be linked and how the system will finally use the classes to process these objects.

A fundamental aspect of this management class system will be the entity structure concept. First we'll develop the general theory and then we're going to give an example of such a structure. The rest of the chapters discusses several issues due to the relationship between entity structure and management classes.

4.1 Entity structure

4.1.1 Physical and logical objects

In the previous chapter have been listed a number of potential objects that could be managed by management classes. When looking at these objects, we can see that they can be put in two categories.

First of all, there are objects such as files, directories, raw partitions which physically exist on the storage medium. During backup, archival and migration, it's these objects that will be processed.

In the second category, there can be found objects such as users, workgroups, workstations or network section. If one of these objects, a workstation for instance, is backed up, it's of course not the workstation itself that is written to the backup medium, but only its data, the physical objects that belong to it. That's why the objects of the second category will be called logical objects.

The **physical objects** are the most basic ones and they depend on the platform. There are for instance pipes and links in UNIX but they don't exist in DOS.

The **logical object** category is much less defined. Objects of this kind do not depend on the technical implementation of the platform but on the organizational view a company has from its computer system and its human resources. The organizational view of the computer system is the way the hardware is structured. Workstations can be grouped into pools, pools can belong to a network section and network sections can form a computer system. This is only one of an infinite number of possible structures. It's the same for the human resources. They too can be structured in an infinite number of ways in order to form a hierarchy of users, workgroups, teams, departments,...

In an organizational view, hardware and humans can even be mixed. Workstations can belong to a pool which belongs to a team and so on.

A problem with logical objects is, that they do not exist in the file system and management classes can't get assigned objects which don't exist. To be able however to discuss the potential profits of the management class concept, we must suppose that there is some way to represent these logical objects and to make the link to the management classes.

4.1.2 The entity structure

4.1.2.1 Principles

The link between the real world objects and the management class concept will be made via the **entity** concept which is the representation of the real world objects in the management class system.

As said before, physical objects are imposed by the platform and they will be represented by physical entities. We could imagine a different kind of entity for each kind of physical object but for logical objects, it's more difficult because there is an infinity of possibilities. But even if logical objects have all different names, they have the common particularity that they all can 'possess' physical and logical objects and they can belong to other logical objects.

That's why it could be useful not to define a big number of potential logical entities but only one generic logical entity. This generic entity could be configured to represent every possible logical object and used to set up an entity structure that would reflect the management needs.

How these entities and the structure can be implemented will not be discussed here but the question will be tackled in the realization aspects chapter [6.3]. We'll however outline how such a structure could be represented, in order to be able to discuss the use of management classes.

It's important to notice that entity structure and file system are two completely different things. The structure's **physical entities** represent only these elements of the file system that are interesting from a management point of view but there are a lot of elements in the file system that are not represented.

In the entity structure, there are both logical and physical entities. The main difference is that to every physical entity corresponds one element of the file system whereas logical entities are only created for management needs. **Logical entities** can group other entities that are interesting to be managed together. Besides this, they can also refer to a part of the storage space. To give an example [Figure 4-1], it would be possible to create a logical entity `Team` that would be used to manage all files and directories belonging to a certain team. This entity would be configured to refer to all the places in the storage space where the files and directories of the team and its members are located. In our example this are directories home1 and home2. As it could be useful not only to manage the team as a whole but to manage certain team members' directories independently of the other members', it's possible to create entities for these users in the structure, that

will refer to their directories. These user entities will come under the Team entity because they stand for users that make part of the team.

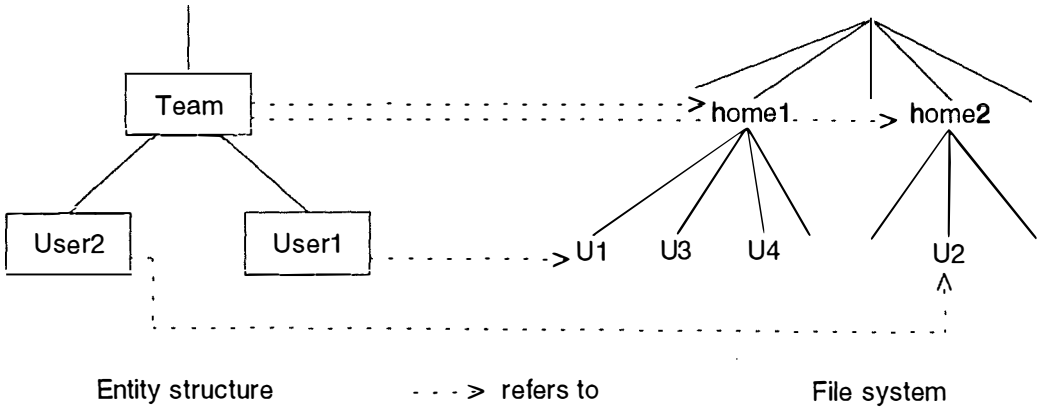

Figure 4-1:Entity structure referring to file system elements

When an action is taken on a physical entity, it's of course its corresponding element in the file system that is processed. For a logical entity, its all the elements in the storage space it refers to, and all the entities that are below it in the structure, that will be processed.

Entities can belong to one or even to several other entities, which are called ancestors. This **belong** link means that the lower level entity belongs to the **ancestor**, from a management point of view: when an action is taken on an entity, it also applies to the owned entities, which are called **descendants**. An entity belonging to several other entities is processed each time when one of his ancestors is processed.

Depending on the platform, there can be certain physical objects, such as directories, that can 'contain' other physical objects. They are close to the logical objects because if an action is taken on them, it also applies to all the objects they contain. The entities representing such objects cannot be treated independently of their descendants.

The other class of physical entities will be called **atomic entities**. They stand for those objects that just like files, cannot contain other objets. Atomic entities are always at the bottom and they can't be ancestors.

4.1.2.2 Formal notation and graphical representation

It's very important not only to consider the direct ancestor of an entity but also the other entities that are above in the structure. If this isn't done, there can be similar problems as in the following example [Figure 4-2].

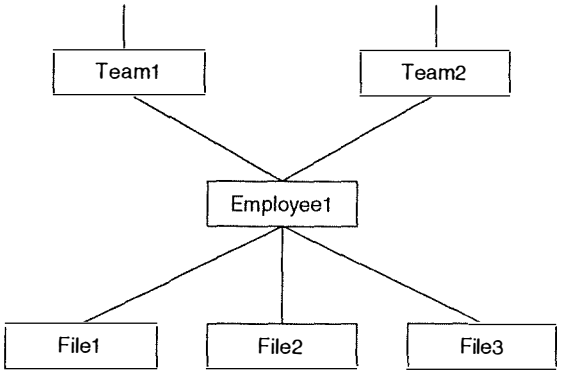

Figure 4-2: Ambiguous entity structure representation

Supposing that there is a logical entity `Employee1` belonging to two different logical entities `Team1` and `Team2` and owning three files `File1`, `File2` and `File3` where the first is the result of some work done for `Team1` whereas `File2` has been created for `Team2`. `File3` is common to both teams. When a backup operation is done on `Team1`, it will propagate on `Employee1` and all the physical entities this one owns, that's to say `File1`, `File2` and `File3`, will be backed up. This is of course not how it should be because from a management point of view, `File2` has nothing to do with `Team1`.

This example should be clear enough to show that it's important not only to consider the direct, but all ancestors of an entity. To do this, we can use a special notation responding to this need.

This **formal notation** has the form of a path: at the left stands the top level and at the right are the low level entities. The different levels are separated by slashes. A level can contain several entities, which are separated by commas. The paths show from which entities the different other entities can be managed. In fact, an entity can be managed by all other entities that are at its left in one of the paths.

If at a certain level, there are several entities separated by commas, this means that for this level, all these entities can manage the physical entities at the right of the path. To allow a better distinction of logical and physical entities, the physical ones will be preceded by an asterisks.

For the example above [Figure 4-2], the formal notation will be:

```
.../Team1/Employee1/*File1,*File3
.../Team2/Employee1/*File2,*File3
```

The notation clearly shows that `*File1` and `*File3` belong to `Team1` via `Employee1` whereas `*File2` and `*File3` belong to `Team2` via `Employee1`.

It's also possible to find a graphical representation of this formal notation. It can be simply deduced from the formal notation using some fundamental rules. The branches are drawn beginning at the top with the high level entity that is a the left of a path. Each time, if there are several entities at a level, there will be a junction for each of these entities. If there are still levels at the right of this junction in the path, these levels will be developed for each of the junctions.

If for a prefix of a path, the representation already exists, it has not to be redrawn but we only have to add the new elements at the right place.

Theoretically, entities having several ancestors would only need to appear once in the representation as long as they have no descendants. It's however possible to find a more general rule. For atomic entities, we can always be sure that they will never have descendants and they can thus have more than one ancestor without any problem.

As said above, non atomic entities cannot be managed independently of their descendants. This is due to the fact that their descendants make not only logically or from a management point of view part of the ancestor but that the objects they represent make also physically part of the objet represented by the ancestor. We will thus fix arbitrary that physical entities are the only one that can have more than one direct ancestor in the graphical representation.

If logical entities appear several times in the representation because they have more than one ancestor, they are different instances of a same entity. These instances can have different ancestors, descendants, management classes and they can refer to different storage spaces but they have the same name. They can be considered as one entity that has been divided into different parts to allow a better management of the different aspects of the object it represents. If one of these instances is selected, we can choose if it's only this instance or all the entities instances with that name that shall be processed.

According to these guidelines, the formal notation of our example will give the following graphical representation [Figure 4-3].

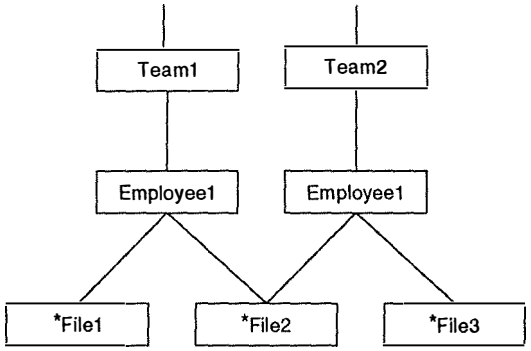

Figure 4-3: Unambiguous entity structure representation

The representation immediately shows that `Team1` manages `*File1` and `*File2` and that `Team2` manages `*File2` and `*File3`. Even if there are two boxes named `Employee1`, they stand in fact for the same logical entity and we can say that `Employee1` manages all three files. This means that if `Employee1` is selected, all the files the correspondent user owns will be processed. If however only one precise `Employee1` instance is selected, there are only two files that are processed.

Here comes another more complete example for the conversion rules [Figure 4-4] :

```
O/D1/T1/U1/*F0,*F1
O/D1/T1,T2/U1/*F2
O/D1/T2/U1/*F3
O/U1/*F4
O/D2/T3/*Dir1/*F5,*F6
O/D2/T3/*F6
```

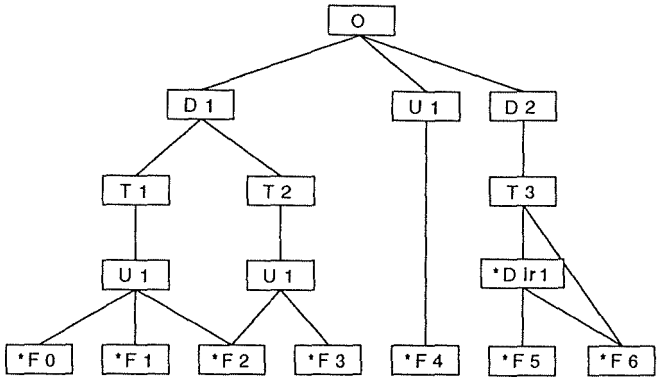

Figure 4-4 : Formal notation and corresponding graphical representation

The graphical representation is of course very useful to explain the concept of the entity structure but in real life, it becomes quickly too large to be of any practical use. Another disadvantage is that it's not possible to represent the information exactly as it's specified in the formal notation without making the representation too complicated. Even if the multiplication of logical entities having several ancestors has been an attempt to eliminate one of these inconsistencies, it must be accepted that the formal notation is much more precise and can contain information that cannot be represented in the graphical representation.

4.1.2.3 References

As said above, if an entity is selected, all descendants and all elements of the referred locations will be processed according to the policies specified by the management classes. In theory, this is quite simple but for one element, there cannot only be several ancestors but also several entities referring at the storage place it's located in. This means that for every element that will be processed, the system has to scan all entities of the structure to find those whose management classes must be used to control the processing of that particular element.

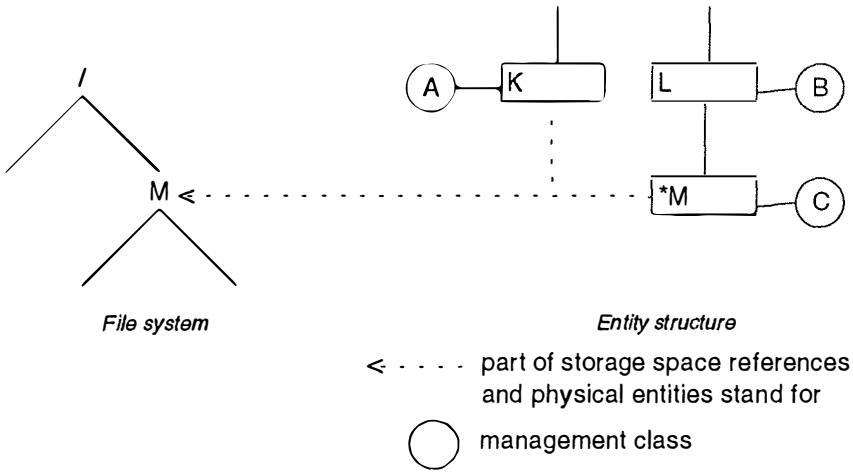

Figure 4-5:Example

In our example [Figure 4-5], this means that if the logical entity L is selected, all elements that are in the directory M will not only be processed according to the management classes 'B' and 'C' but 'A' must also be considered because it's the management class of the logical entity K that also refers to M. To find K, the system has to scan the whole structure.

At the actual state, finding all management classes that must be considered for processing an entity represents an unacceptable amount of work because all the references are independent one from the other and the system must scan all entities to be sure that it finds all references.

A possible solution would be to integrate the references into the entity structure. This would have the advantage that it would be possible to link physical entities and references that stand for, or point to, the same part of the storage space. In this case, as soon as the system has found one **pointer** (reference or physical entity), it only has to follow these links to find all other pointers to this same location. This is of course much faster than scanning all entities.

To add the **references** to the entity structure, we'll first explain how they can be represented in the formal notation. This is not too difficult, the reference of an entity simply has to be added to the notation as if it was a descendant. To differentiate it from the descendants, it will be put into parentheses. In real life, the reference will consist of a formal description of the referred storage space. In this study, we make the simplification only to use an informal description or a symbolic name.

If we remember the example [Figure 4-3] with Employee1, and if we suppose that Employee1 as descendant of Team1 refers to Team1's home directory whereas Employee1 as descendant of Team2 refers to Team2's home directory, the notation would be:

```
.../Team1/Employee1/*File1,*File3,(Team1's home directory)
.../Team2/Employee1/*File2,*File3,(Team2's home directory)
```

In the graphical representation, these references will be represented by boxes with rounded edges. These references are quite similar to physical entities because they also point at a part of the storage space. Just as physical entities, one same reference will only appear once in the structure. If there are several logical entities referring to one same location, there will only be one reference, but it will be linked to each of these logical entities. This principle is also extended to pointers in general and this means that there will never be two pointers standing for exactly the same location.

If a physical entity *M exists [Figure 4-6,a] and a reference pointing to the same location is created, the system does not add a reference to the structure but links the logical entity K with a reference link to the existing physical entity [Figure 4-6,b]. When now the physical entity is removed from the structure the system can see in the formal notation that *M does not belong to the logical entity K but it must be replaced by a reference M [Figure 4-6,c].

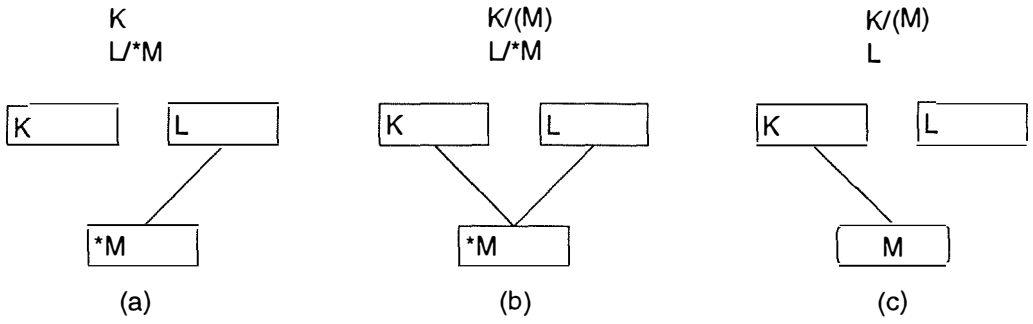

Figure 4-6: Example: Limits of the graphical representation

If there is first a reference M of the logical entity K and a physical entity pointing to the same location and belonging to some entity L is added to the structure, the reference is changed into a physical entity *M that is linked to K and to L

Here appears one of the problems mentioned above, where the graphical representation cannot represent all the information that's contained in the formal notation. When only looking at the structure [Figure 4-6,b], it's impossible to see that *M is in fact not a physical descendant of K but only one of it's references. It's only by looking at the formal notation of the structure that we can get the necessary information to find the structure [Figure 4-6,c] if *M must be removed. We could of course try to solve this problem by using a different kind of links to show that *M is a physical descendant of L and that K only refers to this location. This makes however not much sense because when changing the object structure we can never do without considering the formal notation even if the changes do not concern references but only entities.

An important thing not to forget is that references are not entities. If they are integrated into the entity structure, it's only to show how the system quickly finds back the right management classes that must be used during processing. References can thus not be assigned to management classes, they cannot be selected for processing and they cannot be managed independently from their ancestor(s).

We'll give an example [Figure 4-7] to show how the system works with the references. If a user wants to make a selective backup of the file F1, the system will check if there is a corresponding physical entity. In our example, he won't find any because F1 is not represented in the structure. This is the worst case. The system will then have to scan the references and physical entities to find a pointer to the storage space F1 is located in. If a pointer has been found, the system only has to follow the links to find all other pointers to this location.

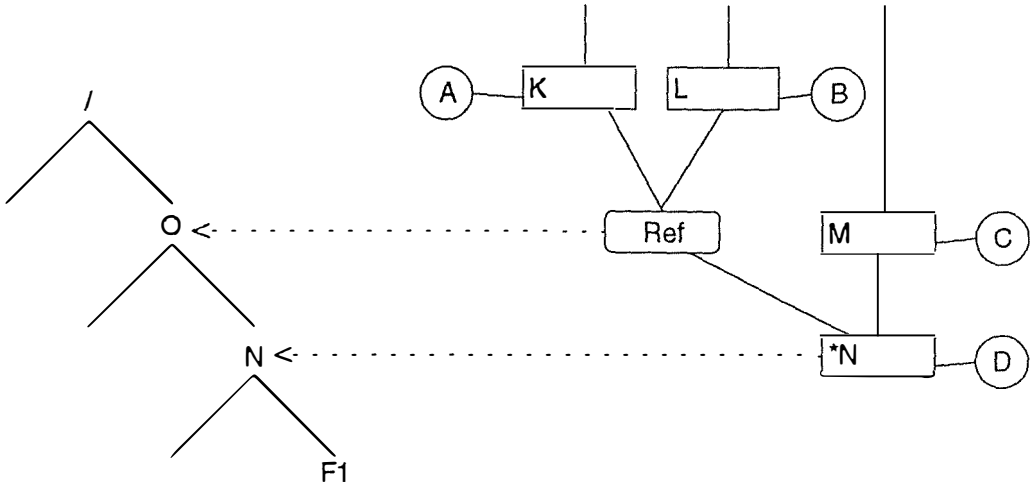

Figure 4-7:References are integrated into the structure

If the system first finds *N, it only has to follow the link to find Ref. As there are no further links to other pointers, the system can be sure that it has found them all. It then only has to consider the pointers' ancestors' management classes. F1 will thus be processed according to the classes 'A','B','C' and 'D'. The procedure is similar if the system first finds Ref.

An aspect that needs to be discussed is the way references are integrated into the entity structure. This has already been anticipated by the link between Ref and *N in the former example [Figure 4-7]. In fact, there will be some kind of hierarchical organization. This is due to the observation that there is also such a hierarchical structure in the referred storage space. If the **structuring of the references** reflects the structure of the real locations they refer at, this has the big advantage that it allows the system to easily find back the entity representing a particular object. In fact, as soon as a pointer to a selected element has been found, the system only has to follow the links above and below it in the structure to find all other pointers to the selected element's location. For processing the element, all management classes of these physical and logical objects must then be considered.

The hierarchical organization that structures references and physical entities must not be build by the administrator but it's done by the system. For this, the system considers the locations that are pointed at by the references and physical entities and then it structures these pointers in a similar way than the locations they point at [Figure 4-8]. This means that in the structure, a pointer will come under another pointer if the location pointed by the second is the next bigger pointed location of which the location of the first pointer makes part of. These links are not represented in the formal notation but they must be calculated by the system.

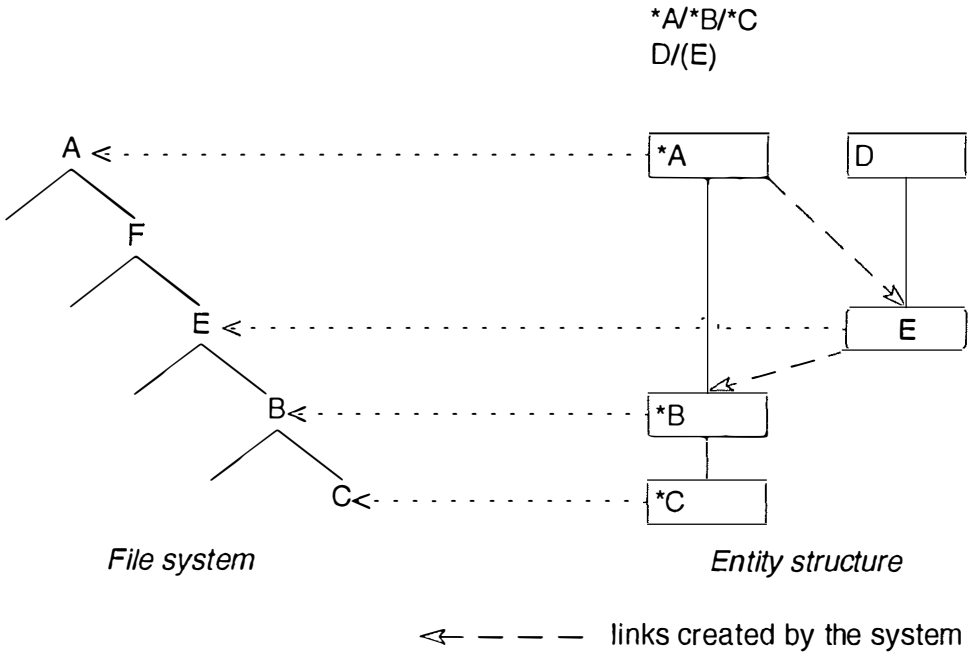

Figure 4-8:Pointers are structured similar to the file system

4.1.2.4 Filters

As everybody sees, it has been very useful to integrate the references into the entity structure. We have until now always considered that references point to a certain location in the storage space. There are however some rare cases where this criteria is not precise enough. There are for example situations where we only want to consider the files that have a particular extension or a particular file attribute. There are furthermore situations where the ideal conditions for the entity structure, i.e. a hierarchical storage organization are not given. An operating system having such a flat file system is Siemens' BS2000. That's why we should think about extending the references by adding some kind of filters that allow only to refer at certain elements of a location.

We can thus say that a reference consists of two parts. First there is the location to which it refers. This is in some sort the name of the reference. Then there are the **filters**, which can be seen as a group of attributes that allow to specify what properties the referred elements must have. They can concern such things like name, extension, owner, size and so on. Together, name and filters form a reference. The rule that two identical references appear only once in the representation still applies. If several references point however to the same location but have different filters, each of them appears in the representation. When a logical entity is processed, the system must not only consider the entity's personal reference but also all other references pointing to the same location but differing only on the filters. To find these other references, we could imagine some kind of link between all references with the same location. In this study, this will be represented in the same way as for the different instances of one same entity, that's to say by the name of the references that will be the same. Two references that only differ by their filters are seen as two different instances of one same reference. For the links the system creates to add the references in the entity structure, the filters are not considered but each instance is linked top the pointers of the level below

and above. In the formal notation, we use a simplified way to represent filters. They will be added in the parentheses that already contain the **name of the reference** (the pointed location) and they will be separated by commas.

For a hierarchical file system, **location** does not only mean the pointed directory but also all other directories below. This means that the filters do not only select the elements of the pointed directory having the required properties but also the elements of the directories below, even if the directory itself does not have the required properties. This default setting can however be changed to give the filters only a range of one level. In this case, if the referred directory contains a sub-directory that satisfies the filters, this one will be completely processed even if its contents does not satisfy the filters.

When now an entity is selected, all elements in the pointed location and having the required properties will be marked for processing. For each of the marked elements, the system then takes all pointers that point at it and which are not contrary to a possible filter of the first pointer. Processing of the element will then be done according to the management classes of the pointers that have been found for the element and the management classes of these pointers ancestors.

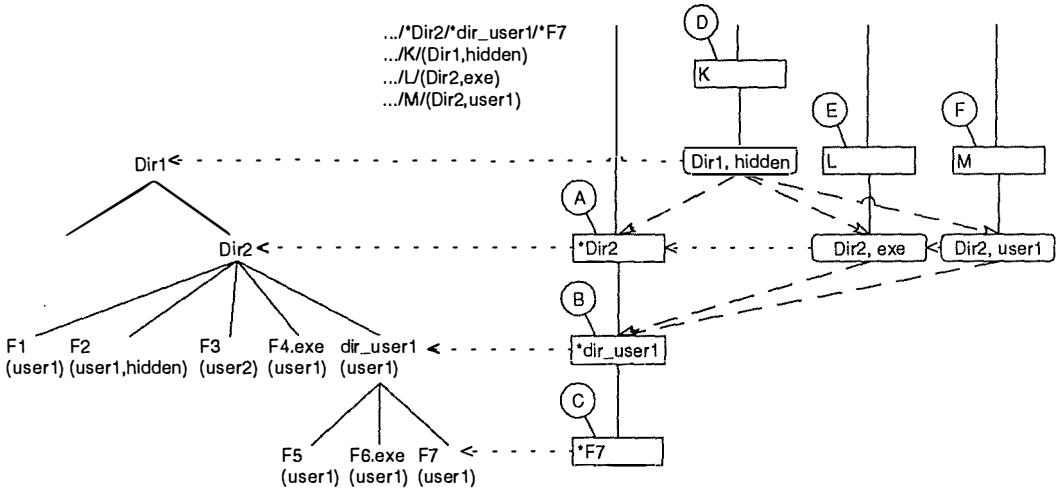

Figure 4-9:Entity structure with filter references

In the example above [Figure 4-9], if the logical entity `M` is selected for backup, its reference `(Dir2,user1)` specifies that all elements of the directory Dir2 where the owner is user1 shall be marked for backup. The marked elements will be : F1, F2, F4.exe and dir_user1 with its files F5, F6.exe and F7.

For each of these marked elements the system now takes all the pointers that point to it and have no contrary filters. The marked elements will then be processed according to the management classes of the pointers that have been found. For the example, we'll give a list [Table 4-1] that contains for each element, the pointers and the management classes that will be used for backup :

element	pointers	management classes
F1	*Dir2 / Dir2,user1	A / F
F2	*Dir2 / Dir2,user1 / Dir1,hidden	A / F / D
F4.exe	*Dir2 / Dir2,user1 / Dir2, exe	A / F / E
dir_user1	*Dir2 / Dir2,user1/ *dir_user1	A / F / B
F5	*Dir2 / Dir2,user1/ *dir_user1	A / F / B
F6.exe	*Dir2 / Dir2,user1/ *dir_user1/ Dir2, exe	A / F / B / E
F7	*Dir2 / Dir2,user1/ *dir_user1/ *F7	A / F / B / C

Table 4-1:Management classes that have to be considered for each element

Now, that we have defined filters for logical entities we could of course see also a need to define them for physical entities. This is however not possible because physical entities stand by definition for one precise element of the file system. A physical entity can stand for a file and a directory but not for a part of a directory. If we only want a part of a non-atomic physical entity to be managed by a class, these entities in which we are interested must either all be assigned to the class or they must be grouped by the filter reference of a logical entity.

It must be accepted that the entity structure concept and more precisely the references and their filters are not perfect and ideal to represent every possible structure. It's also clear that the way the system works with filter references is a little bit heavy but we must not forget that it has only been defined as a kind of add-on to allow the handling of very special cases. A flat file system should thus be the exception and if the storage organization has been done in an intelligent way, it should be possible to do without filter references at all. Even with these inconvenients, the possibilities of the concept are far more powerful than those offered by most of the existing backup products.

4.2 Example of an entity structure

We're now going to give a complete example of how a company's computer system can be represented using an entity structure. The example is based on Siemens-Nixdorf Namur's computer system. The example has however been simplified in that sense that we have replaced the names by more meaningful ones and left aside some minor unimportant details. To build the example, we're first going to present the existing system.

4.2.1 The existing system

4.2.1.1 The SINIX system

The SINIX system consists of a backup server running Legato's NetWorker and seventeen client machines.

The machines are organized in three groups:

- Open session machines: This are the machines that are used by the different research and development groups. Every group has its machine: SINIX_1 for the Spool and Print group (RD1), SINIX_2 for the Presentation services group (RD2) and SINIX_3 for Storage Management (RD3).
- Administration machines: This are the machines that are used for the different administration and support tasks in the organization. To give some examples,

there is the backup server, a machine for the management, machines for mail, invoicing and so on.

- Test machines: This are the machines that are used by the different teams to test their programs. They are assigned dynamically to the teams according to the needs.

We'll see the organization of such a machine at the example of SINIX_3, used by the Storage Management teams.

As on every SINIX machine, there are the system generated file systems for /, /dev, /etc, /usr, /opt, /var and /home. These file systems contain data that changes very rarely. That's why at the moment, backup is only done every three month. For this, the administrator doesn't use any special backup application but only some kind of copy function.

On the other side there are the file systems generated by the administrator. Every team belonging to the group has its own file system. On SINIX_3 these file systems are called /home31, /home32, /home33, /home34 and /home35. On these file systems are located the users' directories in which they save their data.

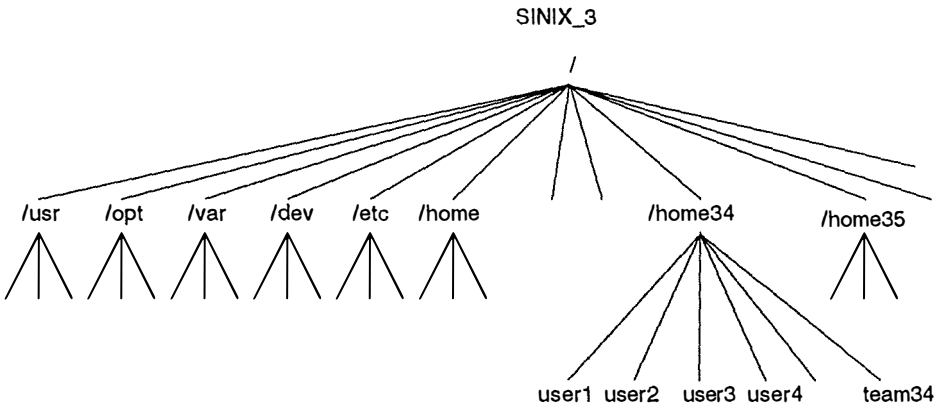

Figure 4-10: File systems of the SINIX_3 machine

The home directories are saved every day, together with the mail directory and the group definition and user login files /etc/group, /etc/passwd and /etc/shadow. The procedure is the same for the other machines.

The server machine is also a client of itself for the NetWorker environment and the index of the saved copies, which are saved every day.

There are two NT stations which are also backed up by NetWorker. It's the backup server for the PC's and the Exchange server responsible for the mail.

4.2.1.2 The personal computers

A certain number of people have their personal computer and these machines are organized into workgroups. There are workgroups representing the different teams, a workgroup for the team leaders, one representing the personal department and so on. These workgroups have however no importance for the storage management and their only benefit is a structuring of the network which allows to quickly find the machine one is looking for in the network environment.

For the SINIX machines, there is centralized backup but for the personal computers there is no backup at all. Users are responsible for backing up their PC themselves. There is no special backup tool to do this and it depends on every user what method he uses. Some of them use a packer before sending their data to the backup server whereas other users simply copy the data to the server.

The backup server is a Windows NT station where every user has been attributed a certain amount of storage space to put backups of his data.

4.2.1.3 The BS2000 mainframe

Users can also work on the BS2000 mainframe. BS2000 has the particularity that it has a flat file system called DMS (Data Management System) and there are no directories to structure the storage space. Data can however be found back due to the User-Id which makes part of the file name.

In backup, data is not processed in a different manner depending on who is its owner but as BS2000 works with hierarchical storage management, there is a difference depending on which storage level data is located. The S0 on-line storage level is saved every day whereas backup of S2 is done every 150 days. S1 is not used.

4.2.1.4 Possible problems

We see that in this system, there are a lot of people who are involved in backup. In SINIX, all system files are saved in the same way (3-monthly) and all user files are saved in the same way (daily). On BS2000, backup differs only between S0 and S2. Backup is thus not too complicated. It's however the personal computers that could cause problem because for them, backup is under the only responsibility of the users. It's always possible that a user forgets to make a backup or does it in the wrong way. It would thus be much better if this task could be centralized and automated. We're now going to discuss how this could be done.

4.2.2 A possible entity structure

When looking at the way a company has organized it's human resources, this structure shows us the different centers of activities and it can thus be very useful of deducing the entity structure from this. We'll now develop a possible entity structure. We do not claim that it's the only or the best solution but the main purpose of the proposition is to illustrate the possibilities of the concept.

4.2.2.1 The organizational view

The top entity of the structure will bee called `SNI`. One of it's descendants will be `Personnel` which groups entities standing for all the main departments of the company such as the Research and Development departments (`RD1`,`RD2`,`RD3`), the Human Resources department (`HR`) and so on.

```
SNI/Personnel/RD1
SNI/Personnel/RD2
SNI/Personnel/RD3
SNI/Personnel/HR
...
```

We can see here at the example of the department RD3 that Research and Development departments own team entities.

```
SNI/Personnel/RD3/Team33
SNI/Personnel/RD3/Team34
SNI/Personnel/RD3/Team35
...
```

Each team entity has descendants that stand for the users of the team. As in a research and development environment, the activities of users, even if they belong to the same team, may be quite diverse, we think that it might be a good idea to create a logical entity for each user. Such a user entity can refer to the user's SINIX home directory, the important directories on his PC and to the BS2000 files having the user's User-ID in their name.

```
SNI/Personnel/RD3/Team34/User1
SNI/Personnel/RD3/Team34/User2
...
```

That's also why none of the user entities' ancestors explicitly refers to a part of the storage space. In fact, all these references are done at the user entity level. For the `User1` entity representing User1, there will be a reference to its files on the mainframe (MF), one to a directory on his PC and one to his SINIX home directory.

```
SNI/Personnel/RD3/Team34/User1/(MF, user1's files),
     (directory on user1's PC),SINIX_3/home34/user1)
```

Besides the user's home directories there are still other directories and files that are important for a team. That's why there is the entity `Mail_Conf` referring to the mail directory, the group definition and user login files of the different SINIX machines associated to the teams.

```
SNI/Personnel/RD3/Mail_Conf/(SINIX_3/var/mail),
     (SINIX_3/etc/group),(SINIX_3/etc/passwd),(SINIX_3/etc/shadow)
...
```

From the formal notation we have until now, it's already possible to deduce a graphical representation [Figure 4-11].

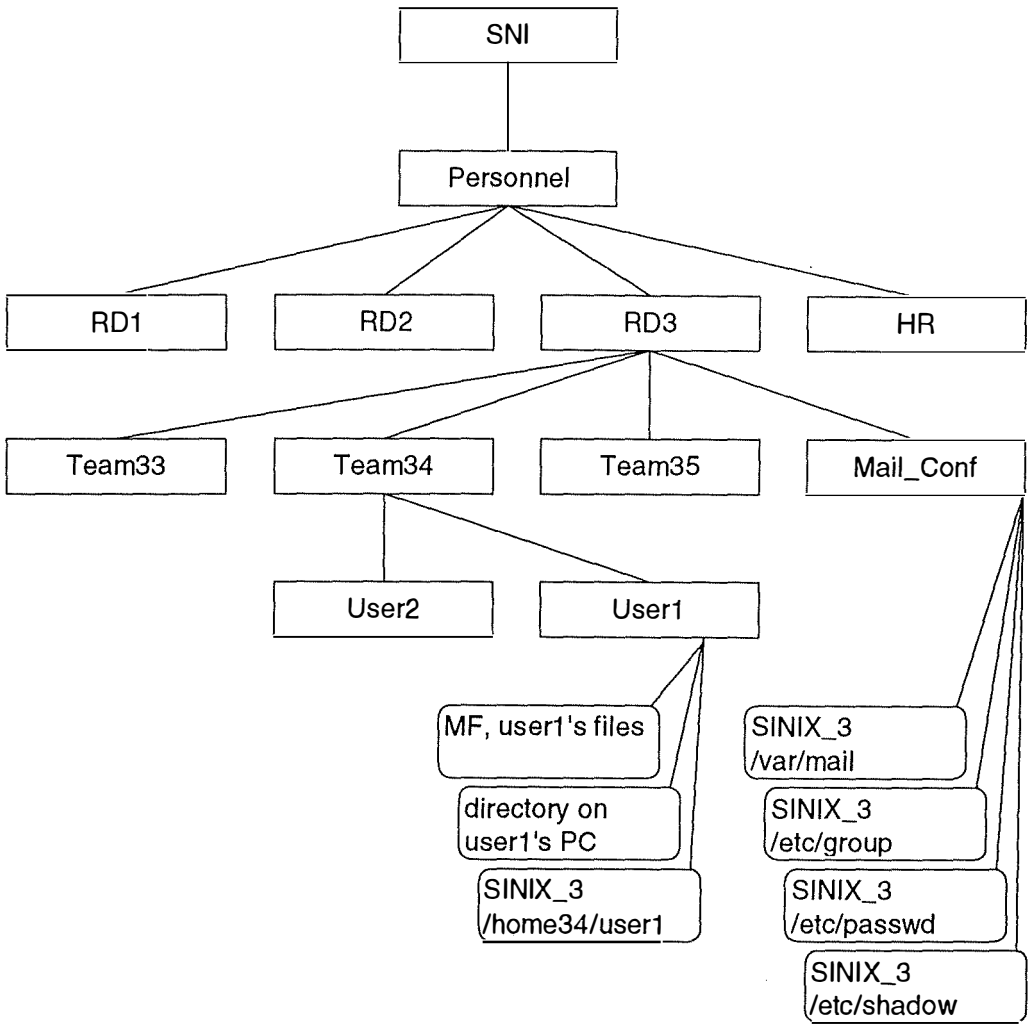

Figure 4-11: Entity structure for the organizational view

4.2.2.2 Mail and configuration files

The part of the entity structure described before bases on the organizational structure of the company and concentrates on the users' directories. We will now see some other aspects of the storage system.

Even if the mail directories are already referred to by the teams' `Mail_Conf` entity, an entity referring to all mail directories can be very useful because it allows to fix a central backup policy for all the mail independently of the users. Assignment to a management class can thus allow to specify one for all that backup of mail will be encrypted and that it will be done at least twice a day.

It's the same for the configuration files. It's very useful to fix in a central way, and one for all, the way all login and other configuration files of the system will be processed. In this example, we only consider SINIX's mail and configuration files.

```
SNI/Mail/(SINIX_3/var/mail)
SNI/Config/(SINIX_3/etc/group),(SINIX_3/etc/passwd),
      (SINIX_3/etc/shadow)
```

We see in the representation [Figure 4-12] that SNI/Mail, SNI/Config and SNI/Personnel/RD3/Mail_Conf use the same reference entities.

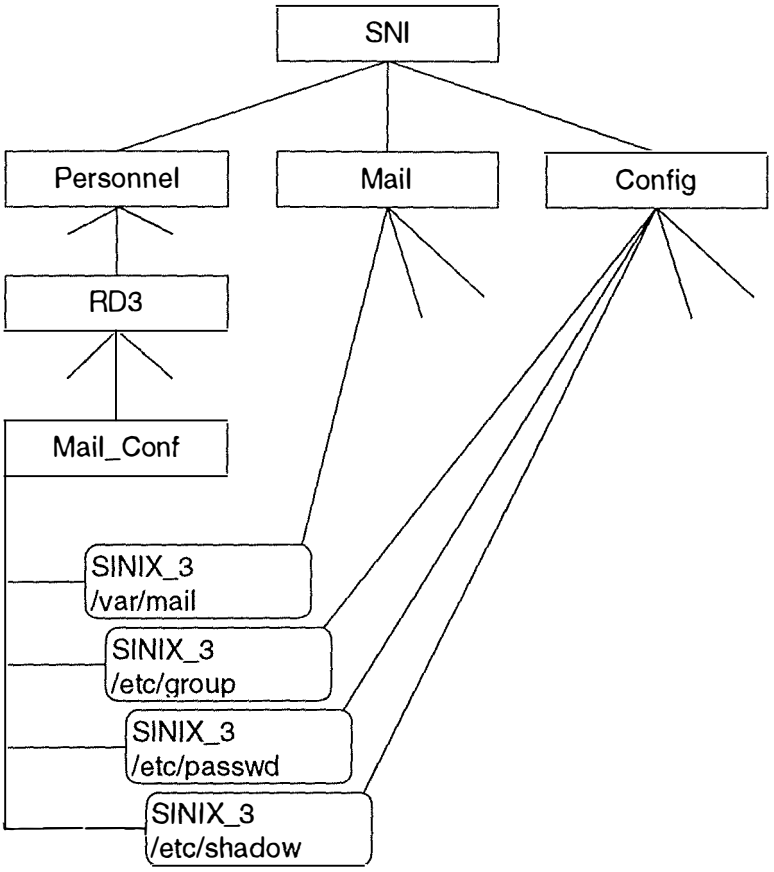

Figure 4-12: Entities for mail and configuration files are added

4.2.2.3 The SINIX view

As backup of the system files is only done every three month, we could do without integrating them into the entity structure. We will however include this system oriented view in the structure to show that the concept allows to manage in one entity structure, different approaches to a same computer system. To do this, we will create a SINIX descendant for SNI.

```
SNI/SINIX
```

The SINIX entity has a descendant System which groups entities (SINIX_2, SINIX_3,...) referring to the system files of the different SINIX machines. If the Systems entity is then assigned to a management class, all system files are processed that way. It's at this level that would be specified that this data is backed up every three month.

```
SNI/SINIX/System
SNI/SINIX/System/SINIX_2/(SINIX_2/var),(SINIX_2/etc), ...
SNI/SINIX/System/SINIX_3/(SINIX_3/var),(SINIX_3/etc), ...
...
```

The other descendant of System is the User_Files entity. As tells its name, it stands for all the data that aren't system, but user generated. This entity would be

assigned to a class specifying that backup should be done at least every day. As we can see at the direct descendants of User_Files, users' data can be managed differently depending on the kind of machine it's located on.

```
SNI/SINIX/User_Files
SNI/SINIX/User_Files/Open_Session
SNI/SINIX/User_Files/Administation,Test
...
```

Instead of developing all branches to the end, we will limit ourself to Open_Session. The other entities could be represented in a similar way. Its descendants SINIX_2, SINIX_3, ... stand for the different machines of that kind.

```
SNI/SINIX/User_Files/Open_Session/SINIX_2
SNI/SINIX/User_Files/Open_Session/SINIX_3
...
```

We see that there are entity names that appear several times. In fact, SINIX_3 appears once as descendant of SNI/SINIX/Systems and once as descendant of SNI/SINIX/User_Files/Open_Session. This are two different instances of the same entity SINIX_3. This means that when SNI/SINIX/System/SINIX_3 is selected for backup, the system files of the machine SINIX_3 are saved whereas SNI/SINIX/User_Files/Open_Session/SINIX_3 saves the user directories that are located on this machine. When SINIX_3 is selected, both aspects are considered and such a backup is equivalent to saving the complete machine.

As said above, every team that uses the machine SINIX_3 has its own home directory. These directories will be added to the object structure and they will be represented by the physical entities */home33, */home34, ... Notice that in this example, whenever no machine name is specified in the pointers, it's SINIX_3 by default.

```
SNI/SINIX/User_Files/Open_Session/SINIX_3/*/home33
SNI/SINIX/User_Files/Open_Session/SINIX_3/*/home34
...
```

It's the same for user directories. They will be represented by physical descendants of the team directory entity.

```
SNI/SINIX/User_Files/Open_Session/SINIX_3/*/home34/*/home34/user1
SNI/SINIX/User_Files/Open_Session/SINIX_3/*/home34/*/home34/user2
...
```

Apart system and user files there are the mail directory and several other important files that are interesting to be included in the structure. This will be done by the following physical entities.

```
SNI/SINIX/User_Files/Open_Session/SINIX_3/*/var/mail,*/etc/group,
     */etc/passwd,*/etc/shadow.
```

There are of course already SNI/Mail, SNI/Config and SNI/Personnel/RD3/Mail_Conf that have references to these elements but from an administration point of view, it might be interesting for the SINIX administrator to manage these elements independently. They are thus added as physical entities to the structure [Figure 4-13].

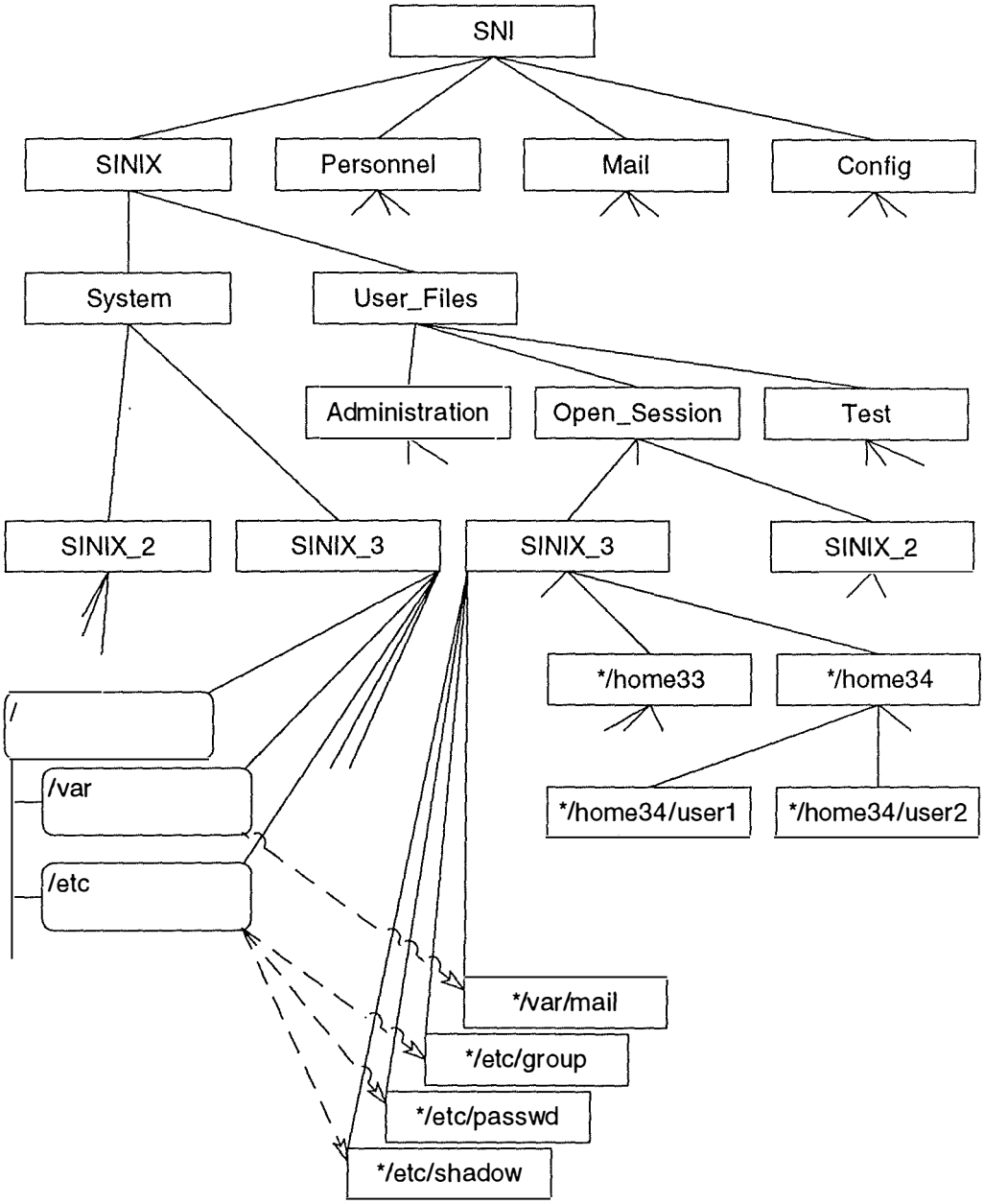

Figure 4-13: The SINIX view in the entity structure

4.2.2.4 The personal computers

When adding the personal computers to the structure, this allows us to assign different workgroups or workstations to management classes that will be used whenever data from these PC's will be backed up. It's thus possible to make a difference between machines used for administration purposes and those used by the teams. The management classes to which these two entities will be assigned can specify a higher security standard for the administration machines. For these entities can be created descendants corresponding to the different workgroups that exist for the moment in the PC system.

```
SNI/PC/Administration,Teams
SNI/PC/Administration/Chef_Groupe
...
SNI/PC/Teams/RD33,RD34
...
```

For these workgroup entities there will be descendants representing the machines of the different users. The machines will be represented by logical entities. These entities will either refer to the different directories of the machine, or if it's interesting to manage certain directories or files by management classes, to create physical entities for them. It's thus possible to specify a policy that will be used every time when a file of a certain PC is saved. In our case, we will only create one physical entity corresponding to the same directory that has already been referred to by the `User1` entity. Notice that in the representation, the reference is now changed into a physical entity. It's only by looking at the formal notation that we can see that for `User1`, 'directory' is only a reference.

```
SNI/PC/Teams/RD34/PC_user1/*PC_user1/(directory)
...
```

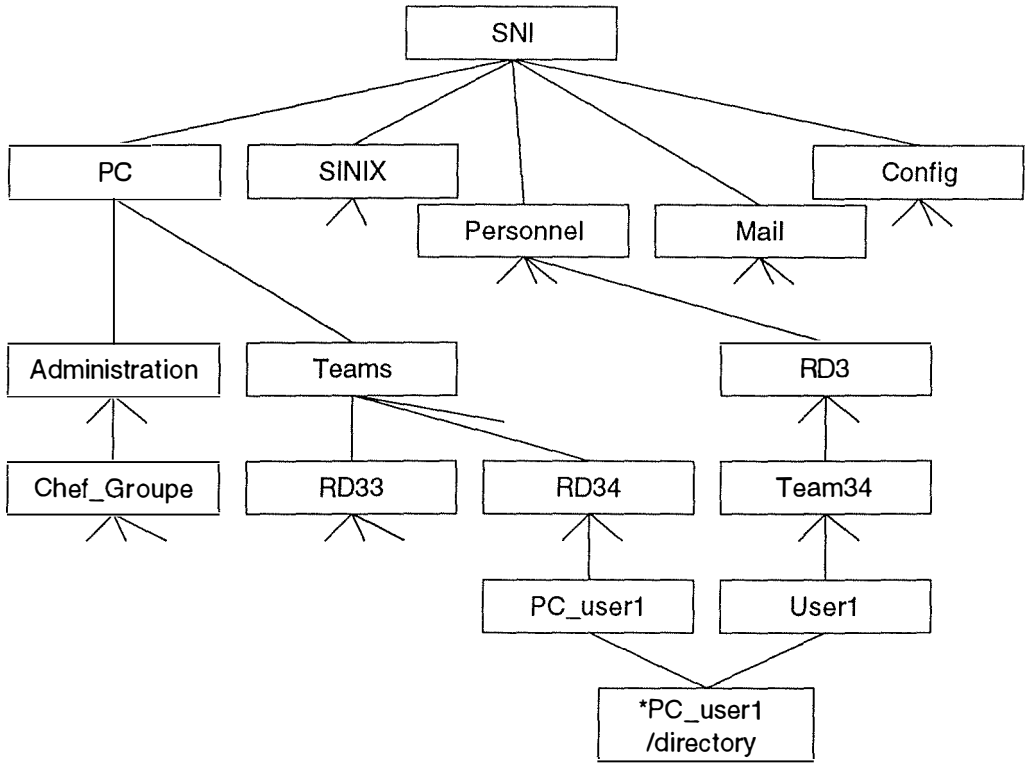

Figure 4-14: PC's are integrated into the entity structure

4.2.2.5 The BS2000 system

As for SINIX machines and personal computers, we can also add a hardware oriented view of the BS2000 system to the entity structure. For this, SNI gets a new descendant BS2000. This entity will get two descendants, S0 and S2 corresponding to the two storage levels that are used. As in HSMS, the catalog

entry of migrated files stays on the S0 level whereas the data is on the S2 level, S2 will refer to the user files that have a migrated attribute.

On the S0 level, there are not only user files, but there are also system files that are never migrated. The S0 entity will thus get two descendants, S0_System and S0_User_Files, corresponding to these two kind of files. The management class to which S0_User_Files is assigned specifies that backup is done each day whereas S2 is backed up every 150 days. This is possible because files are backed up before migration.

```
SNI/BS2000
SNI/BS2000/S0/S0_System/(MF, system files, on-line)
SNI/BS2000/S0/S0_User_Files/(MF, user files, on-line)
SNI/BS2000/S2/(MF, User files, migrated)
```

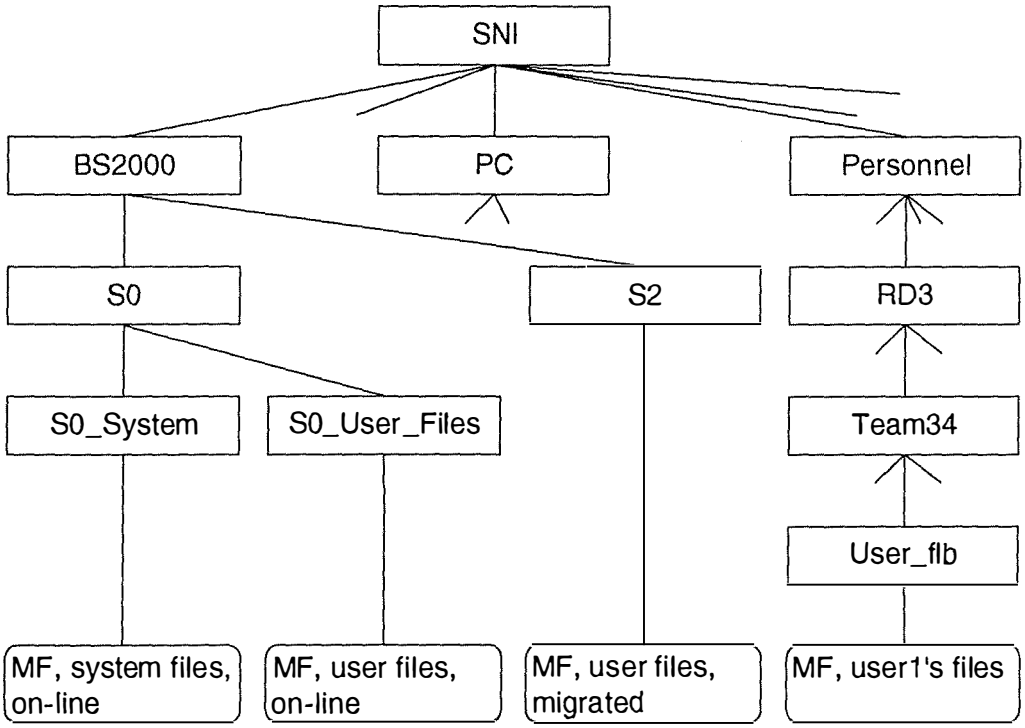

Figure 4-15: The BS2000 view in the entity structure

4.2.2.6 Integration of the different views

The different views we have seen until now, and which represent each another aspect of the computer system, can all be integrated into a same structure [Figure 4-16].

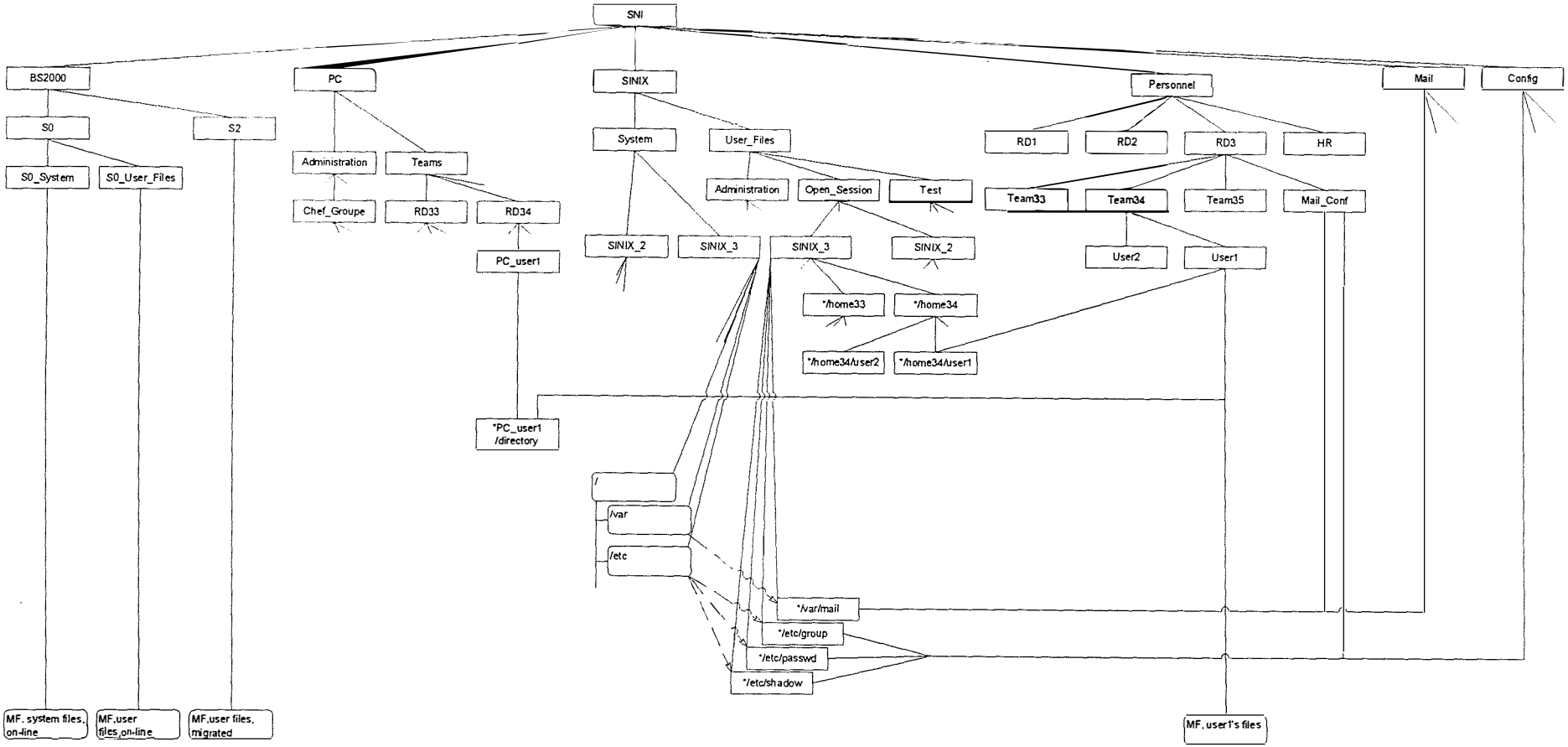

Figure 4-16: Entity structure integrating all previous views

4.3 Management classes in the entity structure

4.3.1 Assignment

We said above [3.1.2.3] that it's normally the owner of an object who does the assignment. It's this assignment to a management class that creates the corresponding entity. For physical objects, some platforms, such as UNIX, offer the owner concept but it's not necessarily the case for every platform. In this case, if we don't choose the restrictive solution which would be only to let the administrator make assignments, we must accept that every user can make or change assignments of an objects. This is of course due to the fact that there is no owner concept.

For logical entities, it's slightly different: some person asks the administrator to create the entity according to his specification. This person can also give the name of a user who will become the **manager** of the entity. We call this user a manager and not owner because he has no absolute rights on the entity but he can only manage it. If not specified otherwise, the manager can only assign the entity to management classes of the special policy domain that comes with the entity. On the entity's creation, the administrator assigns it to one of the classes of the entity's domain.

It's not only after their assignment that the processing of file system elements will be controlled by management classes. Due to the fact that they can be in a location referred to by a physical or logical entity, their processing will be similar to that of these entities. This means that even if the elements don't have their own management class, their processing is already well defined.

When the owner of an element decides to assign it to a management class, he chooses a class in his domain and the system then adds an entity representing the element in the entity structure. The system uses the location of the element and the pointers of the entity structure to find out what entities will become the new entity's ancestors.

A potential problem could be seen in the fact that if a user belongs to two teams and has access to the management classes of both, he could assign a file created for the first to a management class inherited from the second team. A possible solution to this problem would be to create different logins for every group the user belongs to, in order to be able to let the system control what policy domain he's allowed to use. It's however more than probable that users will not like such a system, forcing them to work under several logins.

It's also not possible to build a system, finding the management classes that the user can use, by deducing it from the element's location in the storage space because even if the pointers allow to find the new entity's ancestors, there exists no link between the entity structure and the organizational structure used to determine which classes users can access.

It seems thus that the only solution in such a case is to let the choice of the right policy domain under the user's responsibility because in general, a user who is interested enough to assign his entities should also be capable to choose the right

domain. And even if he chooses the wrong one, this shouldn't be too dangerous because in general very important attributes, such as encryption, will probably already be set at a level above in the management class of one of the new entity's ancestors. This solution depends on how well the policy domains have been defined and assigned to the different users. The more restrictively it's done, the less users can make wrong.

4.3.2 Physical entities belonging to more than one ancestor

The fact that physical entities can have more than one ancestor can be the source of one or the other problem. If an entity shall belong to several other entities, the first problem appears during its creation in the structure, when the system has to detect who are the ancestors. This is due to the fact that the system uses the location in the file system where the object is stored to find out who is the ancestor of an entity. The physical object can however only be stored in one place and so, the system can only detect the ancestor that represents the directory in which the object is saved.

There is no problem if this location is a directory that's common to the new entity's ancestors. In that case, this directory would make part of all ancestors' configuration. Such a common directory is unfortunately not always the case and it's possible that the objects are located in one ancestor's directory and the other ancestors have been granted rights on them.

This is a problem where even sophisticated mechanisms can't help because it's always possible that members of one team create a file that's common to a second team without that the second team does even use the file for months. This means that there is no other solution than binding the entities manually if they have more ancestors than the one in whose directory they are located. This method has of course the big disadvantage that before the manual linking, entities can only be managed by one ancestor and according to this ancestors policy. The choice of the location is in this case very important and the user has the responsibility to choose the directory of the logical entity that assures the best service.

4.3.3 Possible conflicts between management class attributes

4.3.3.1 Conflicts between attributes of different levels

The fact that the manager of a logical entity can assign his entity to a management class means that there will be management classes linked to the entity structure at different levels. So, there is in fact some kind of management class hierarchy.

In such an hierarchy, it's of course always possible that a same management class attribute appears at different levels with different values. This can however be a source of conflict and there must be some rules specifying which values will finally be used to process the physical entities.

We can immediately discard the possible solution where the value that is at the highest or the lowest level in the hierarchy, will be used. The first would mean that if at the top level only one backup copy is specified, it would never be possible to make more than one, even if specified at a lower level. The second possibility

would mean that a user could decide only to make one backup copy even if it's specified at the team level above, that there should be three copies.

The rules should in fact be a little bit more sophisticated and that's why another possibility could be to give the priority to the most restrictive one. This would of course have a positive effect on security but it would have the one or other inconvenient because of its strictness. It could always be possible that one wanted a less restrictive processing than specified above.

In order to assure a certain flexibility, it would be a good solution to fix in the management classes, together with the values, the priority rules for the different attributes. Possible **priorities** would be:

- free: there are no priorities: the attribute value of the lowest level is taken.
- fixed: the attribute's value cannot be changed below.
- restrictive: at a level below, the value can only be replaced by a more restrictive one.

If priorities are set at a certain level, they cannot be changed at some level below. This is of course not the case for the free priority under which any other priority can be put. It's thus possible to specify that in a department there should be done two backup copies for every file (free priority) without prohibiting that a certain team imposes (fixed priority) that for all of it's users files, there must be four copies. Because of this fixed priority at the team level, users have no possibility to make more or less than four backup copies.

'Restrictive' has of course only a meaning if, for the different attributes, the administrator has fixed somewhere in the tool's options which of the possible values are more restrictive than others. If he doesn't do this, the 'restrictive' priority cannot be used.

4.3.3.2 Conflicts between attributes of the same level

We described above [4.3.2] that a physical entity can belong to more than one ancestor. A problem that appears in this case is the question which of its ancestors' attribute values will be used to process the physical entity? We must find a way to solve possible conflicts between attributes of a same level. **'Same level'** is used here to differentiate this kind of conflict from conflicts between management class values of different levels [4.3.3.1].

An idea would be to select for each attribute the value that is the best from a security point of view. This would mean that if one ancestor's data must be encrypted and the other ancestor's data not, their common entities must however be processed in the most secure way, i.e. be encrypted.

Such a behavior where the weight lies on security is of course ideal for backup and archival but it might be possible that for migration, access time is more important than security. The needs can change from one system to the other and it might thus be useful to give the administrator the possibility to fix himself these 'same level' conflict resolution priorities one for all for the whole management class system. When a physical entity is then processed, it's done according to its management class, its ancestors' attributes and the conflict resolution priorities.

For better comprehension, we will now see an example with 'same level' conflicts [Figure 4-17]. When the system processes the entity K, it will use management class 'A' and for M it will use 'C'. The policy for L will be the synthesis of 'A' and 'B'. Possible conflicts between these two classes are solved according to the priorities fixed in these classes. But what's really interesting is the processing of *N. It has its own management class but it's also influenced by the policies used for its direct ancestors. There's on the one side L's policy, computed from 'A' and 'B' and on the other side M's policy which consists of 'C'. These two policies are considered as being of the same level and they will be synthesized using the 'same level' conflict resolution priorities. The policy that results of this computation is finally synthesized with the 'D' management class to give the policy to be used for processing *N.

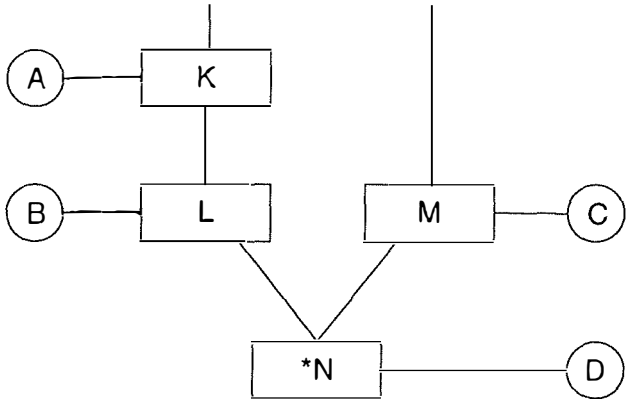

Figure 4-17: Example for 'same level' conflicts

4.3.4 Incomplete management class definitions

The question is, if in a management class definition, every attribute must get a value or if the class can be incomplete.

The advantage of **incomplete management classes** is that they make it possible for a manager only to control one aspect of management, because only certain attributes are set at this level. It may be possible that a manager is only responsible for fixing the number of backup copies that shall be done for his entity but has nothing to do with the other settings. If incomplete management classes were not allowed, the administrator would have to copy the values of the other attributes from the entity's direct ancestor. Besides the inconvenient of additional work, an even bigger problem would appear in case the ancestor's attributes changed. In this case, our user's entity would still have the old values. If attributes have to be redefined at every level, there can be no propagation of changes. Thus the importance of incomplete management classes.

Another question is at which levels of the entity structure incomplete classes would be allowed. The use of incomplete management classes at the top level makes of course no sense because their main benefit, the fact that they allow propagation of attributes, doesn't play. A more important argument against an incomplete management class at the top level is however that in principle, as at the levels below, classes can be incomplete, it could happen that for an entity, a certain

attribute would not yet have been set. To discard this, we stipulate that the management class at the top level must be complete and so the tool can find a value for each attribute of every entity's management class.

At the levels below, incomplete classes can be useful. The attributes that are set will be used and propagated if not specified otherwise by a priority in some level above. For the other, non specified attributes, the values propagated will be the same than those used by the direct ancestor.

4.3.5 Default management classes

We have seen above, that elements that have not yet been assigned can be processed as specified by the management classes of the entities that refer to their location. We have also seen that logical entities that are created by the administrator get assigned to a class of the policy domain that comes with them.

This means that a default management class is not any longer the only way to assure the processing of file system elements, even before their assignment. We decided however not to abandon the concept because it can always be possible that a particular user wants to customize the default processing of his newly created objects.

Default management classes are also concerned by priority rules. As for normal classes, their attributes will have priorities. If a default management class is used at a certain level, it's not absolute. This means that during processing, it's not inevitably the class's attribute values that are used but processing dynamically changes in function of the management classes that are used in the levels above.

For default management classes, incomplete class definitions can be used in the same way than described before.

4.3.6 Modification of the entity structure by the users

One detail that has not yet been discussed are the changes users can make to the entity structure. As said before, if a user decides to assign one of his file system elements, a corresponding entity will be added to the entity structure under the ancestor referring to the storage place in which the user's object is located. If the new entity stands for a non atomic physical object that contains other objects that have already been assigned, the entities of the latter will become descendants of the new entity. Just as for the integration of the references into the entity structure, the hierarchy, as it exists in the file system, must be respected.

We'll give an example of such a special case to show where's the problem [Figure 4-18]. In the example, there is a `User1` entity owning an entity `*File1`.

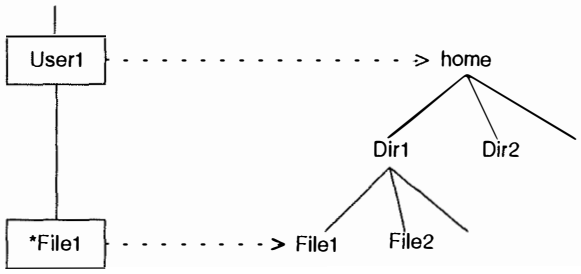

Figure 4-18: Entity structure before assignment of Dir1

As we can see in Figure 4-19, File1 makes part of the directory Dir1. When the user now decides to assign Dir1 to a management class, a corresponding entity must be added to the structure. Even if entity structure and file system are two completely different things, the hierarchy of entities in the file system must also be respected in the entity structure because if `*File1` was not a descendant of `*Dir1`, it would get much more complicated to determine which attribute values shall be used for the processing of `*File1`. It's clear that it's processing must be influenced by the management class to which the directory that contains File1, has been assigned.

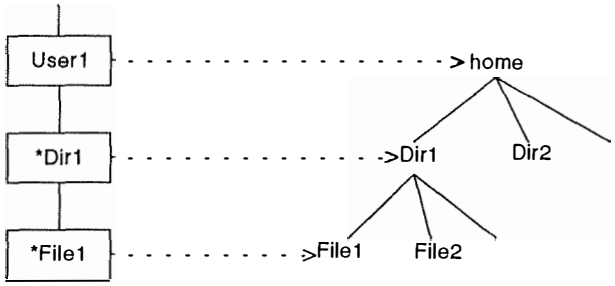

Figure 4-19: Entity structure after assignment of Dir1

In the same example, just imagine `*File1` had two direct ancestors: `User1` and `User2` [Figure 4-20]. If now, Dir1 is assigned to a management class, `*Dir1` must become `*File1`'s direct ancestors. As `*Dir1` stands for a directory, it cannot be created for each of the users, because even if two users have rights on a same file, this doesn't mean yet that they have both rights in the directory it is located in. That's why when `*Dir1` is created, the system should only add it under the ancestor in whose storage space Dir1 is located. In our example, Dir1 is located in User1's home directory and so `*Dir1` becomes a descendant of `User1`.

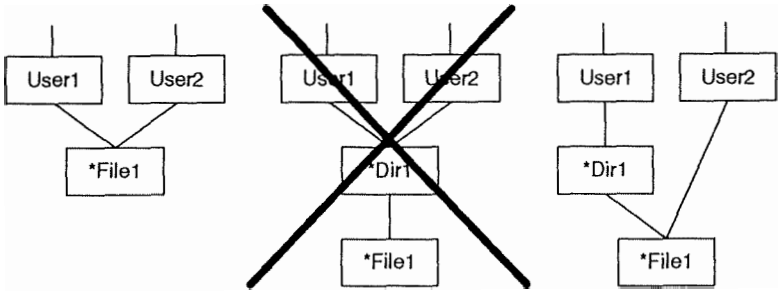

Figure 4-20: How to add a direct ancestor to `*File1`

4.3.7 How management classes are linked to the entities

Another question which we have not yet dealt with, is the way entities, management classes and file system are linked. This depends of course on the way these different elements shall be used.

If one talks about assigning entities to management classes this makes people think that in every management class, there is a register of all entities assigned to it. Such a register is particularly useful in cases one wants to backup, archive or migrate all entities assigned to the management class, because the system only has to look in one place to get a complete list. This may happen quite frequently because scheduled (automated) processing works by management classes.

If an action is done on a logical entity, the file system elements that will be processed are not only those that are in the locations the entity directly refers to, but also all elements referred to by the logical entity's descendants. The tool then uses the method that has been explained with the references to find the entities referring to each of the elements. In this case, all the management classes to which the different entities have been assigned, must be considered and if the only link between entities and management classes is the classes' register, then the system has to scan the registers of all classes to find out which ones apply to which entities. This is of course not the ideal situation and that's why it's a good idea to register in the logical entity, the name of its class which allows to find immediately the right class.

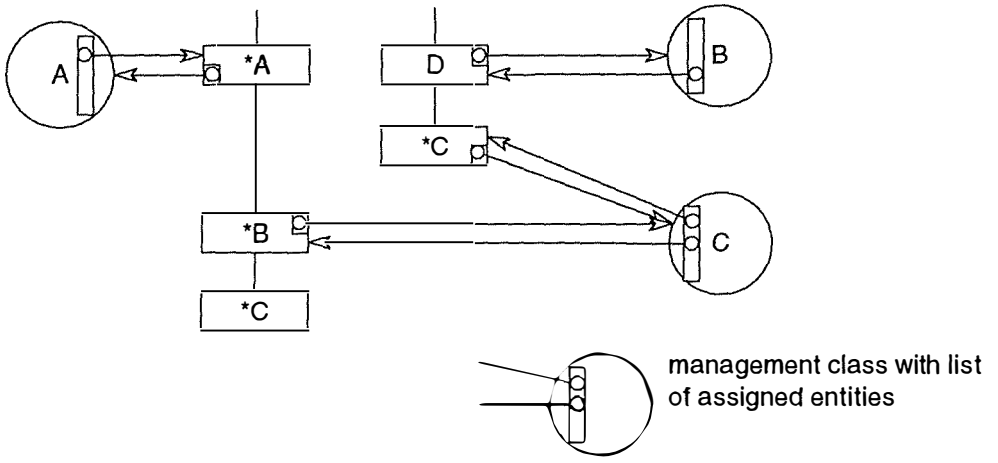

Figure 4-21: How entities and management classes are linked

An inconvenient of the system as it has been explained until now is the amount of work it makes to find all the pointers to a file system element. This can however be solved by computing power and it should not be the main problem because for the moment data transmission, through the network is more susceptible to slow down backup, archival or migration than the analysis of the entity structure.

A big advantage of this management class system however is its complete independence from the file system. It's completely managed by the storage management tool and it leaves the file system intact. As no modification in the file system is necessary, the solution is platform independent.

4.4 Separated management classes for backup, archival and migration

When management classes are used to control backup, archival and migration there can either be one kind of management classes grouping all attributes or a different kind of management classes for each aspect.

It's the second solution we recommend, and this for several reasons. First of all, because separated classes for each aspect will be much less complex, they are much easier to survey and easier to use.

Another advantage is that if someone wants to work with one aspect of the storage management system, he only has to work with the management classes of this particular aspect and he doesn't need to matter about attributes of aspects he isn't interested in for the moment.

Separated classes are also much more flexible. It's for example possible that a company needs a very sophisticated backup but only a very simple migration. In this case, there will be a big number of different backup management classes, but only very little migration classes. If the company is not interested in archival, it wont even create any correspondent classes at all.

Last but not least, it's also a question of the number of management classes that have to be defined. To give an example, if a firm wanted management classes specifying 5 different policies for backup, 4 for migration and 3 for archival, it would have to define 5x4x3=60 different combined classes to have every possible permutation of policies. With separated classes, one would only need 5+4+3=12 different classes. This is quite an important difference. For less than 6 management classes it's of course the opposite but this is quite unimportant because with such a small number of classes, definition work should not yet be too important.

For this chapter, the initial problem was the question how to make the link between the management classes and the objects of the real world. We have introduced the entity concept that's used to represent real world objects and it's these entities that are finally linked to the management classes. Entities can be structured in an hierarchical way and this structure can then be used to deduce the way in which objets will be processed.

Now that the additional concepts and mechanisms needed to use management classes have been defined, we're going to see in what a management class actually consists.

5. Backup management class

In the third chapter, we have seen a large number of potential attributes for backup, archival and migration and we also said above [4.4] that it is advantageous to have different classes for controlling each of these storage management aspects. In this study, only the backup aspect will be completely developed and we're now going to define the management classes attributes for backup.

5.1 Attributes

Quality of backup and usability of a backup tool depend on the way such a backup management class is defined. This means that the more performant the attributes are and the easier it is to control them, the better it is.

5.1.1 Selection attributes

Attributes can be put into different categories and the selection attributes are those which are used to specify when and how the entities of a management class will be selected for backup.

5.1.1.1 Schedule

For backup, it's very important that the latest backup version is not too old in order to assure that when there is a problem, not too much work has been lost. This means that backups must be done regularly and that their frequency will depend on the frequency of changes that are done to the data. To give two examples, system files or applications change very rarely and must thus not be backed up very often. Users' home directories however change almost every day and must be saved very often. It's thus important to have a possibility to specify the frequency of backup.

In modern computer systems, there is each day more data to be saved. Saving large systems costs not only much in storage space but it also takes a lot of time and computing power. Some systems have become so big that a **full backup** (complete saving of all the data) launched in the evening is not yet ready for the next morning. This means that we must find a possibility to reduce the amount of data to be saved.

A solution is not to save all the data but only the one that has changed since the last backup. By this way, the amount of saved data is drastically reduced. This technique, which is called **incremental backup**, requires the first backup to be a full backup. The following schema [Figure 5-1] represents a full backup followed by incremental savings.

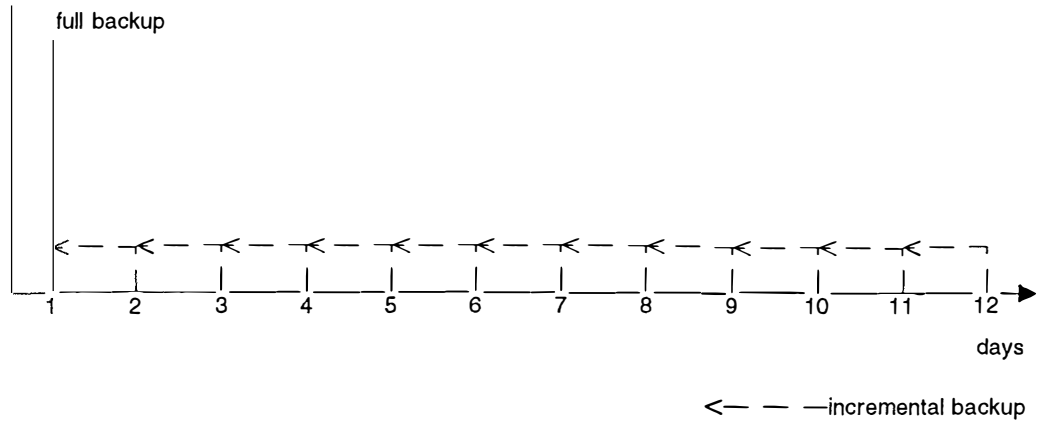

Figure 5-1: Full backup followed by incremental backups

The disadvantage of incremental backup is however that for recovering, all backups since the last full backup are needed. Such a recovering can thus require a quite important number of backup versions and backup mediums. If, for example, an administrator does a full backup on the last Friday of the month with daily incremental backup, and there was a problem just the day before doing the next full backup, he would need more than 20 backup versions to recover. This shows that it's not ideal to do incremental backups all the time.

In order to avoid having to use too much backup versions without however having to do full backups, it's possible to do **differential backups** from time to time which means that all the data that has changed since the last full backup is saved [Figure 5-2]. This means that all the versions that have been done between the last full backup and this differential backup won't be needed anymore.

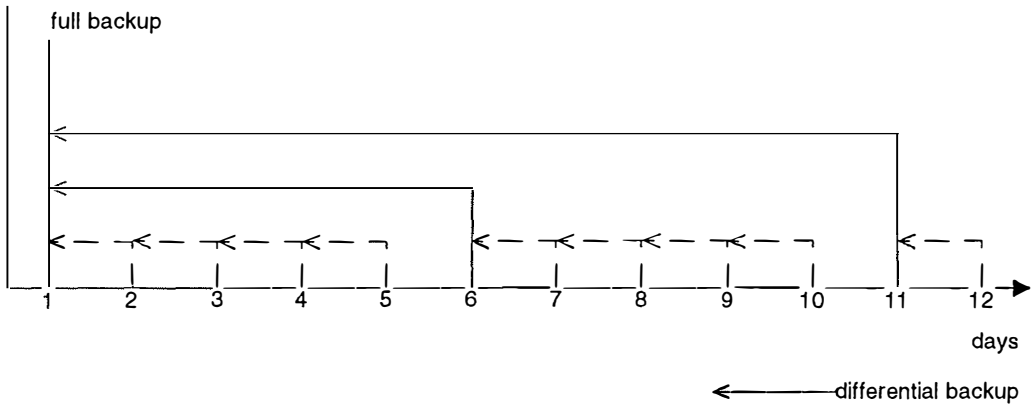

Figure 5-2 : Schema for backup with differential savings

If full backup is done every last Friday of the month, differential backup on all other Fridays and incremental backup on the other days of the week, this means that when there's a problem the day before full backup, one only needs the full backup, the last differential backup and three incremental backups. Recovering with these five backups will be much faster than using the twenty backups that would be needed if one didn't have differential backups.

With the time, differential backups will however become always bigger and sooner or later one will have to do a new full backup. This does not necessarily mean that this new full backup has to be done on-line. In fact, it's technically possible, and it has even been implemented by the HSMS product, to build a new full backup for a certain day simply by generating it off-line from the backup versions.

The plan that fixes on which days which kind of backup will be done, is called the **schedule**. To define the schedule, the tool should offer some kind of calendar on which the administrator could specify for each day the kind of backup to do. The administrator can define the plan for a certain period of time, and the system then calculates it for the whole year. There may be days, such as during holiday, when backup is not necessary. For these days the administrator can define an alternative backup plan in the calendar.

Defining a schedule makes quite a lot of work. In general however, the number of different schedules in a system is not that big and very often, several management classes use the same schedule but differ in other aspects. It would thus be better not to define the schedule within the management class but in some other place and add to the management class only an attribute giving the name of the schedule to be used. This has the big advantage of reusability of the schedules and saves quite a lot of work.

5.1.1.2 Daytime

The maximum frequency a schedule allows to define is a backup each day. There is however data that should be backed up more frequently. There are for example the redo log files of databases that several companies save every two hours or even oftener. For these cases it would be interesting to offer an attribute that would allow to fix the number of minutes to elapse between two backups. It could be of the form: `Backup every: 120 minutes.`

If there are several backups a day, there is however the question of what kind they are going to be. Full, incremental or differential? One thing is sure: they mustn't affect the backup plan as it's defined in the schedule in any way. This means that the number of backup versions needed to recover a particular backup fixed by the schedule mustn't grow because there are several savings a day. These are in fact only additional savings that allow to reduce the amount of lost work of the day the problem happens. It will thus be some kind of **additional incremental backups**. The first one saves the changed data since the last scheduled backup. The following ones save the changes since the last additional incremental backup and so on. Once a day, this additional saving will be replaced by a scheduled saving. The scheduled main backup of that day completely ignores these additional backups and only knows the backups that have been defined in the schedule.

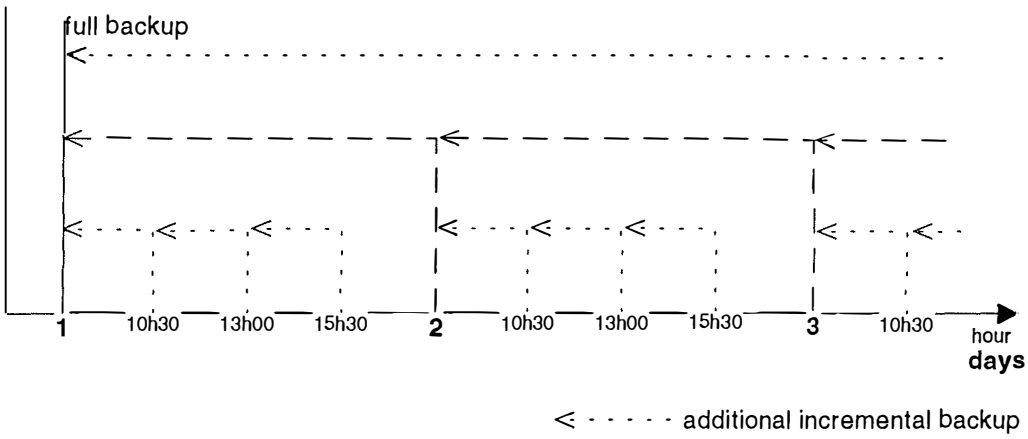

Figure 5-3 : Backup schema with additional incremental savings

A drawback of this technique is of course that the more additional savings there are, the more backup versions are needed to recover a particular situation. It's however still faster to recover all these versions than having to redo the lost work. Also is the amount of data that needs such a high backup frequency, such as log files or some important configuration files, quite limited so that the different additional backups can be on one same medium. The inconvenient of having a big number of versions is thus not that important.

Another aspect that is not fixed by the schedule is the moment of the day when backup is launched. In most companies, this is done in the evening when users have stopped working and the server then works during the night. We propose to add an attribute allowing to fix the moment of backup. If there are several backups a day, the attribute could also be used to fix the launching time for each of them.

An example of such an attribute would be: `Backup at: (12.00 AND 19.00)`

For these backups, the technique of additional incremental backups described before will be used, except for the last saving of the day, that will be done as specified by the schedule and the medium that goes with it.

As said above in the potential benefits of management classes, they allow automation of backup which can have a positive effect on network traffic because the system will be able to distribute backup tasks in order not to bring the net down by doing all at the same time. The precedent 'Backup at:' attribute allows the administrator to distribute backups by hand but there will also be needed an attribute to give the tool the possibility to decide when to launch the tasks. For this, the attribute will give the time interval(s) during which the tool can launch the backup. It will be similar to: `Backup between: ((19.00-21.00 OR 23.00-24.00) AND 3.00-4.00)`. Here too, all savings, except the last one, will be additional incremental savings.

When setting the daytime attribute, the administrator must take care not to put the backup too late in the night, so that it's not yet ready for the next day. With backup, it's always difficult to say how much time it will exactly take and if an interval has been defined for the moment of backup, the tool should start the savings as soon as possible within this interval.

The different attributes described before are closely linked because only one of them can be used at a time. They will thus be the three possibilities of a 'Daytime' attribute. By giving values to one of them, this method will be selected.

```
Daytime:    Backup every .... minutes
            Backup at: ....
            Backup between: ....
```

The 'Daytime' attribute is closely linked to the schedule to which it gives additional precision. This means that the 'Daytime' settings only take effect if according to the schedule, a backup has been planned for that day.

The attribute makes not part of the schedule itself in order to make the system more flexible. It's thus possible to reuse the same schedule even if the backups start at completely different moments of the day.

As the 'Daytime' attribute is always defined in relation to a certain schedule, each management class that has a daytime value must also contain the name of a schedule. In that sense, 'Daytime''s only some kind of sub-attribute of schedule and so, there's a particularity about the propagation of its values : they can only propagate through the levels of the entity structure as long as for these levels, the schedule value stays the same, either by propagation of a higher level value or because the attribute values of these levels are the same than above. Daytime's priorities are only considered in that range. As soon as, at a certain level, the schedule changes, the propagation of the daytime value is stopped and a new value is chosen.

5.1.1.3 User triggered backup

A common attitude concerning backup is that it can only be done by the administrator. Sometimes it might however be useful to allow the user to do the backup of his files and this as often as he wants. A typical situation where such a service could be useful is when a user has to do modifications in a file but doesn't want to loose the precedent version or have to rename the file. If user triggered backup is allowed he can backup his files before modifying them and he has then the possibility to recover the version he wants. This service will be activated by the attribute 'User triggered backup'. The user triggered backups will be done as additional incremental backups. Recovery is only possible during the retention period which depends on the pool on which the data is saved.

A reason for not giving the user the possibility to trigger his backups himself could be that the administrator wants to avoid that the user fills up storage space with his savings or if it's technically not possible because there is no free device.

5.1.2 Backup control attributes

This kind of attributes describes how the backup tool will behave during processing.

5.1.2.1 Serialization

Computer systems have not only grown bigger but they are also used more than ever before. There are a lot companies where the system runs 24 hours a day. This

means that there is less and less time where the backup server can be sure that it has exclusive rights on the data. It becomes thus more and more probable that the data, the tool wants to backup, is already used by another application. Depending on the nature of the data, the backup tool must then adopt a different behavior. For telling the tool what to do in such a situation, we will add the 'Serialization' attribute to the management class.

The most radical behavior for a backup tool in case a file is already used is simply not to back it up. It's however probable that in most cases it will be more intelligent to try at least several times before abandoning. The number of attempts the tool makes to backup a file that's momentary in use will be specified in the attribute '`Static attempts: ....`'

The other possibility if a file is momentary in use is not to abandon, but to save it even if it's in use. For this mechanism there are again several possibilities. The tool can make the backup if the data is used during the first attempt or he can try several times. If the data is still used during the last attempt it will be saved although it's in use. The number of attempts will be fixed by the attribute '`Dynamic attempts: ....`'.

For both methods however there can be situations where backup is not done as it should be. This is the case when data assigned to the class has not been saved because it was used at the moment of backup or has been saved although it was in use. The names of such files will be put in some special report file that informs the administrator about what has happened.

In a computer system there can also be files that are either never closed or for which the time when they are closed is too short to make a backup. There are for example the databases of banks that are extremely big and that can never be closed. To allow however the backup of such files, there have been created tools that allow to make a consistent backup of a file that's in use and permanently changed. The possibility of making such a hot backup can also be specified in the management class.

This will be the third possibility for the 'Serialization' attribute for which only one of them can be selected.

```
Serialization:    Static attempts: ....
                  Dynamic attempts: ....
                  Hot backup
```

5.1.2.2 Dependency

We have already mentioned in the potential benefits of management classes [3.2.3] the possibility to manage dependencies between objects that belong together. We gave the example of files that must always be saved and recovered together in order to assure that there is not one that has a different timestamp, which could lead to inconsistencies.

In order to implement this dependency management feature, the management class will get an attribute called 'Dependency' which tells the tool that the entities assigned to this class can only be backed up together and that they can only be recovered together.

5.1.2.3 Partial backup

The main benefit of the two methods, incremental and differential backup, is that all the data that has not been changed will not be saved. This means however that a large file is completely saved even if only a small part of it has been changed. It would however be more economic only to save the part of the file that has been changed. This method is called partial backup.

The management class will thus contain an attribute 'Partial backup > .... KB' which, if it's activated, allows to fix a minimum size for files for which partial backup will be done in case of incremental and differential backup. An arbitrary value, such as 'no' is used to disable partial backup.

5.1.2.4 Multiple backup method

Having only one backup copy is very dangerous because when this one becomes corrupt, recovery is not possible anymore. Thus the idea of making more than one copy of a backup version. For doing multiple backups, we can chose between the two methods of parallel backup and grooming. Parallel backup means that the tool saves the different backup copies in parallel. Grooming means that the first copy is saved from the original data. All other copies are done from the precedent copy. Due to the sequential saving of the different copies, grooming is not as fast as parallel backup but it has the advantage that it allows to verify that the precedent copies can be read. The time that grooming takes can however be only insignificantly longer because writing of the second copy doesn't require the first copy to be completely written. The writing of the second copy can start as soon as a couple of bits of the first one have been written and on which the system can read.

```
Multiple backup method :      parallel
                              grooming
```

5.1.2.5 Verification

It's very important that backups that have been done can be read. Even if grooming offers some verification, we should extend this service to parallel and single backup. If a backup copy can be read, this does however not yet mean that it's correct. If an error has been made when reading the original data, it will be in each of the backup copies. A more advanced verification functionality that does not only reread but also compare the backup versions with the original data seems thus very useful.

The attribute that will control this functionality will be called: 'Verification' and it will have three possible values : 'None' specifies that no additional verification shall be done. 'Reread' means that the tool verifies that the backups can be read and 'Compare' means that each backup copy will be compared to the original data.

The values set for the 'Verification' attribute will be used independently of the number of backup copies that are done.

```
Verification:     none
                  reread
                  compare
```

5.1.2.6 Encrypted transmission

Sometimes, data is so confidential that it should not be send through the net without being encrypted. There exist transmission protocols which offer encrypted transmission but they don't exist in every environment or certain people could want an additional degree of security. The 'Encrypted transmission' attribute allows thus to specify that the data of that management class must be encrypted before it can be transferred from one node to the other.

5.1.3 Result control attributes

These attributes describe of what form the output of the backup operation will be.

5.1.3.1 Multiple backups and pools

With backups, it's not only important that they are done but they must still be there once they are needed. It' would sometimes be very interesting to retain every backup that is done but this is of course not possible because it costs too much in storage space. The retention period of backups must thus always be a compromise between the security and cost aspects.

In the potential attributes, there were several attributes that seemed very interesting to make part of a management class. These attributes were 'Versions data exist' and 'Versions data deleted' that should be used to fix the number of backup versions to be retained in case the original data still existed on or had been deleted from the platform.

There are however practical limitations to the use of these attributes. In fact, using these attributes would mean that a medium must be retained as long as there are still versions on it for which the retention period has not yet expired, even if all the other data wouldn't be needed anymore. In the worst case there would be some data with infinite retention time and so the medium couldn't be used for new backup anymore. These problems are due to the fact that backup is generally made on mediums such as tape, where it's not possible to replace only the data that isn't needed anymore. We could of course imagine that the system would reorganize the mediums by copying the data from the partially used mediums to a new one and reuse the old one. We consider this however not a reasonable solution because this copying would cost too much in computing time and the devices would be permanently used.

The only acceptable solution seems thus to be to fix the retention period not for particular data but for the medium. This means that after the specified period, the backup medium will be reused and all the data on it will be lost. By defining the retention period for the medium, it will count for all the data of the versions that have been saved on it.

The mediums that can be used to contain the backups of the time interval fixed by the retention period form a set that is called a **pool**. It seems thus that the retention period is more an attribute associated with the pools than with the saved data.

Just to say a word about the backups with infinite retention time. As backup, as we see it, does not offer this service, the only solution is to archive the data.

A pool is also located on a particular device and a device can have several pools. This means for example that a tape drive can have one set of mediums for daily backup and another pool for full backups where the mediums are stored in a fireproof cellar for security reasons. The choice of a device on which a pool shall be located is of course influenced by aspects such as access time or retention period.

Just as schedules, pools will be defined apart and they will only be referred to in the management class.

The retention period is closely linked to the schedule and it has the constraint that it must be long enough to hold all the versions that are necessary for recovering the latest backup. This means that the retention period of a pool must at least be as long as the time that separates two full backups.

We have already mentioned the danger of having only one backup copy. Making several backup copies is however only a solution if the copies are located on different mediums. For security reasons, these copies should also be stored in different locations.

For this, we propose to include into the management class an attribute under the form of a table with two columns [Table 5-1]. The first column will contain the pools used by this class and the second one will give the different kind of backups (full, incremental, differential, additional) that will be saved in the pools.

Pool	Backup
save1	full, incremental, differential
save2	full, incremental, differential
security	full
2-day	additional savings
week	user triggered

Table 5-1:Example of a multiple backup table (draft)

We could imagine one pool 'save1' with a two month retention period for full, incremental and differential backups. For security reasons, a copy of the same data would be stored on a second pool : 'save2'. Full backups could also be stored in a third pool called 'security' and having a retention period of one year, where the mediums would be stored in a fireproof cellar.

As additional savings, specified by the daytime attribute, are only used for files such as log files where it's important always to have a backup that's as up to data as possible but where it's completely uninteresting still to have such a backup after three days because in this time, the changes that had been saved in the additional backup have already been saved by a scheduled main backup, we could imagine saving this data on a pool with a retention period of only 2 days. User triggered savings go to a special 'week' pool.

As the 'Multiple backups' attribute specifies the number of copies for three different kind of concepts, scheduled, additional and user triggered backups, the table layout should be modified in order to consist of three parts and there will also be a column for the priority for each of them. The table will be similar to the following one.

	Pool	Backup	Priority
Scheduled backups	save1	full, incremental, differential	fixed
	save2	full, incremental, differential	
	security	full	
additional backups	2-day	-	fixed
		-	
user triggered back.	week	-	fixed
		-	

Table 5-2:Example of a multiple backup table

When looking at the Table 5-2, we can see that for scheduled and other backups, an infinity of different configurations is possible. It's thus not possible to give a complete list of them which would be necessary for the use of the 'restrictive' priority and this means that for the composed 'Multiple backups' attribute, only the 'fixed' and 'free' priorities can be used.

We see that the 'Multiple backups' attribute is closely linked to the different kind of savings : the scheduled ones, those triggered by the daytime attribute and those triggered by users.

As part of the 'Multiple backups' attribute bases on the schedule, this means that if in a management class a new schedule is set, one also has to adapt the 'Multiple backups' table in such a way that it specifies pools for all these savings. On the other hand, if the 'Multiple backups' table, the settings for scheduled savings are changed, the management class must also contain the name of the schedule according to which the savings will be done. The settings for the scheduled savings in the 'Multiple backups' table depend on the schedule. This also means that these settings will propagate according to the priority fixed in the table for these kind of savings, but only within the propagation range of the schedule which they depend on.

The 'Daytime' attribute fixes another kind of savings, the additional ones, for which there are pools defined in the 'Multiple backups' table. There's a similar dependency between 'Daytime' and the 'Multiple backups' table than between the schedule and the table : if in a management class, 'Daytime' gets a new value, specifying an additional saving, the 'Multiple backups' table must also get a pool for these backups. On the other hand, if the pool for these additional savings is changed, the class must also contain a value for the 'Daytime' attribute.

Due to the **dependencies** that exist between 'Schedule', 'Daytime' and the 'Scheduled backups' and 'Additional backups' sections of the 'Multiple backups' table, none of these attributes can be changed in a management class without giving a value to the other attributes. So, in a management class there are values for each of them or for none of them.

Due to these dependencies which don't allow to give only parts of these attributes a value in one management class, value propagation and priorities must be seen in a different light for these attributes. As for the daytime values which propagate only as long as the schedule doesn't change, we have a similar propagation mechanism for the values of the 'Multiple backups' table : The values of the 'Scheduled backups' section can only propagate as long as the schedule for which they had

been defined also propagates. Their priorities are only used within that range. As soon as at a certain level, there's another schedule value that's used, the propagation of the sections values is stopped.

The propagation of the 'Additional backups' section's values depends on propagation of the daytime value. They have been defined for a certain daytime value and they can propagate according to the defined priorities only as long as the daytime value stays the same.

The third section of the 'Multiple backups' table depends on the user triggered attribute and it gives the pools for these savings. Whenever a management class specifies that users can trigger backups, it must also give a pool for these savings. The priorities of this third section of the 'Multiple backups' table are used to control the propagation of the pool values as long as user triggered savings are allowed.

5.1.3.2 Compression

If the 'Compression' attribute is set, this specifies that the data assigned to this management class will be saved under a compressed form. These can be very useful depending on the kind of data.

5.1.3.3 Encryption

Depending on the confidentiality of the data, it can be assigned to a management class where the 'Encryption' attribute to specifies that it will be saved under encrypted form.

5.1.3.4 Browse time

We have now seen the attributes that control the way the data is backed up. There will however also be attributes in the management class that concern recovery of the data.

To allow the users to recover themselves the data that they need, there is an on-line index containing all the information about the saved entities of the different backups. The user than can select the entity in the index and it will be automatically recovered.

Even if such an index is very useful, it takes a lot of expensive on-line storage space. This is why it must be possible to offer an index not covering the whole retention period but only the last versions that are the most susceptible to be needed by a user. The time during which a version stays in the browsable index will be fixed by a special attribute that will be of the following form : 'Browse time :....days'.

5.1.3.5 Recovery rights

It's generally considered that the people who have the right to recover a file are the owner and the administrator. We will add an attribute to the management class that allows to specify what other people still have the right to recover the assigned entities. This attribute will be called 'Recovery rights: .....'.

5.2 Priorities

Apart the selected value, there will be for each attribute a priority to be set. The priority will either be 'fixed', 'free' or 'restrictive' and it will be used to specify this attribute value's propagation in the entity structure. As said above, 'fixed' means that the value of an attribute cannot be changed below whereas the value of an attribute with the 'free' priority can still be changed. The 'restrictive' priority means that a value can be changed below in the entity structure only by a more restrictive value. In order to use this kind of priority, an order has to be fixed once for all for the possible values of the attributes, from the most restrictive to the less restrictive. This will be done in the priority table. In this table, the most restrictive values are at the left and the less restrictive at the right. For more or less continuous values such as numbers or hours, it should be possible to give arbitrary extreme values separated by a hyphen. To say that small number are more restrictive than large numbers, one would write '0-N' and if moments late in the day are more restrictive than those early in the day, one would write 'late-early'.

For certain attributes, such as 'Daytime' and 'Serialization', a value consists of two parts: the first one selects the method and the second one is used to configure it. The configuration values are put within parentheses (see Table 5-3). For these additional values, it's the same than for the main values: most restrictive at the left, less restrictive at the right.

Attribute	Ordered values (more restrictive ↔ less restrictive)
Schedule	daily, weekly, monthly
Daytime	every(0-N), at (early-late)
User triggered backup	yes, no
Serialization	Dynamic (N-0) , Static (N-0)
Dependency	yes, no
Partial backup	no, yes
Multiple backup method	grooming, parallel
Verification	compare, reread, none
Encrypted transmission	yes, no
Compression	yes, no
Encryption	yes, no
Browse time	0-N
Recovery rights	-, User's team

Table 5-3: Example of a priority table (draft)

The priority table such as we've seen it until now only contains the attribute values ordered by restrictiveness to be used to control propagation of attribute values with the 'restrictive' priority. Another kind of priorities, which will be specified in an additional column of the priority table [Table 5-4], are those that are used to solve conflicts between values of the same level. The content of this column, 'restrictive' or 'less restrictive' refers to the defined order of values. It's clear that there is a big chance that the order that has been defined for values of different levels will also be used for different values of the same level and it's thus not surprising to see almost only 'restrictive' values in the third column.

For this second kind of priorities, there are however some attributes having certain particularities that make it necessary to reexamine the way they will be managed.

For the 'Schedule' attribute, it's difficult to know how to synthesize different values. We could decide to choose only the most restrictive one but the problem

would appear with the pools because the backup copies specified by the other schedules would not be done. On can of course not accept, not to put a copy on a certain pool if it has been specified in some management class because it could have as consequence that recovery is not possible anymore. So we could think about choosing the most restrictive schedule but putting copies to all the pools specified in the considered management classes of the ancestors. This is also not possible because pools are closely linked to the schedule and so, using a pool with another schedule than the one it has been defined for does not give the desired result. If a pool has been defined to contain the incremental backups of a monthly saving, we can of course not use it with another schedule that defines incremental savings for each day of the month.

The only possible solution is thus to consider for a certain entity all schedules of the level just above and making savings according to each of them on the specified pools. This means that for schedules and the 'Scheduled backup' attribute specifying the pools for the savings, no synthesis, but a **merge** will be made and so no 'same level priority' is needed.

It's thus under the administrator's responsibility to define the management classes in such a way that by propagation of the schedules, entities in the structure will not be subject to multiple and useless redundant savings.

The 'Daytime' attribute is closely linked to the schedule and there will be no synthesizing for this attribute either. In fact, as said above, each of the scheduled backups will be done and during these savings, the 'Daytime' attribute will be used. If the 'Daytime' attributes of different management classes specify additional savings, they will all be done and go to the pool specified in these classes. Here too, no 'same level priority' is needed.

The priority table will thus be of the following form.

Attribute	Ordered values (most restr. ↔ less restr.)	Same level
Schedule	daily, weekly, monthly	-
Daytime	every(0-N), at (early-late)	-
User triggered backup	yes, no	restrictive
Serialization	Dynamic (N-0) , Static (N-0)	restrictive
Dependency	yes, no	restrictive
Partial backup	no, yes	restrictive
Multiple backup method	grooming, parallel	restrictive
Verification	compare, reread, none	restrictive
Encrypted transmission	yes, no	restrictive
Compression	yes, no	restrictive
Encryption	yes, no	restrictive
Browse time	0-N	less restr.
Recovery rights	-, User's team	less restr.

Table 5-4: Example of a priority table

If there are different values for 'User triggered backup', the priorities for conflicts between same level management classes fixed in the priority table will be used. This is not a problem but in case the priorities allow the users to trigger backup and each management class has fixed a pool for these savings, it's much more difficult to decide where the data shall effectively be stored.

One solution would be to put them on each of the specified pools. This would however mean that the same data would be saved several times which is not the

ideal solution. If we only want to have one copy of the data, it must of course go to the pool with the longest retention period. This means that the data is not necessarily on the pools on which we originally thought it would be but this should not be a problem because due to the fact that this data has been triggered manually, it's probable that it will also only be recalled selectively. In this case, it's not too important on which pool the copy is located because the performance aspect is not that important. It's enough that the retention time is respected.

A problem with pools for user triggered backups is the storage capacity of these pools. It's a priori impossible to know how much data users will effectively send to this pool. It must thus be possible for the backup administrator to extend the capacity of the pool by adding additional mediums. He can however not add an infinity of mediums to the pool and the capacity is thus practically limited.

Practically, it would be very useful to make it possible to limit the amount of data a certain user can backup himself. A possible idea would be to do it by including into the management classes an attribute for this limit. This will however create a certain number of problems, especially with propagation. If an entity's management class specifies a limit for the amount of data that can be backed up, the limit will concern this entity. But will there be propagation? If no, it's against the principles and if yes, a limit specified for an entity would have no sense because each of its descendants would inherit the same limit.

Even if it's not impossible to find a solution to this problem, we think that it would be better to use some kind of organizational structure [Figure 5-4] similar to the one used for policy domain inheritance. At the top would be fixed the total disposable capacity and for each node would be fixed the amount of storage capacity that could be used by the users represented by this node. This could be used to fix boundaries to users that tend to use up more storage space than they should.

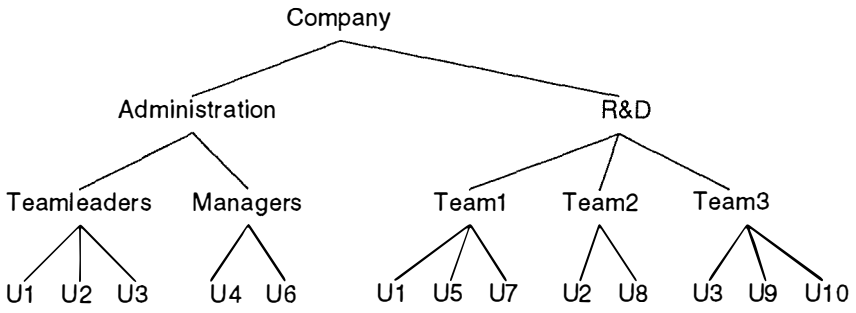

Figure 5-4: Organizational structure

5.3 Using management classes

As said above, the management classes must be performant and easy to use. The large number of attributes defined above to form the management class, have a positive effect on the performance of the system because they allow to specify quite advanced backup policies. It has however also the disadvantage that it's very difficult for the users to choose the right classes for their objects.

Facilitating the use of the management classes has also been an aim of the entity structure. If, for a certain part of the structure, the administrator doesn't want that the users can change particular attributes, he only has to fix them himself. This is quite simple because of the propagation of attributes. In fact, once an attribute is fixed for an entity, it's also fixed for all its descendants. In the management classes of the descendants, this attribute does not need to get a value. This mechanism allows the administrator to reduce drastically the number of attributes in the management classes at the lowest level.

As the more important attributes, such as 'Schedule' or 'Encryption', are generally fixed at a higher level for the logical entities, either by the administrator or some other competent manager, it may happen that at the low levels the number of attributes in the classes between which the users can choose are quite limited. It's even possible to fix all the attributes at a higher level so that the users don't have to bother about assigning their files at all.

Propagation of attributes in the entity structure has the advantage of simplifying the definition of processing policies for the different data elements but it also has an inconvenient whenever reassignment or modification of a management class changes an attribute at a high level in the structure. In fact, the changes propagate through the structure and the processing policies of users' entities can be changed. This is of course susceptible to interest the users.

In general, such changes should be very rare. When however, for some reasons, assignments or management classes must be changed, the person that makes the changes can choose to inform the descendants' owners if he thinks they should know it.

What we said about the management classes sounds very complex and it would be the tool's task to simplify the use of management classes by the services it offers. When the user wants to make or change an assignment, the tools would list him all the entities and file system elements for which he is the owner. When the user selected one of them, the tools would show the policy according to which it is currently processed. To do this, the tool will searche all the referring entities and from the values and priorities of their attributes computes the policy that is used for the selected entity or element.

The user can then make an assignment to one of his management classes. For this, the tool will allow the user to browse through his management classes and only the attributes that are susceptible to change the processing will be shown. Before assigning an entity to a management class, the user can look at a preview showing how the entity would be processed if he did the assignment.

5.4 Example of the use of management classes

Now, that we have defined the attributes forming a backup management class, we're also going to give an example of their use. For this, we take the SINIX view of the first entity structure example [4.2.2.3] and assign the objects of this sub-structure to management classes. Again, we don't claim that the proposed solution is the only possible or the best one but it shall only serve as illustration of how the management class system is supposed to work.

The example will be simplified in the sense that we will limit ourselves strictly to the SINIX view and not consider the rest of the entity structure We've constructed in [4.2.2].

5.4.1 Priorities

A precondition for this system to work are the priorities that have to be fixed. As said before, the middle column of the priority table [Table 5-5] defines an order of restrictiveness whereas the last one specifies which value to choose in case of conflicts between management class values of the same level.

Attribute	Ordered values (most restr. ↔ less restr.)	Same level
Schedule	daily, weekly, monthly, 3-monthly	-
Daytime	every(0-N), at(early-late), between(early-late)	-
User trig. backup	yes, no	restrictive
Serialization	Hot, Dynamic (N-0) , Static (N-0)	restrictive
Dependency	yes, no	restrictive
Partial backup	no, yes	restrictive
Mult.backup method	grooming, parallel	restrictive
Verification	compare, reread, none	restrictive
Encrypted transm.	yes, no	restrictive
Compression	yes, no	restrictive
Encryption	yes, no	restrictive
Browse time	0-N	restrictive
Recovery rights	-, User's team	restrictive

Table 5-5:Priority table

We'll just explain how to read the values' order for the 'Daytime' attribute : the 'backup every...' method is more restrictive than the 'backup at...' method and the 'backup between...' method is the less restrictive. The values between brackets tells us that for each of these methods, small parameters representing a short interval between backups, respectively early hours of the day are considered more restrictive than big numbers or late hours. The rest of the table can be read in a similar way.

5.4.2 Management classes

We will now explain the different management classes that will be assigned to the different entities. As said above, the top entity of a structure, in this case SNI must be assigned to a complete management class. The management class we propose is called 'Top_1' and it has the following attributes :

Attribute	Value	Priority
Schedule	monthly	free
Daytime	backup at : 20h	free
User triggered backup	yes	free
Serialization	static attempts : 3	restrict.
Dependency	no	free
Partial backup	no	free
Multiple backup method	parallel	free
Verification	none	free
Encrypted transmission	no	free
Compression	no	free
Encryption	no	free
Browse time	1 week	free
Recovery rights	-	free

	Pool	Backup	Priority
Scheduled backups	default	full, incremental, differential	fixed
	security	full	
additional backups		-	
		-	
user triggered back.	2-days	-	free
		-	

Table 5-6:Attributes of the 'Top_1' management class

This means that, whenever there's no other class below that specifies otherwise, the descendants of the top entity must be processed according to this management class. The top class specifies that backup will be done according to the 'monthly' schedule. Savings start at 20 o'clock and user triggered backups are allowed. The scheduled savings will be done to the 'default' pool, a second copy of the full backups will be saved on the 'security' pool and this cannot be changed below without changing the schedule. The user triggered backups go to the '2-days' pool. As besides the scheduled savings, no additional backup has been planned, no pool needs to be specified for these savings. The full backups will be written in parallel to the two pools and there will be no verification. If a file is still in use during the third backup attempt, it will not be saved. The saved files will stay in the on-line index for one week. There will be no dependency management, compression, encryption and encrypted transmission. Partial backup is disabled and recovery is only possible by the administrator and the owner or manager of the entity.

As a complete management class for the top entity has been defined, the classes to which the descendants will be assigned don't have to be complete but must only have values for the attributes that are interesting to be managed at this level.

The SINIX entity gets no assignment because it only has two descendants and assignments will be done at the descendants' level.

One of these descendants, the System entity stands for the system files of the different SINIX machines. This logical entity is assigned to the 'SINIX_System_1' management class [Table 5-7] that specifies that for these elements, backup will be done according to the 3-monthly schedule and that the saving must begin before 3 o'clock in the morning. The data will be put to a special pool which will only contain system files and the saved files will not stay in the on-line index. It's not so frequent that system files must be recovered and when it's the case, the administrator can always use the index that's on the backup medium.

Whenever the users triggers backup in addition to the ones defined in the schedules, they will be put to the additional system pool 'system_add'.

Attribute	Value	Priority
Schedule	3-monthly	free
Daytime	backup before : 3h	free
User triggered backup	yes	free
Serialization		
Dependency		
Partial backup		
Multiple backup method		
Verification		
Encrypted transmission		
Compression		
Encryption		
Browse time	none	free
Recovery rights		

	Pool	Backup	Priority
Scheduled backups	system	full	free
additional backups		-	
		-	
user triggered back.	system_add	-	free
		-	

Table 5-7:Attributes of the 'SINIX_System_1' management class

The other descendant of the SINIX entity is the User_Files entity which stands for the users' files on the SINIX machines. In the existing system, these files are saved every day and so, the class to which this entity will be assigned, will use the schedule 'daily' and it will start after 8 o'clock in the evening. All scheduled savings go to two default pools and in addition to this, full backups are stored on a security pool.

It's very useful for users if they can backup their files and directories at any moment they want because this allows them to undo changes they have made to their files by simply recalling the last backup. To make the example more interesting, we decide arbitrary that the 'SINIX_System_Files_1' management class fixes that user triggered savings of the User_Files entity and its descendants must go to the 'week' pool.

Attribute	Value	Priority
Schedule	daily	fixed
Daytime	backup at : 20h	free
User triggered backup	yes	free
Serialization	dynamic attempts :3	free
Dependency	no	free
Partial backup	no	free
Multiple backup method	grooming	free
Verification	none	free
Encrypted transmission	no	free
Compression	no	free
Encryption	no	free
Browse time	1 week	free
Recovery rights	-	free

	Pool	Backup	Priority
Scheduled backups	default_1	full, incremental, differential	free
	default_2	full, incremental, differential	
	security	full	
additional backups		-	
		-	
user triggered back.	week	-	fixed
		-	

Table 5-8:Attributes of the 'SINIX_User_Files_1' management class

The Open_Session entity, a descendant of User_Files, will be assigned to the 'Open_Session_1' management class specifying that recovery cannot only be done by the owner or the administrator but also by the user's team members and that the browse time will be two weeks. In the following table, only those attributes that really get a value are represented. The schedule is 'daily' and savings can be done automatically between 21 o'clock and midnight. Due to the dependency that exists between the schedule, the daytime and the multiple backup table [5.1.3.1], Open_Session's management class must have values in the 'scheduled backups' section of the multiple backup table.

Attribute	Value	Priority
Schedule	daily	fixed
Daytime	backup between : 21h-24h	free
Browse time	2 weeks	free
Recovery rights	user's team	free

	Pool	Backup	Priority
Scheduled backups	default_1	full, incremental, differential	free
	default_2	full, incremental, differential	
	security	full	

Table 5-9:Attributes of the 'Open_Session_1' management class

The following figure represents the entity structure and shows which entities are linked to which management classes.

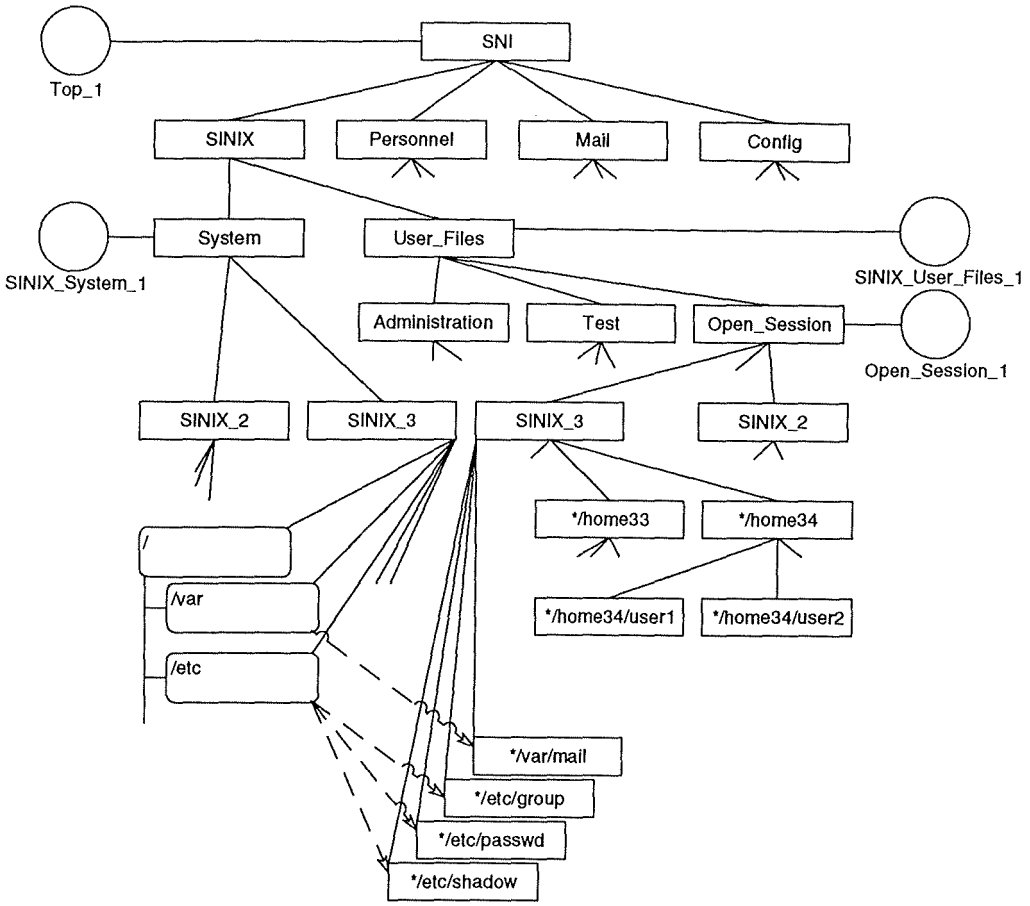

Figure 5-5: Entity structure and management classes

5.4.3 Computed policies

We will now see how the system will use the entity structure, the management classes and the priorities to compute the policies for the different file system elements. When the system looks at the entity structure, it can see for each entity which management classes have to be computed in which way to give the entity's backup policy.

The top entity SNI will of course be processed according to the 'Top_1' management class. The policy for System and its descendants will be computed from the 'Top_1' and the 'SINIX_System_1' management classes and the values for the different attributes can be seen in the following table. In this example, due to the fact that most priorities of the 'Top_1' class are free, the values of the 'SINIX_System_1' management class win.

Attribute	Value	Priority
Schedule	3-monthly	free
Daytime	backup before : 3h	free
User triggered backup	yes	free
Serialization	static attempts : 3	restrict.
Dependency	no	free
Partial backup	no	free
Multiple backup method	parallel	free
Verification	none	free
Encrypted transmission	no	free
Compression	no	free
Encryption	no	free
Browse time	none	free
Recovery rights	-	free

	Pool	Backup	Priority
Scheduled backups	system	full	free
additional backups		-	
		-	
user triggered back.	system_add	-	free
		-	

Table 5-10:Computed policy for 'System'

Notice that in table above, the last column with the priorities is in fact not needed for the processing of the System entity but we've represented it for the practical reason that the priorities could be needed below in the structure to compute policies for other entities.

As User_Files has been assigned to the 'SINIX_User_Files_1' management class, this one must be synthesized with the 'Top_1' class to find the backup policy for the entity. The computed policy is represented by the following table :

Attribute	Value	Priority
Schedule	daily	fixed
Daytime	backup at : 20h	free
User triggered backup	yes	free
Serialization	dynamic attempts :3	restrict.
Dependency	no	free
Partial backup	no	free
Multiple backup method	grooming	free
Verification	none	free
Encrypted transmission	no	free
Compression	no	free
Encryption	no	free
Browse time	1 week	free
Recovery rights	-	free

	Pool	Backup	Priority
Scheduled backups	default_1	full, incremental, differential	free
	default_2	full, incremental, differential	
	security	full	
additional backups		-	
		-	
user triggered back.	week	-	fixed
		-	

Table 5-11: Computed policy for the User_Files entity

As in the 'Top_1' management class, most priorities are free, it's the values of the 'SINIX _User_Files_1' class that will be used.

For Open_Session and its descendants, three management classes, 'Top_1', 'SINIX_User_Files_1' and 'Open_Session_1', must be considered to compute the effective backup policy. Practically, the system will synthesize the policy found for User_Files [Table 5-11] with the management class 'Open_Session_1' [Table 5-9] to find the following policy :

Attribute	Value	Priority
Schedule	daily	fixed
Daytime	backup between : 21h-24h	free
User triggered backup	yes	free
Serialization	dynamic attempts :3	restrict.
Dependency	no	free
Partial backup	no	free
Multiple backup method	grooming	free
Verification	none	free
Encrypted transmission	no	free
Compression	no	free
Encryption	no	free
Browse time	2 weeks	free
Recovery rights	user's team	free

	Pool	Backup	Priority
Scheduled backups	default_1	full, incremental, differential	free
	default_2	full, incremental, differential	
	security	full	
additional backups		-	
		-	
user triggered back.	Week	-	fixed
		-	

Table 5-12 :Computed policy for 'Open_Session'

The only things that have changed to the policy for User_Files, are that for Open_Session the browse time is two weeks and a user's files can be recovered by all members of the team.

For the physical entities */var/mail, */etc/group, */etc/passwd and */etc/shadow, policy computation is a little bit more complex because these entities are descendants of System and of Open_Session. Their policy must thus be a synthesis of the policies of these two ancestors. This is the case we referred to before as conflict between management classes of the same level. To solve the possible conflicts between attribute values of these two policies, the last column of the priority table [Table 5-5] will be used.

We explained above [5.2] that for certain attributes no synthesis, but a merge will be made. That's why in our example, the concerned physical entities' values for schedule, daytime and pools will be those of System's and Open_Session's policies. This means that a full backup will be done according to the 3-monthly schedule, before 3 o'clock in the morning, to the 'System' pool and backup will also be done according to the daily schedule, after 8 o'clock in the evening, with full, incremental and differential savings to both the default_1 and default_2 pool and an additional full backup to the 'Security' pool.

User triggered backup is possible for these entities and if we suppose that the 'System_add' pool has a longer retention time than the 'Week' pool, the savings will go to 'System_add'.

The rest of the attributes (backup control and result control attributes) will be synthesized according to the 'same level' priorities of the priority table and these computed values will be used for the savings according to the 3-monthly and the daily schedule and for the user triggered backups.

Attribute	Value	Priority
Schedule	3-monthly	free
Daytime	backup before : 3h	free
Schedule	daily	fixed
Daytime	backup at : 21h-24h	free
User triggered backup	yes	free
Serialization	dynamic attempts : 3	restrict.
Dependency	no	free
Partial backup	no	free
Multiple backup method	grooming	free
Verification	none	free
Encrypted transmission	no	free
Compression	no	free
Encryption	no	free
Browse time	none	free
Recovery rights	-	free

	Pool	Backup	Priority
Scheduled backups (3-monthly)	system	full	free
Scheduled backups (daily)	default_1	full, incremental, differential	free
	default_2	full, incremental, differential	
	security	full	
user triggered back.	system_add	-	free
		-	

Figure 5-6: Computed policy for the considered physical entities

The purpose of this chapter has been to discuss the attributes and the possible values for backup management classes. The attributes that we have found are of three kinds. Selection attributes are used to specify when which entities are processed, backup control attributes control the tool's behavior during backup and the result control attributes are used to control the form of the backup copies. Then we've seen the priority concept and how it's used to solve conflicts between different management class values. The chapter has then been completed by an example illustrating the previously explained theory.

6. Realization aspects

In the previous chapters we've tried to explain the different concepts and mechanisms of a possible management class system. To complete this view, we will now discuss certain practical aspects that must be considered when implementing the management class system in a heterogeneous distributed environment.

6.1 The different nodes

We must not forget that the main purpose of the management class system has been to ease the storage management in a distributed environment. When discussing how such a system should work we must immediately define the different kind of nodes that exist in a distributed environment.

There are first of all the **active nodes** which have a client module which allows them to make certain operations concerning the storage management. On the other side there are the **passive nodes** which are those on which no sign of a storage management system can be found. Passive nodes can be those platforms for which a client module has not yet been developed or where all the benefits of a client module are not needed.

Finally there is still the server node on which a special server application runs.

6.1.1 Passive nodes

We will start with passive nodes because they are the most restrictive. As there can be no module on the client platform, if we want to offer storage management, the whole infrastructure has to be on the server. The client user can make no assignments and start no processing. This work can however be done by the administrator on the server. The operator can create an entity configured for referring to the interesting parts of the client node's file system. If the client user has some particular management needs, there is no other possibility than asking the operator to change the entities in the structure or the assignments.

Migration is not possible for passive nodes because there is no application that could do the recall whenever a migrated file is needed. For backup and archival there should however be no problem because for them, a recall is not necessary very frequent and is only done in exceptional situations. In this case, the user of the client node must ask the backup operator to do the restore.

If backup or archival of a passive node is started on the server, then the data in the parts of the node's file system that have been configured in the entity structure is accessed via the network (by NFS for example) and taken to the server where it is saved.

Whenever an element of the node's file system that's referred to by the entity structure is deleted, there is of course no way of reflecting this change also in the entity structure. It's only at the moment of processing, when the system sees that the element doesn't exist anymore, that the structure will be modified.

A bigger problem occurs however when an assigned element is moved because the structure will not be changed. When the system doesn't find the file at the indicated location, it considers that the element has been deleted. The element's assignment is removed from the entity structure and is lost. The element will now be processed as specified for its new location. This is why for passive nodes, one shouldn't assign elements that are susceptible to be moved but limit assignment to elements such as directories that are rather static.

Even if passive nodes don't allow the user to make assignments by himself, the assignments done by the operator can offer very sophisticated policies for the nodes' different directories. The users can't change the policies but this doesn't mean that the services that are offered on passive nodes are less performant than those on active nodes.

6.1.2 Active nodes

In contrast to passive nodes, all nodes having a client module can be qualified as active ones.

The main task of this **client module** will of course be to allow the users to make assignments. They can select an element of their file systems, assign it to a management class of their policy domain and by this, create a corresponding physical entity in the entity structure.

The client module also allows the user to select entities and to trigger processing. For this, the user selects either one of his file system elements or one of his entities in the entity structure. The client module then asks the server if it's ready for processing. When the server is ready, it tells the client to send the data. According to the client/server philosophy, the workload of the task is shared by the two actors. The client module groups the data and compresses it before sending it through the net. This allows to increase the transmission rate. The server then decompresses the data and saves it. It's possible that the server saves several clients in parallel.

The client module gives the users the possibility to consult the index of the saved data and when the users select some element for recovery, the server compresses the data and sends it to the client that makes the restore.

For active nodes there are two possibilities : either there is a daemon or there is none. Such a **daemon** is a prerequisite for migration because it must, once the fixed watermark is passed, start migration. On the other side, the daemon must see when a migrated file is needed and recall it. The daemon cannot only be used for migration but it can also detect the changes that are made to the file system. If an assigned element is deleted or moved, the entity structure will be modified.

A daemon requires that each time when certain events, such as deleting an assigned element, passing the watermark or accessing a migrated file, take place, the

operating system generates some messages that will be intercepted by the daemon. There are also certain basic services of the operating system that must be modified. The file access command must for example be modified so that the file is first recalled to the on-line level before it can be accessed. This requires thus a certain number of modifications to be done to the operating system and makes the system less portable. We see however no possibility to make run a daemon on a node without doing the necessary modifications to the operating system, if this one doesn't offer the necessary interfaces.

Whenever for some practical reasons it's not possible to implement for a certain platform a daemon, migration is not possible at all and backup and archival are only possible with the limitation that assigned elements that are deleted from the file system, will still be represented in the entity structure. Depending on the file system, moved files can stay assigned. On a UNIX platform for example, the entity concept could be adapted in that sense that it not contains the path of a file system element but points to its i-node.

Depending on the needs, there can thus be different kind of client modules on active nodes. For backup and archival it's possible to do without daemon but for migration, it's absolutely required.

6.1.3 Server

The server is the node on which runs the server module and which controls the whole storage management system. It is not only responsible for offering backup, archival and migration services to the other nodes but it can also offer these services to itself. This means that the server node can be its own client.

The fact that there is one backup server in the system could make people think that all backup data is necessarily saved on that node. Such a centralized saving would however be the bottleneck of the storage management system. If all the data must be saved on the server node, an important network traffic is generated around the server and this slows the system down. It's the same for recall. Here too, the system is slowed down by the fact that all the data has to pass through the server node.

Thus the idea to make a difference between the server on which runs the server module, called the logical server, and the data servers with their data module, on which the data is finally saved. This allows to have a **logical server** that manages how the backup will be done and **data servers** that make the savings. The traditional view, where the server does both tasks is still possible by putting server module and data module on the same node but it's also possible to build a more effective system where both tasks are located on different nodes. The difference to the system with only one server is that here, the logical server tells the data server which data must be saved and then, when the data server is ready, it makes the request to the client.

6.2 Centralized approach

Before we can say exactly how this client module will work, we first have to discuss where management classes and entities will be located. If for passive nodes,

all information concerning entities and management classes must be located on the logical server, for active nodes a distributed approach would be possible.

We propose however to use the centralized approach. For the management classes, this has the big advantage that there is no problem of inconsistencies between the original management classes on the server and possible copies on the client nodes. For the users this means of course that when they want to make an assignment, their policy domain first has to be transferred from the server to their node.

It's the same for the entities, they too should be managed in a centralized way. When now a user wants to make an assignment, the information about the entities he already owns, have to be downloaded from the server. In case he wants to know how a certain entities will be processed, it's the server that computes the different management classes that must be used and sends the result to the client module. All these transfers require of course some network traffic, but this should not influence the performance too much because the volume of the transferred data is not too important and assignment is not so frequent.

Computing the management classes on the server and sending the result to the client nodes has also the practical advantage that this functionality doesn't have to be implemented for the client modules of the different platforms.

When a scheduled processing is executed or when the administrator has selected a management class for processing, the names of the concerned entities can be found in the management class. The server module then uses the entity structure and the management classes to compute the policies for the different entities and than sends them to the data server. When this one is ready, it makes a data request to the active clients. The client modules compress the data and send it to the data server. The data is then decompressed and saved as specified. For selective processing triggered on the logical server, the procedure is quite the same.

Another advantage of the centralized approach is that from a management point of view, there is no major difference between entities on active and on passive nodes. Even if the distributed approach was adopted, all information about entities and used management classes on the passive nodes would have to be stored on the server side. The centralized approach allows to treat active and passive nodes in the same way and locate all entities and management classes on the logical server. The only difference is in processing but not in the way entities and classes are managed. When with active nodes, the data server only has to make a data request to get send the data, for passive nodes it has to make the transfer itself by accessing the data via the network.

For backup and archival, the data request send to the active client simply contains the names of the concerned elements and information concerning the transmission format. The client module then takes these elements and sends them to the server. For migration however, such a request is not absolute but there are a certain number of attributes that specify under what conditions an element is eligible for migration. This are attributes such as those specifying how much time an element must be on a certain level before it can be migrated to another level. The client module thus uses the data request to select itself the data to be migrated.

6.3 Architecture

The centralized approach has the big advantage that the logical server is the only one who is really concerned by managing entities and management classes. The client modules can thus be quite elementary. A client module must of course offer a network communication functionality. This allows to take information about management classes and entity structure to the node. When a user wants to make an assignment, it's simply the name of the entity and of the management class that are sent to the logical server and it's the server module that modifies the structure and the classes. In addition to this, client modules must be able to select, compress and decompress data.

The limited functionality of the client modules makes it very simple to develop a version for different platforms.

Until now, we have seen a certain number of theoretical concepts but we never said in what they really consist. In fact, all the concepts that form the management class system are collections of information that are used by the server module to know how to behave. Management classes specify how processing must be done. The entity structure defines what management classes must be considered for which data and so on. A good solution for managing such information will be the use of a database.

The server module must offer the administrator the possibilities to manage information about users, policy domains, management classes entities and assignments. All these information will be stored in the system database and used to compute the processing policies for the data and to determine the rights of the different users. Just as client modules, the server module offers network communication facilities. Processing can either be triggered by hand or launched at the scheduled time.

The data servers' module offers of course networking facilities to communicate with the logical server and the client nodes. The module must also offer compression, encryption and functions to save and recall the data.

Just a word about the place where the on-line index of the saved data will be located. It's in fact of no importance on which server it's located. It's however possible that the logical server has not the necessary storage capacity for such a large file and a more performant solution than having the complete index on one server would be to have on each data server an index of the data saved on that node. In this case, when a user makes a request to get the index of his backups, the logical server uses the entity structure and the database to find the data the user can recover and the server on which it's located. The logical server then tells the appropriate data server to send the index to the user. After the user has selected the data to recover, this request first passes through the logical server which then tells the data server to send the data.

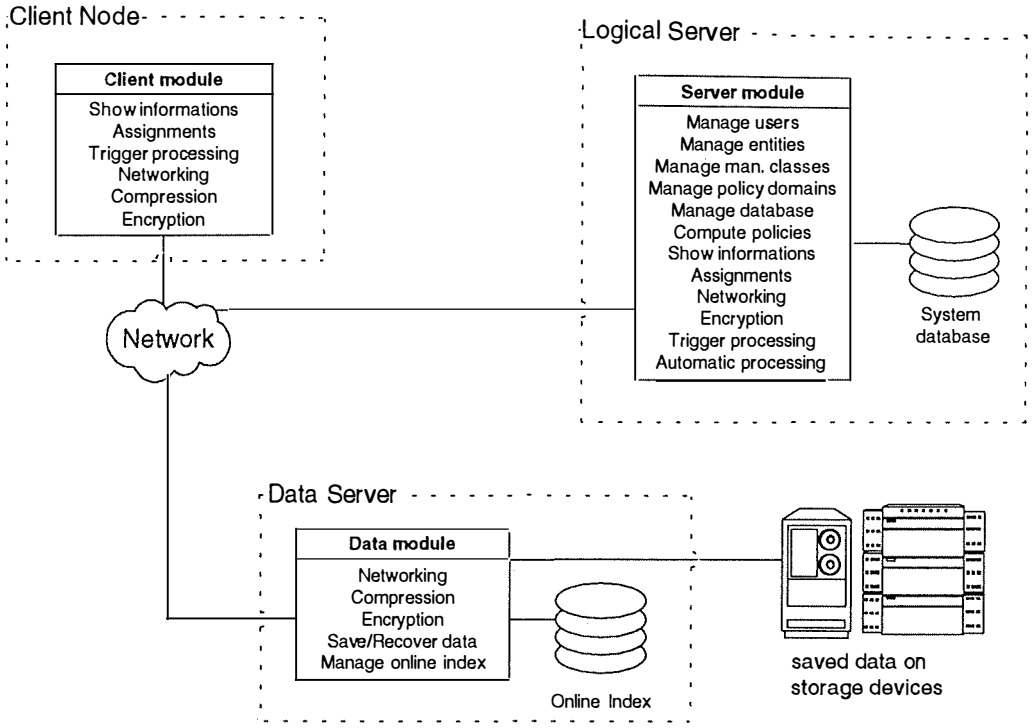

Figure 6-1:Modules and their main functions

When implementing client and server modules, a lot of work can be saved by using existing tools. In fact, it should be possible that the server module is only some kind of manager that makes the interface for the administrator but makes execute all real work by other tools. It's thus possible that the server module uses an existing database tool to manage the system database. Compression and network communication can also be done by existing tools.

It's the same for the client and data modules that could use existing communication, compression and encryption tools. It's even possible to use an existing backup tool to save the data to the devices.

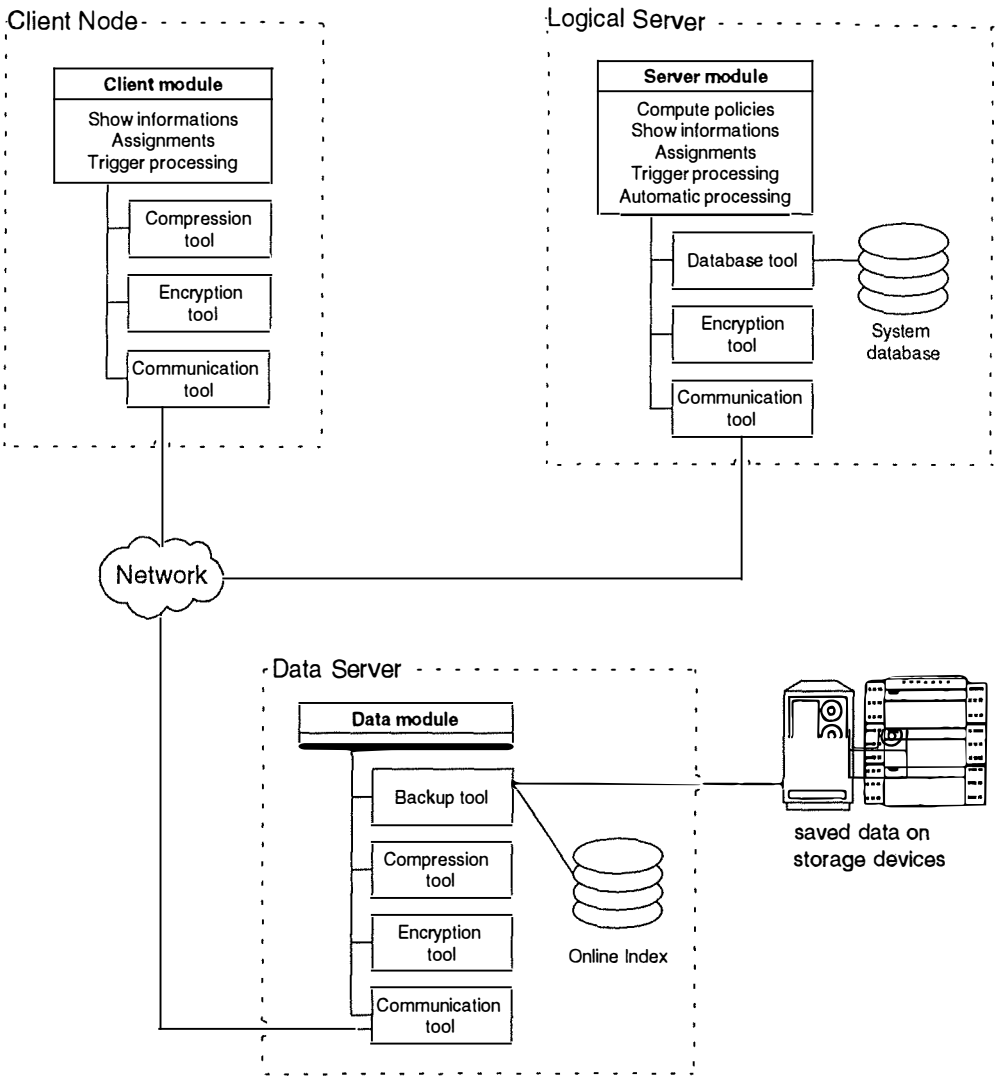

Figure 6-2:Modules and reuse of existing tools

6.4 Security

From a security point of view, it's important that nobody can bypass the storage management system and do things that he isn't allow to do.

As the system database is located on the logical server, we can be sure that users can't do modifications on it without passing by the server module. In fact, every information users demand at the client module is not directly taken from the database but the request is send to the logical server. This one checks if the user is authorized and if it's the case, sends the information to the client.

It's the same for the assignments. The user can only access his own policy domain and every assignment is checked by the server module before it's added to the database. The client module is not much more than the user's interface to the server module. Apart compression, decompression and putting data to, or taking it from the network, it does nothing itself but only asks the server module to do it.

When the user wants to trigger processing of some element, the logical server first checks if the user is authorized to do this operation and only then, the data server is allowed to send the data request to the client node. The server also controls

recall and recovery because it only lets the user select the data in the index for which he has access rights. The user's choice is then passed to the logical server which tells the data server to send the requested data to the client.

The data servers too must be protected in order to assure that nobody who's not authorized can access the backup data that's managed by the server. The backups must only be accessible via the data module and this one can only be controlled by the logical server's module.

It's very good that users can only do things they are authorized to do, but how can the server be sure that the user that makes a request, is really the one he pretends to be? For this, accessing the client module requires the user's user-id and his password. The password can also be pre-configured so that by logging on to the platform, the client module is automatically unlocked. Authentication between client node and servers can then be done using an authentication protocol such as Kerberos. The logical server can thus be sure concerning the identity of user on whose behalf the client node's module is really running.

Another security aspect is the communication between clients and servers or between servers because it's always possible that the transferred data could be intercepted. The messages that are used to coordinate the different modules are not too large so that it will be possible to encrypt them. The amount of backup data however is much more important so that it's not possible to encrypt all the data transmissions by default. The question is thus, if it's really necessary to have the data encrypted before putting it to the net.

This depends on the services the client and server modules use. In fact, if communication is offered by a special tool, it can be that the security problem is already solved at this lower level or that it makes part of the communication protocol. It's however also possible that the communication is not secure or that client and server modules do the communication themselves. In this case, it's the nodes' encryption modules that are responsible for security.

Encryption is not for free but it takes some computing time. It's thus better not to generalize encrypted data transmission to all transfers but only to the data for which it's really needed. That's why this functionality is controlled by the 'Encrypted transmission' attribute in the management classes.

The architecture that emerges out of this chapter is that of a system of active and passive client nodes, respectively with or without client module. The logical server is responsible to manage the whole system and data servers actually save the data. We took a closer look at the architecture of these modules before going over to the security issues. We only pointed out a certain number of aspects such as authorization or communication security but we can't of course pretend to be complete at this subject, which is much too vast to be entirely discussed here.

7. Conclusion

The main objective of this work was to study in how far the management class concept could be used to control the storage management in heterogeneous distributed systems.

The designed system allows to assure good and even customized storage management in this particular complex environment with a minimum or even no user intervention at all. The management classes had thus to be defined in such a way that they form some kind of formalism that can be used to specify storage management tasks so that they can be automated.

To find a compromise between the two important aspects of management classes, their performance and the ease of their use, the entity structure concept plays an important role. It allows different people to express their opinion on how a certain object should be managed and it's the tool that uses predefined conflict resolution priorities to synthesize these opinions and to decide how management will finally be done. The fact that important attributes, such as the schedule or the number of backup copies are probably fixed at the high levels of the structure by the administrator or an entity manager reduces the number of interesting attributes in ordinary users' management classes. Because of this, we could imagine that users loose interest in assigning objects themselves.

This is however not a drawback of the management class system itself but it's due to the way it's used. In fact, the administrator is absolutely free to configure the system as he wants. There is no mechanism to assure that there are no omissions, redundancies or contradictions in the defined policies and the administrator is alone responsible for the correctness of the policies. The administrator's liberty to define what he wants is in the same time a strength but also the biggest drawback of the system.

A big advantage of the management class system is that it doesn't need the file systems to be modified. As the main components of the system are centralized on the server which offers the services to the clients, the client modules can be very light, which is an important advantage to build modules for multiple platforms. It must however be accepted that client-side daemons, needed for migration, can't always be implemented without doing some modifications to the operating systems.

In the third chapter we have introduce potential attributes for backup, archival and migration but in this study only the backup aspect has completely been developed [Chapter 5]. The selection of attributes and the development of management classes for archival and migration would be a good subject for further work. The next logical step would then be to build prototypes of the different modules.

To conclude, we can say that, if the initial objective was to study in how far management classes could be used to do storage management, it has become more and more clear that the system that emerged out of this work, must not stay strictly limited to this domain. In fact, the fundamental concept and methods are completely independent from the signification of the management class attributes and its thus possible to define management classes not only for storage management tasks such as backup, archival or migration, but also for a lot of other tasks such as security or performance management of the computer system.

8. Bibliography

[IBM, 93] *ADSTAR Distributed Storage Manager/ Administrator's Guide*, Release 1, IBM Corp. July 1993

[Sieme, 94] *NetWorker V4.0 (SINIX V5.41) Leitfaden für Systemverwalter*, Siemens Nixdorf Informationssysteme AG, 1994

[Hixon, 94] Ronald Hixon, *Legato NetWorker*, Unix Review, June 1994

[Sieme, 96a] *HSMS : External Interface Specification*, Siemens Software S.A., 1996

[Sieme, 96b] *BS2000 als Backup-Server in einer offenen Welt*, Siemens Nixdorf Informations-systeme AG, 1996

[Intel, 96] *Hierarchical Storage Management for the Distributed Client/Server Environment*, White Paper, Intelligent Solutions Inc.

[Mango, 94] Greg Mangold, *HSM and Backup Products Differ by Design*, White Paper, August 1994, http://www.sresearch.com/search/105141.htm (January 1997)