MASTER IN COMPUTER SCIENCE

Study of the Aggregate Concept in the frame of a System-managed Storage Environment

Weiser, Henry

Award date:
1993

Awarding institution:
University of Namur

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique
Rue Grandgagnage, 21
B-5000 NAMUR

Study of the Aggregate Concept in the frame of a System-managed Storage Environment

Henry Weiser

Mémoire présenté en vue d'obtenir le grade de
Licencié et Maître en Informatique

Promoteur : Jean Ramaekers

Année académique 1992 - 1993

Abstract

This thesis deals with system-managed storage concepts. Though the need for system-managed storage was established in the mid 1980s, the main concepts continue to evolve. This paper gives an overview of the storage concepts, and develops the aggregate concept. The aggregate concept aims to guarantee the consistency of semantically related data objects in a system-managed storage environment, by providing aggregate-level management functions. It is developed in the frame of the SNI BS2000 environment.

Résumé

Ce mémoire aborde les concepts de gestion par le système de l'espace de stockage. Bien que le besoin d'une solution automatisée a été établi au cours des années 80, les principaux concepts continuent à évoluer. Ce mémoire décrit une vue d'ensemble des principaux concepts, et étudie le concept d'aggregat. Ce dernier vise à garantir la cohérence de fichiers sémantiquement liés dans un environnement géré par le système, en fournissant des fonctions de gestion au niveau groupe de fichiers. L'étude a été menée dans le cadre de l'environnement BS2000 de la société SNI.

I would like to show all my gratitude

to my promoter Mr. Ramaekers, for all the helpful advises he gave to me;

to Mr. Piperakis, who made it all possible;

to the team leader Mr. Hucq, for his welcome, his availability and his contribution to the study;

to the teams SWN15 and STM OS226;

to all the people who helped me in this work.

Table of Contents

Introduction

One of the most important function needed in a computer system is storage space. Unfortunately it is not possible to produce an ideal memory support that would be infinitely fast, infinitely large, infinitely reliable, directly accessible, and cheap.

Rather than an ideal memory support, a wide spectrum of storage technologies is available, each technology having its own advantages and disadvantages. Combined with other factors like the ever growing amount of data or new service requirements, storage administration problems quickly reached significant proportions.

The system-managed storage concepts were developed to meet these problems. Though the main concepts were introduced in the mid 1980, they continue to evolve. This paper contributes to their evolution by introducing the aggregate concept.

One of the major concerns of storage management is data consistency. Present management systems usually guarantee the consistency of individual data objects. The aggregate concept aims to enhance storage management functions for semantically related data objects. It aims to solve consistency problems among related objects in a system-managed environment.

The study was conducted in the frame of the Hierarchical Storage Management System (HSMS) product of the SNI - BS2000 environment.

Part one presents a general storage management framework. The first chapter gives the main trends of the mainframe environment evolution, with their impact on storage administration activities. Then an approach of system-managed storage concepts is presented in order to build a framework for the aggregate concept study. The approach is integrated in a hierarchical storage manager. Some concepts are illustrated with the HSMS product.

Part two presents the aggregate concept study. The first chapter reviews the motivations and the general requirements. Chapter two presents the main feasibility issues for the aggregate object. The issues are quite general, though some are linked to the HSMS - BS2000 environment. Chapter three integrates the aggregate object in the system-managed storage pool concept.

Appendix A presents a survey of the BS2000 environment. The important concepts of two systems are briefly reviewed: the Data Management System (DMS) and the Hierarchical Storage Management System (HSMS).

Appendix B presents an example of a BS2000 installation in terms of storage capacity and backup management policies. Some figures are quite interesting, and help to understand the issues behind system storage management.

Part One :

General Storage Management Framework

Table of Contents

1.1. Evolution of the mainframe environment

1.1.1. Introduction

This chapter presents an overview of the evolution of the mainframe environment. This is essential to understand the issues of storage management. Two main aspects are presented:

- the evolution of storage devices;
- the evolution of data.

In the frame of these two aspects, a third aspect is presented: the evolution of storage administration.

1.1.2. Evolution of storage devices

Since the early days of modern computing, the need for large storage capabilities has become more and more apparent. Backed by technological changes, the main trend in the evolution of storage devices is a constant race to larger capacities, higher access speeds, and lower costs. As mentioned in [Matick,10], the major factors responsible for these trends can be summarized in terms of two fundamental laws:

1) Law of Expanding Computer Power

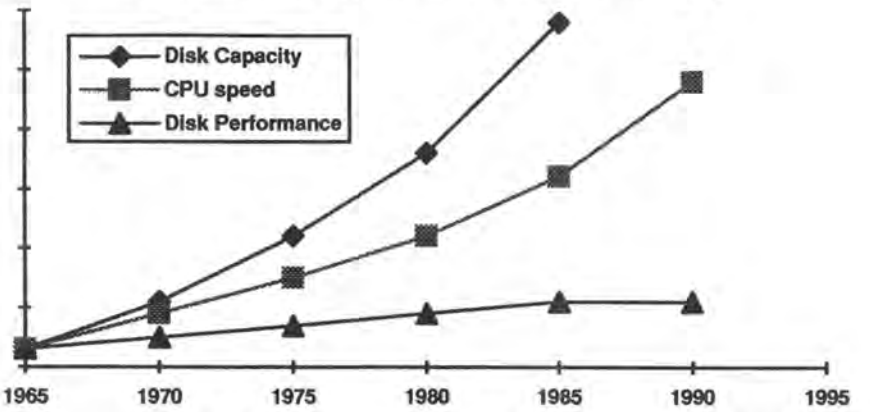
   The law of expanding computer power states that problems expand to fill the power of the CPU allowed for their execution. The complexity of the jobs is always growing, demanding more and more CPU power.

2) Law of Expanding Storage

   The law of expanding storage states that problems expand to fill the storage allowed for their completion. The amount of data to be processed is always growing, requiring more and more storage capacity.

The intensive research efforts led to a drastic progression in all the components of mainframes. The figure 1.1 illustrates this progression, but also the outbreak of a performance gap. Since the introduction of IBM's System 360 in 1965, the capacity of a disk has increased 390 times, processor performance has been multiplied by over 220, whereas the performance of a disk has only improved four times.

Fig 1.1. Device evolution and Performance gap



The discrepancies in the evolution created very early performance gaps between the various components, like between logic (CPU) and main memory, and between main memory and disks. In the 1980s, research led to the introduction of new enhancements to help to bridge the gaps, like cache memory in disk control units, solid-state disks, multi-port controllers and tape buffers. Despite significant improvements, the performance gap between processors and I/O subsystems continues to widen.

Another direct result was the development of a variety of new storage devices, like magnetic tape cartridges, optical disks CD-ROM or WORM, and opto-magnetic disks. Each device has its own characteristics in terms of performance, capacity, access method and cost. Other developments include limited automatic shelves up to fully automated (robotized) library systems. New ideas were introduced to take better advantage of existing technologies, like the RAID technology (Redundant Array of Independent Disks) which mainly enhances data availability and I/O concurrence.

Considering capacity, I/O performance and cost per megabyte, each storage technology fits in a hierarchical organization. Figure 1.2 gives a general survey of the storage hierarchy, and shows that as capacity increases, both performance and cost per megabyte decrease accordingly.

Many computer manufacturers have adopted this hierarchical approach. As we will see, it is a sound principle for efficient resource utilization. Each storage technology of the hierarchy is reviewed in terms of:

- availability: the device is on-line (always accessible) or off-line (accessible after a mount operation);
- access methods: the device supports random and/or sequential access;
- sharability: the device is dedicated to one task, or can serve several tasks;
- reliability: the device has some protection against failure or not (fault tolerance).

Level 1: Processor - Main Memory

This top level includes the registers and the small amount of built-in memory of the processor, and fast main (DRAM) memory. Our study does not consider this level.

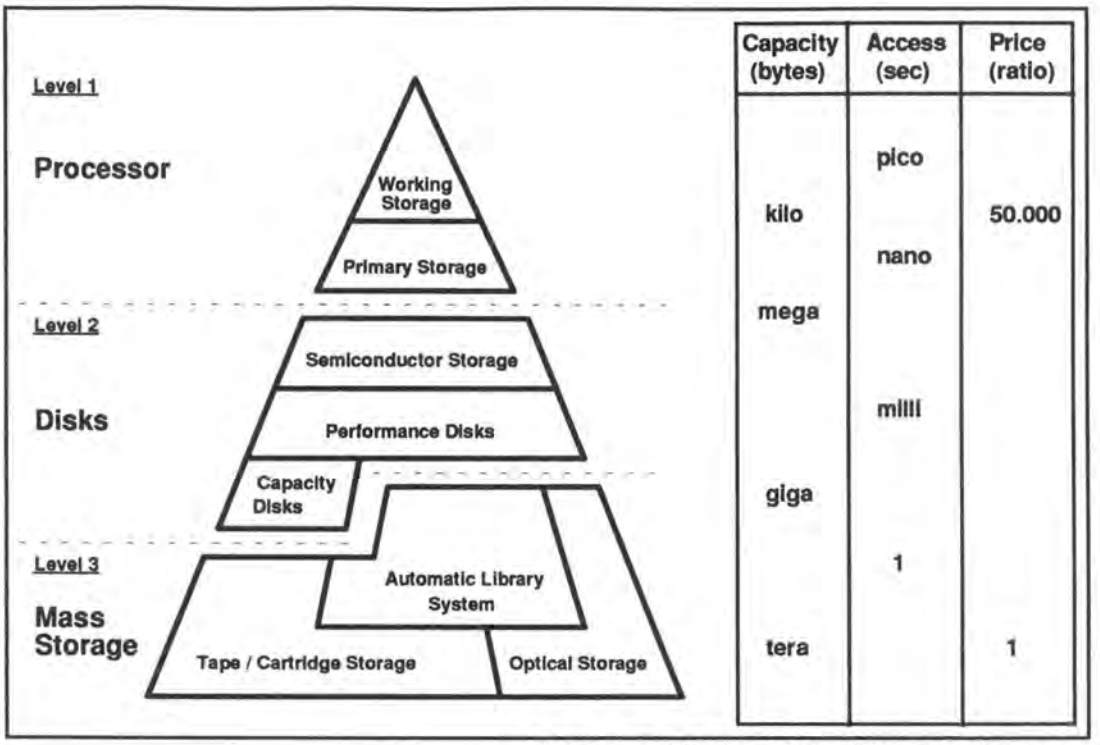


Fig 1.2. Definition of the Storage Hierarchy

Level 2: On-line Secondary Storage

This level deals with fast on-line storage (mainly disk technology). Solid state disks are usually made of semiconductor memory equipped with a backup battery. Capacity disks are disks where capacity is the main goal, whereas performance disks are disks equipped with special enhancements like cache memory to boost their I/O performances. Computer manufacturers propose a wide variety of magnetic disks:

Simple disk:

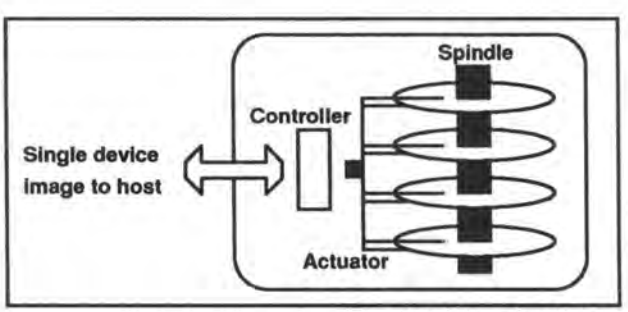


Fig 1.3. Simple Disk

A simple disk is an assembly of a set of magnetic disks and a set of read/write heads (see figure 1.3). The magnetic disks are piled up and share a unique rotation axis (spindle), and all the heads are mounted on a single actuator. The controller drives the actuator for all the I/Os.
This kind of disk is able to serve one request at the time (single device image to host), and has no protection against failure.

Multi-actuator disk:

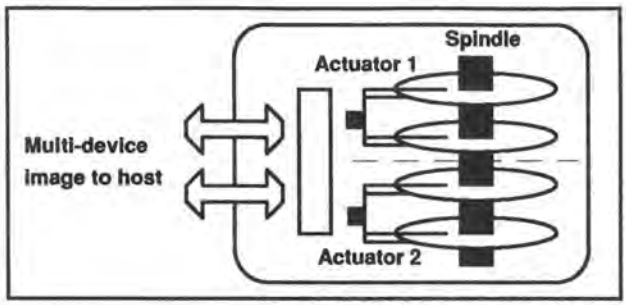


Fig 1.4. Multi-actuator disk

In figure 1.4, the read/write heads are mounted on several independent actuators.
This kind of architecture enhances concurrency, as several requests (one per actuator) may be served simultaneously.

Dual copy disk:

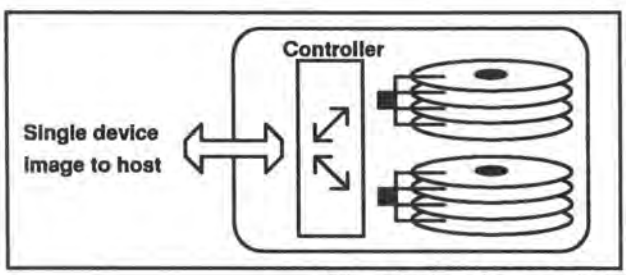


Fig 1.5. Dual copy disk

A dual copy disk (see figure 1.5) is equipped with a special internal controller and an array of two independent disks. The internal controller ensures a continuous data mirroring by writing the same information once on each disk.
This kind of architecture enhances the reliability of the complete disk. In case of failure of one of its disks, the dual copy disk is able to continue operating without any loss of data or performance.

Cache disk:

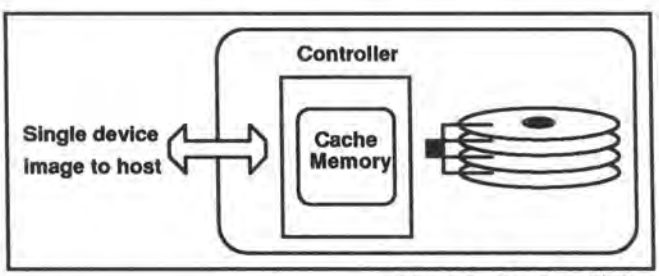


Fig 1.6. Cache disk

A cache disk (see figure 1.6) is equipped with an intermediate DRAM memory, called cache memory. In addition to the magnetic disks, the internal controller stores in the cache memory a set of most recently used records. A record residing in cache is immediately available for a read operation. A write operation is only written into cache, and is downloaded on the disk a bit later.

The caching technique increases the average I/O performances.

Redundant Array of Independent Disks (RAID):

A disk array groups multiple disk devices in a single logical volume (see figure 1.7). The RAID technology aims to organize the disks of the array in order to achieve major benefits:

- increase performance, by working the components in parallel;
- boost total capacity of a volume, by grouping many disks;
- increase the global reliability of the volume, by storing redundant data on one or several disks. In case of failure of a disk, it can be replaced and the system is able to rebuilt the lost data.

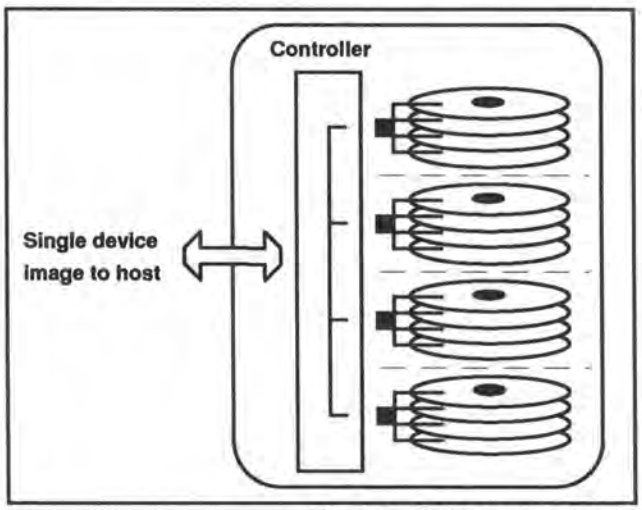


Fig 1.7. RAID technology

RAID is based on the distribution (striping) of data across multiple synchronized disks. The striping can be done on a bit level or on a sector level. Five RAID levels are established, and each level has its advantages and its trade-offs [Alford, 8].

Level 3 : Off-line Mass Storage

This level is the base of the storage hierarchy. It consists mainly of magnetic tapes, magnetic cartridges, and optical (read only) or opto-magnetic (read and writable) disks. These devices are removable, so they can be gathered together to form mass storage facilities. They often require the support of an operator (for mounting), although fully automated libraries are now available.

It is interesting to compare magnetic disk, optical disk and magnetic tape/cartridge technologies. The following table (figure 1.8) shows that each technology has its specific advantages and disadvantages.

Fig 1.8. Comparative of Storage Technologies

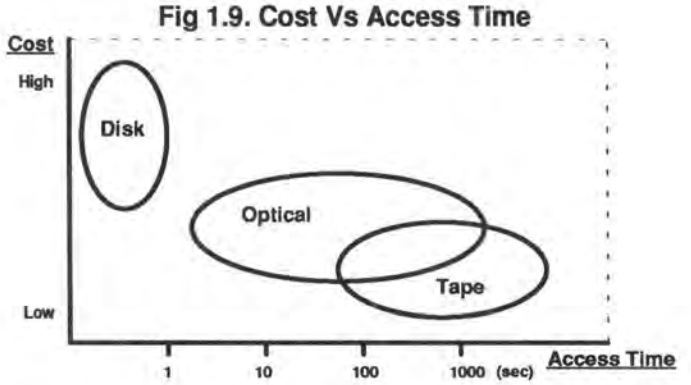
	Online	Random Access	Sequential Access	Trans-portable	Data Lifetime	Cost per Mb
Magnetic Disks	✪✪✪	✪✪✪	✪✪✪	■	✪	✪
Optical Disks	✪	✪	✪	✪	✪✪✪	✪✪
Tape	■	■	✪✪✪	✪	✪	✪✪✪
Cartridges	■	■	✪✪✪	✪✪	✪	✪✪✪

✪ to ✪✪✪ : poor to very good.

■ : Not applicable.

The next graphic illustrates the relation between cost and access time for disk, optical and tape devices.

Fig 1.9. Cost Vs Access Time



Magnetic disks are always on-line, and offer the sub-second access time required by many applications. Tape and optical storage technologies are now beginning to compete for the same market. Magnetic tape is less expensive on a per-megabyte basis than optical, and is very commonly used in the mainframe area.

1.1.3. Evolution of data

The evolution of storage devices has been backed by drastic increases of storage capacity needs. Computers are present in many offices in a wide range of economical sectors. In many aspects we can talk of evolution of data:

Vast amounts of data:

Mainframes clearly process but also produce more data than ever. Trends are for bigger and bigger configurations and storage systems. Mainframes have to deal with vast amounts of data.

Strategic data:

Companies increasingly entrust machines with the task of storing their data. This means that they become dependent of the machines, especially for strategic data. A major loss of data would be a real disaster for computerized companies. Computer centers must be able to overcome any kind of disruption to guarantee the availability of vital data.

New types of data objects:

Increasing computing power makes it possible to process new types of data objects, like sounds, images or video recordings. Each new object has its own requirements in terms of accessibility, storage consumption, etc. The storage system has to meet specific requirements.

New services for data objects:

Data objects require new services, like 24-hour-per-day on-line operations or mass-storage capabilities. New services have a direct impact on the way the computer center is managed. Computer systems have to respond to new constraints.

As a result, computer systems have to provide specific services to satisfy these requirements. Furthermore the management principles of computer systems have to be reconsidered.

1.1.4. Impact on storage administration

Storage administration encompasses the tasks of

1) controlling and globally optimizing storage usage;
2) guaranteeing the availability of data.

The traditional way of optimizing storage usage consisted in manual allocation and movement of data objects, mainly done by the end users themselves. The users determine the locality of each object. They are directly involved in the storage administration activities, leading to poor space optimization, low data availability and high configuration dependencies.

In the mid 70's, a cost-effective way for storage administration was to pay a small team to do the job, as the storage capacity of computer centers was ranging 50 Gigabytes.

Nowadays, it is not conceivable to employ a whole team only to manage storage in the range of 1000 Gigabytes. Other factors like

- the large variety of storage devices presently available (more diversification and specialization),
- the ever growing amount of data,
- the data-specific storage requirements,
- the new constraints due to 24-hour on-line operations

make the task still more complicated.

Complexity costs money at many levels. The computer center has to employ a large staff of qualified people, the end users lose time searching for their data, data are lost or not available, procedures are configuration dependent, and expensive storage devices and other system resources are not used at their best.

As the Eighties were the decade of new storage technologies, the Nineties are the decade of how to exploit these technologies in order to meet all the requirements in a cost-effective way.

1.2. Hierarchical Storage Management

1.2.1. Introducing the System-managed Storage

In the mid Eighties, computer manufacturers started to develop efficient tools (Storage Management Systems) to help storage administrators in their job. Space, performance and availability management are the key principles.

Space management:

This function deals with the optimization of storage use. The main principles are quite straightforward, namely:

- to place frequently used data objects on fast storage devices, in order to deliver best I/O performances;
- to redirect less used data objects towards slower and less expensive devices, in order to free space on fast devices;
- to move obsolete objects towards mass storage devices, to relieve the whole storage system;
- to delete old data objects which don't need archiving (like temporary files);
- to provide consolidation of multi-extent files (in static allocation systems), in order to reduce fragmentation of data blocks and unused space;
- to provide data compression techniques to reduce space consumption.

Performance management:

This function deals with the disk allocation process. The principle is to allocate data objects on a fast device which meets specific performance requirements.

Availability management:

This function ensures the availability of data objects. Main ideas are:

- to perform regular backup copies of all the data objects of the system;
- to provide a made-to-measure backup service: strategic data objects as well as heavily used objects must be backed up frequently, whereas there is less need to backup poorly used or read-only data objects;
- to provide efficient restore operations: catalogs of backups and precise information over the data objects must always be available;
- to provide for long-term archiving;

A storage management system should also reduce configuration dependencies and user involvement, by providing a more logical view of the storage system and by automating the various management functions. Finally, it should provide new techniques to reduce the nuisance (system stop, etc.) caused by the management functions.

The global benefits are quite attractive:

- It should reduce the time spent for storage management and therefore increase the system availability for the users.
- It should be possible to increase simultaneously the number of users and the number of applications supported at once by the system.
- It should optimize the use of expensive storage devices and other system resources.
- It should increase the system performance and global reliability.
- It should reduce the abends (abnormal endings), due to space allocation problems.
- It should ensure for the user a device-transparent access to any data object.
- It should increase the security of data objects.

The space, performance and availability management principles are not independent. Each one has a direct impact on the two others. For example, availability management produces several copies of data objects. Multiple copies of data objects reduce the amount of storage space left and the copy process implies a degradation of system performances.

Therefore an integrated approach is a viable solution. The following sections introduce a general survey of a solution integrated in a Hierarchical Storage Manager. We will first define a logical view of the storage hierarchy and present the concepts of processing and background levels. Then we will develop the space and availability management functions in the frame of the logical storage hierarchy. The performance management functions and the system-managed storage pool concept are introduced. Automation of the management functions is achieved with the management class and performance class concepts.

1.2.2. Logical Storage Hierarchy

A physical view of the storage hierarchy would not be very appropriate. Users would have to master extensive knowledge about the devices and their organization, and procedures would be hardware-dependent. Therefore it is useful to define a set of logical storage levels.

The number of logical levels and their extent is a compromise between friendliness of use, the flexibility required for a large variety of data objects, and the variety of devices supported by the system.

In the BS2000 environment, the Hierarchical Storage Management System product (HSMS) provides three logical storage levels (see figure 1.10):

- The first logical storage level (SL0) is an on-line level made of disk devices which have a very short access time. It offers a sub-second direct access to data objects. SL0 is a mandatory level for the HSM system.
- The level SL1 includes a set of capacity disks (on-line and short access time). Its existence is optional.
- The level SL2 is the archive level. Made of tapes and magnetic cartridges, it is off-line and is characterized by a long access time. An automated library system can be added. SL2 is mandatory for HSMS.

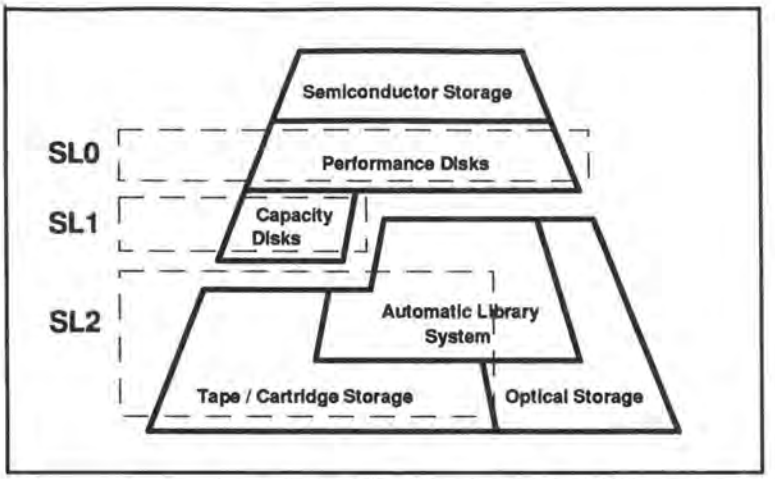


Fig 1.10. Storage levels in the HSM system

In the HSMS view, the SL0 is the processing level, whereas SL1 and SL2 are background levels (see figure 1.11). Only data objects residing on SL0 can be directly processed. The HSM system uses the background levels to store data objects that are not needed for processing. An object stored on a background level is indirectly available through a HSM function that recalls it to SL0.

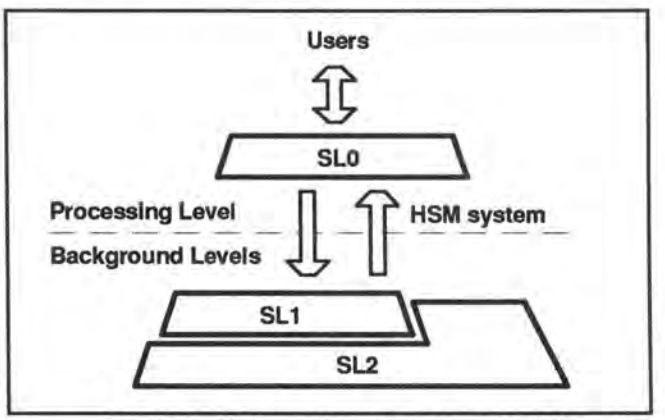


Fig 1.11. Processing / Background Levels

The reason for such a design is to ensure device-transparent access to any data object. As we have seen in section 1.1.2 (Evolution of storage devices), the various storage devices don't have the same abilities: magnetic tapes and cartridges of SL2 are only sequential devices, block sizes are device-dependent, etc. The definition of a processing level ensures that the data objects needed for processing are stored on fast and on-line disk devices.

The movement policies of objects through the storage levels are developed in the next section.

1.2.3. Managing Space and Availability in a Hierarchical Environment

1.2.3.1. Functions: migration, backup, archival

This section deals with the space and availability management functions. The functions commonly supported by storage management tools are namely migration, backup and archiving. These are developed in the frame of the HSM system of the BS2000 operating system.

Migration:

Migration is the task of moving inactive data objects towards a more cost-effective storage level. A migrated data object is still fully available through a recall function.

The HSMS product supports migration from the processing level towards both background levels, and from SL1 to SL2 (see figure 1.12). The aim is to avoid saturation of the SL0 by moving inactive data objects to lower levels.

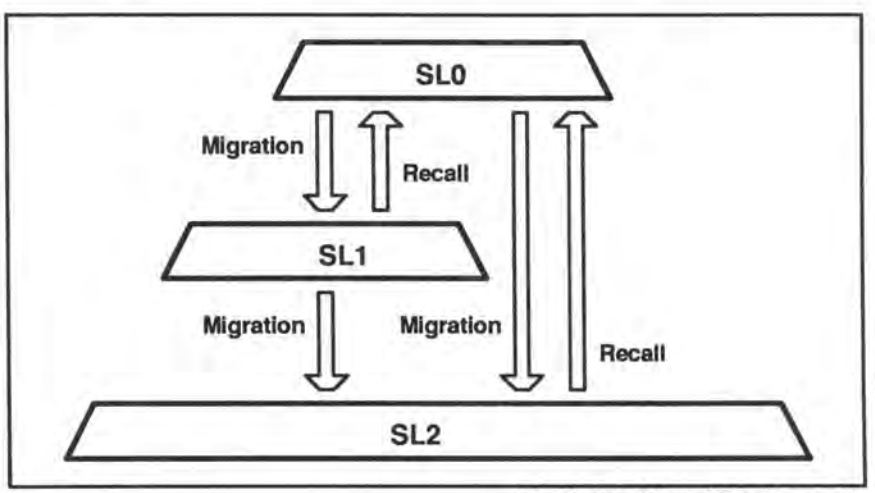


Fig 1.12. Migration / Recall

The migration of a data object produces a migration copy of the object. The original data object is deleted, but its DMS-catalog entry is preserved and marked 'migrated'. The migration copies are fully managed by the HSM system. They are stored on SL1 or SL2 in background storage units called archives. A migrated data object is available through a recall function. An attempt to read a migrated object causes a transparent recall to SL0, or a user can force a recall of an object by issuing a special command.

Backup:

Backup is the task of making short-term copies of a data object to guarantee its availability. In case of destruction of a data object (deletion, corruption ...), a restore operation can be performed to rebuild it by using the most recent backup copy of the data object.

The HSMS product supports backup/restore functions between the processing level and the background levels (see figure 1.13). Only objects stored on SL0 can be backed up.

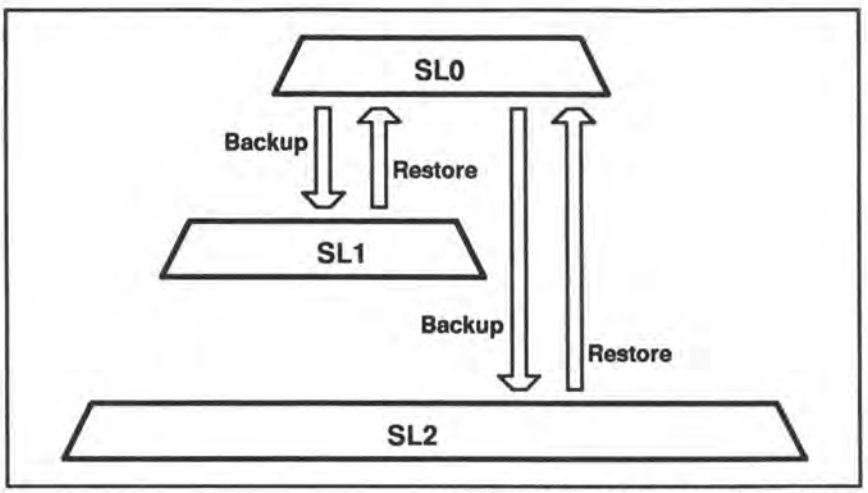


Fig 1.13. Backup / Restore

The backup of a data object produces a backup copy of the object. The original object is not modified. The backup copies are fully managed by the HSM system. They are stored on SL1 or SL2 in background storage units called archives. Each archive has an associated catalog which lists all the backup copies it contains, as well as important information's like the backup and retention dates. Backup copies are available through the restore function.

The storage administrator is responsible for making regular backups of all the data objects of the system. Besides, any user can make a backup of his data objects.

Archiving:

Archiving is the task of moving obsolete data objects towards long-term storage devices. Long-term archiving is often a legal necessity, although it can be done for private purposes.

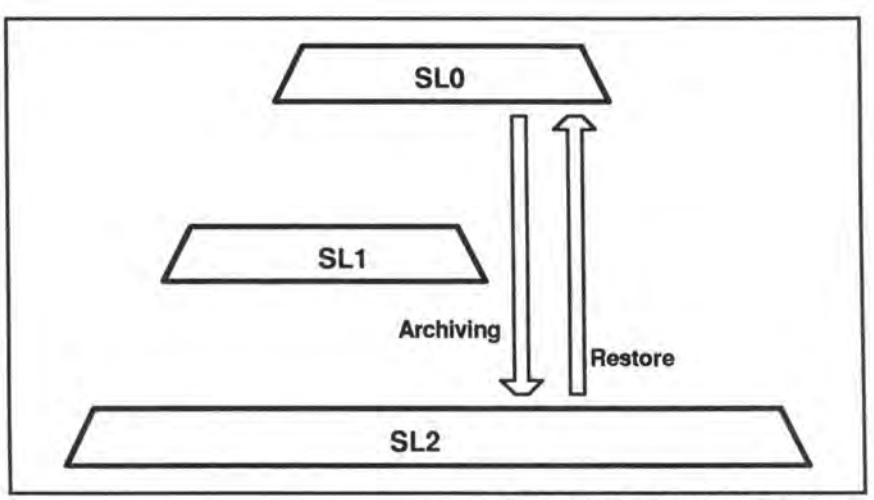


Fig 1.14. Archiving / Restore

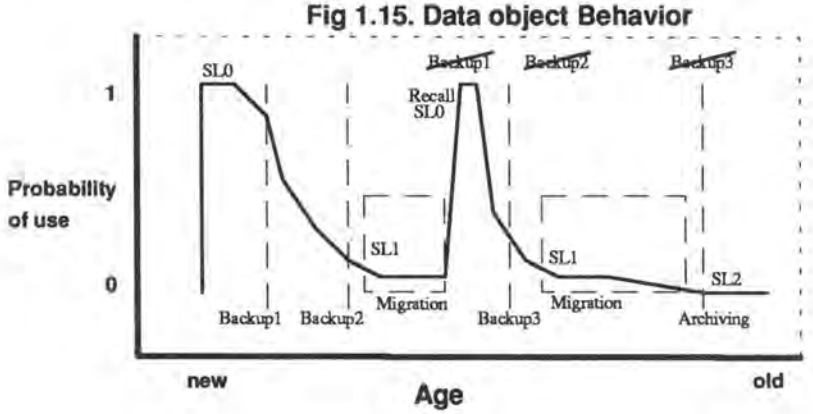
Archived data objects often don't require high availability. The HSMS only supports archiving towards cost-effective SL2 (see figure 1.14).

The archiving of a data object produces an archiving copy of the object. It deletes the original data object as well as its catalog entry. The archiving copies are also fully managed by the HSM system. They are stored on SL2 in background storage units called archives. Each archive has an associated catalog which lists all the archive copies it contains, as well as important information like the archiving and retention dates. Archived copies of objects are available through the restore function.

Migration, backup and archiving functions are quite linked. The way they work together mainly depends of the behavior of the data object. The following graphics are examples showing how the management operates in function of the probability of use of a data object.

Example 1:

Fig 1.15. Data object Behavior

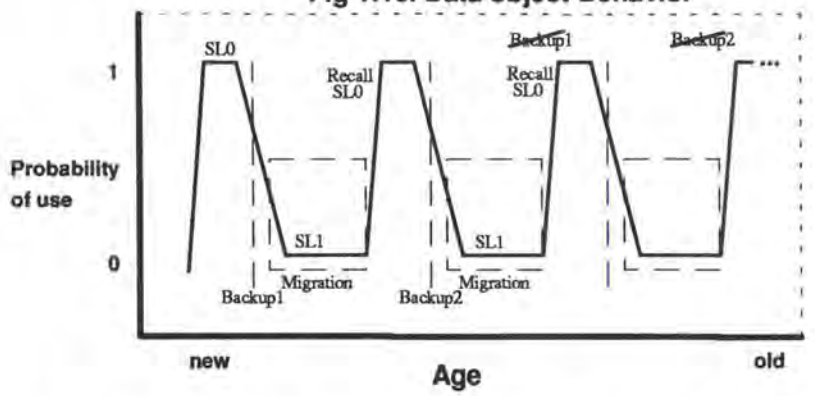


The data object of figure 1.15 is first created by the user and is highly active. The HSM system takes two backups at regular intervals to ensure availability. After a period of inactivity, the system migrates the object to a lower level (SL1). While the object is migrated, no more backups are taken because the object is not processed. At expiration date, the system deletes the backup copy backup1. The object is then recalled by the user for processing. Backup3 is taken, then the object is again migrated. Backup2 is deleted. The object ages unused, and reaches its archiving date. The system archives it on a cost-effective device (SL2), and deletes the backup3.

This can be the behavior of an electronic mail file.

Example 2:

Fig 1.16. Data object Behavior



The data object of figure 1.16 is used periodically. After the activity period, It is quickly migrated to a lower level because it is not needed anymore. For security, a new backup is taken before migration. The data object is recalled when needed.

This can be the behavior of the payroll application.

The two examples show that each data object requires a lot of management attention. This is not realistic for large storage configurations. An automated solution is a necessity.

1.2.3.2. Automation of these functions: the Management Class Concept

An automated storage management facility is a necessity for dealing with a large number of data objects. The task is too tedious to only rely on manual interventions of the storage administrator or of the users.

On the other hand, the management of a data object is quite specific to the object itself. Each data object has its own behavior and requirements in terms of availability, performance ... and these requirements should influence the way the object is managed by the system.

So the idea is to equip each data object with a set of management attributes. The system will use these parameters to automatically manage the object [IBM, 5].

Migration attributes:

- automated migration: enable / disable switch;
- delay of inactivity (days) before migration to SL1: controls migration to SL1;
- delay of inactivity (days) before migration to SL2: controls migration to SL2.

Backup attributes:

- automated backup : enable / disable switch;
- backup frequency: controls the selection for backup;
- guaranteed backup: forces a backup before migration;
- number of backups retained while the data object exists;
- number of backups retained while the data object is deleted;
- retention period of a backup.

Archiving attributes:

- automated archiving: enable / disable switch;
- expiration date or expiration period: date of archiving;
- expiration period after archiving;
- maximum retention period.

Only the owner knows the requirements and the importance of his data objects. Therefore it is to the owner to tune the management attributes.

For each data object, the new management attributes could be stored in an extension of its DMS-catalog entry. This solution allows a fine tuning of the parameters, each object being independent of the others. On the other hand, it requires a certain amount of space in the DMS-catalog entries. This is not desirable because space is usually limited and because these attributes have no sense if the HSM system (which is usually optional) is not present. Another problem is that the users could define a management incompatible with the capacity of the computer center, like always defining very long migration delays for every data object, or defining a backup to tape while there is no operator for mounting operations ...

Another solution is the Management Class. A management class is a pre-defined management policy. The storage administrator has to set up a collection of named management classes that are compatible with the computer center policy, and the users have to choose for each of their data objects a specific management class. This solution gives the storage administrator enough control over the management policies, reduces the amount of work for the users, and finally reduces the risk of errors.

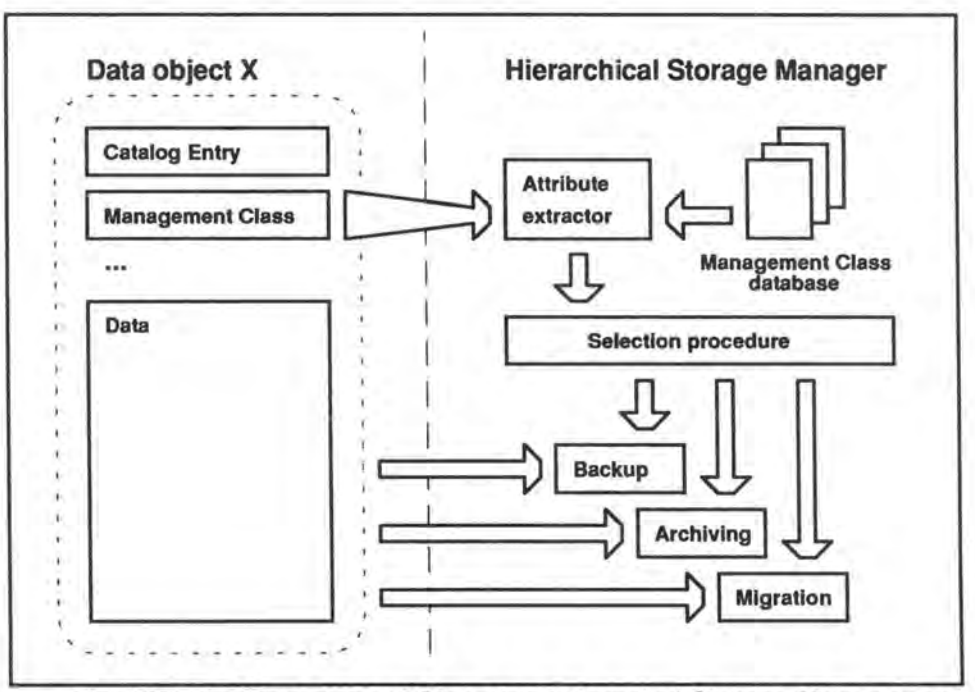


Fig 1.17. The Management Class for Automated Storage Management

Figure 1.17 shows a system-managed data object. The management class describes its requirements in terms of backup, archiving and migration. The HSM system is able to perform automated space and availability management.

1.2.4. Managing Performance in a Hierarchical Environment

1.2.4.1. Quality of service at the data object level

This chapter deals with the performance management function. This function only works in the field of the processing level SL0 of the storage hierarchy, and aims to optimize the use of fast disk devices.

In section 1.1.2, we introduced the evolution trend of magnetic disk storage. Technological enhancements can be summarized in two words: specialization and diversification. Therefore the SL0 is not a uniform storage space. Figure 1.18 shows a physical view of a processing level including a simple disk, a disk equipped with cache, a dual copy disk and a RAID.

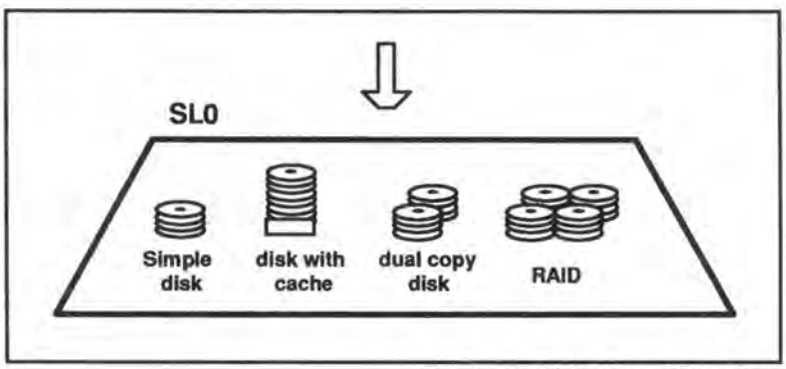


Fig 1.18. SL0 diversification

On the other hand, each data object has specific performance requirements. For example:

- a database usually requires high random access performances;
- data objects like sound or video recordings require sequential access performances;
- the system catalogs require very high availability ...

The aim is to allocate the data objects on the disk that meets at best the given requirements. It should optimize the quality of service for the data objects and globally optimize the use of expensive disk devices.

1.2.4.2. Automation of this function: the Performance Class Concept

Again, a physical view of the internal components of SL0 is not desirable. It would involve the users in the allocation job, imply configuration dependencies and lead to poor optimization results.

A wiser solution is to provide a logical view of the various disks, and entrust the allocation job to the system. For each data object, the owner must define a set of performance attributes [IBM, 5], in terms of

- random access performance: standard / high;
- sequential access performance: standard / high;

- availability (fault tolerance): standard / high;
- concurrency: standard / high ...

With this logical view, an automated solution is possible. The HSM system has now all the needed information to allocate the data objects on the disk that meets at best the given requirements. If the user modifies the performance attributes of a data object, the HSM system can reallocate it on a more suitable disk. If the settings of the processing level are changed (add of a new disk, loss of a disk), the HSM system can adapt the current allocations to the new situation. These movements are called SL0 migrations.

Like for the space and availability management, it is better to introduce the performance class concept. A performance class is a pre-defined allocation policy. The storage administrator has to set up a collection of named performance classes that are compatible with the performances of the SL0 disks. The users have to choose for each of their data objects a specific performance class.

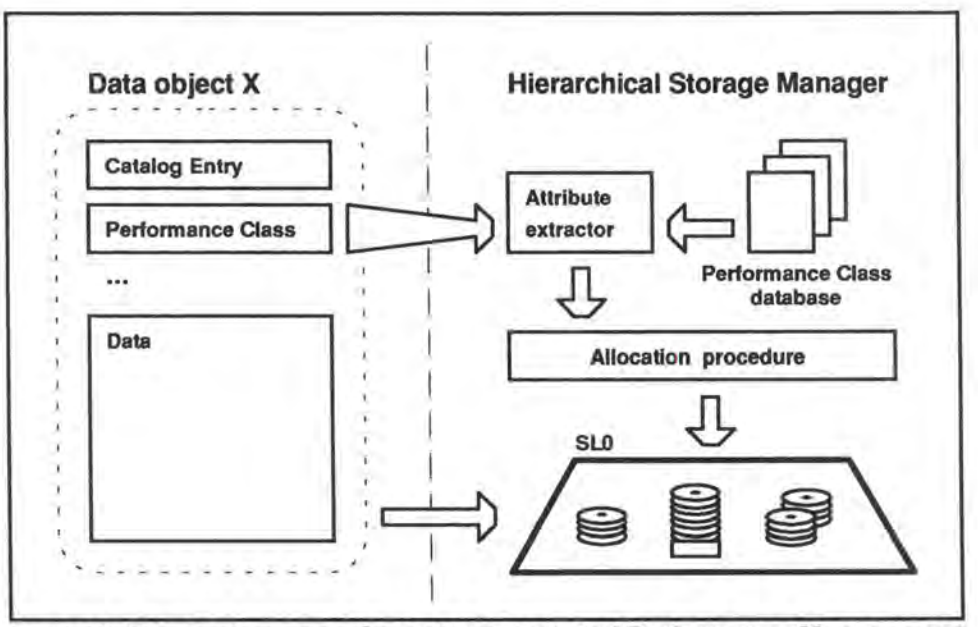


Fig 1.19. The Performance Class for Automated Performance Management

Figure 1.19 shows a system-managed data object. The performance class describes its requirements in terms of access performances and availability (fault tolerance). The HSM system performs automated disk allocation.

Note: In [IBM, 5], the performance class is introduced as the storage class concept. We have adopted the 'performance class' terminology because it seems less ambiguous.

1.2.5. System-managed Storage Pool

The storage hierarchy as previously introduced leads to a unique storage system. This is not always desirable, because:

- trends are for bigger and bigger configurations. A system-wide storage hierarchy requires a vast amount of metadata. Data object catalogs, archive catalogs etc., are likely to become inefficient. Accounting and space allocation controls become difficult to manage;
- it makes storage system reconfigurations quite difficult, because a device volume on its own is not a self-consistent entity. Space and availability management functions lead to a distribution of data and metadata among several devices. A device never contains all its related metadata.

The concept of system-managed storage pool (SMS-Pool) solves these problems (see figure 1.20). A SMS-Pool is an independent and self-consistent hierarchical storage entity, i.e. it contains all the data and related metadata.

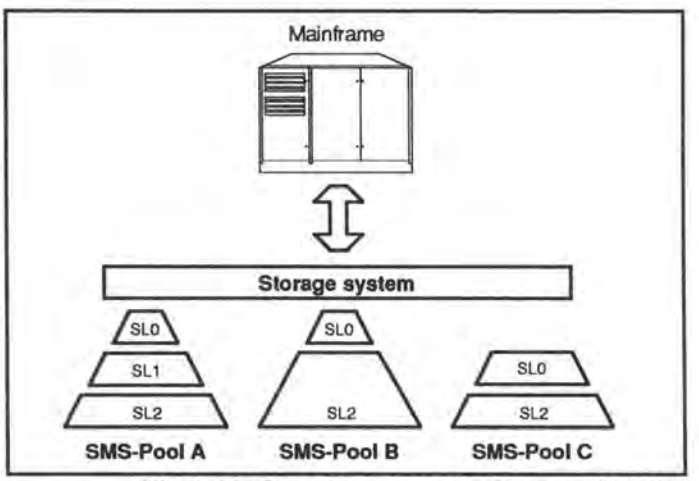


Fig 1.20. System-managed Storage Pools

The management functions (space, availability and performance) are reintroduced within the SMS-Pool entities. Therefore the SMS-Pool contains all its data objects, the related system metadata and all the metadata related to the backup, archive and migration functions.

The advantages of the SMS-Pool topology are straightforward:

- it downsizes the system-wide storage system to smaller entities;
- it eases storage system reconfigurations, as SMS-Pools are independent. A pool can be easily removed of the storage system. A pool can be added to the storage system without any disruption;
- in a business-oriented view, a mainframe usually covers the needs of several segments of a company. One SMS-Pool can be defined per business segment, and tailored to meet at best the requirements of the stored applications.

1.3. Conclusion

In the mid Eighties, computer manufacturers started to develop efficient storage management systems to help storage administrators in their job. System storage management must comply with the constant evolution of storage devices (specialization and diversification) and the constant evolution of data (vast amounts of data , vital data, new services etc).

The revolutionary principle is to migrate from a user-managed storage towards a system-managed storage. An approach integrated in a Hierarchical Storage Manager is achieved by

- defining a logical view of the storage devices: three logical storage levels are defined within the storage hierarchy (one fast processing level and two background levels), and a logical performance view is defined for the processing level;
- defining logical storage requirements for each data object, in terms of
  - space and availability requirements: to control migration, backup and archiving functions between the logical storage levels;
  - performance requirements: to control allocation within the processing level.

The management class and performance class concepts are introduced to give the storage administrator enough control over the storage policies. A data object equipped with a management class is system-managed for migration, backup and archiving functions. A data object equipped with a performance class is system-managed for the allocation function.

To avoid a system-wide storage hierarchy, the system-managed storage pool concept (SMS-Pool) is introduced. The SMS-Pool concept is a self-consistent hierarchical storage entity. The space, availability and performance management functions are reintroduced within the SMS-Pools.

This ends our general storage management framework.

Part Two :

Aggregate Concept for Storage Management

Table of Contents

2.1. Introducing the Aggregate Concept

2.1.1. Overview

Traditional data management systems and storage management tools provide services at the data object level. Data objects are viewed as independent objects.

This point of view doesn't reflect the semantic relations that can exist between data objects, like:

- an application is often a set of data objects;
- a set of data objects can have a special meaning for a user;
- a set of data objects can have a special meaning for a group of users;
- a set of data objects can be highly critical for the company.

Storage management should ensure the global consistency of related data objects. This is presently not the case because objects are independently managed. The aggregate concept has been introduced to fill this gap. It provides storage management of aggregated data objects.

The aggregate concept was developed in the frame of the HSMS product. Part one of this document presents a general storage management framework.

The chapter 2.1 describes the motivations under the aggregate concept, the general requirements and the concept definition.

Chapter 2.2 presents a study of the main feasibility issues. The aggregate concept is developed in a general HSM framework and more specifically in the BS2000 environment.

Chapter 2.3 presents an integration of the aggregate object within the SMS-Pool topology. The integration is quite linked to the BS2000 environment.

2.1.2. Motivations

The main motivation for the introduction of this concept is to enhance storage management for data objects that have something in common, therefore which need to be managed at once. Two linked motivations are consequences of the introduction of automated management:

1) The Management Class:

As described in section 1.2.3.2, the management class allows a wide automation of the backup/archiving/migration functions. All these services are data object-oriented, whereas data objects are often related to applications. The system is not aware of this. The aggregate concept aims to give a special semantics to related objects.

The automation of the availability management functions at the aggregate level leads to the concept of management classes for aggregates.

2) The Performance Class:

The performance class allows the user to give for a data object a set of performance requirements (see section 1.2.4.2). It is used by the HSM system to select the most suitable device of the processing level SL0.

This means that for a group of related data objects with various performance requirements, the objects are very likely to be distributed among many devices of SL0. Migration will distribute them further on. A device crash at the SL0 would be disastrous, because many applications are likely to be partly damaged. A data object level restore tool would not be very efficient, and cannot really guarantee full consistency of all the damaged applications.

A second motivation is to decrease the nuisance of present system backups (i.e., a complete backup of the system). To reach a consistency point, a system backup usually requires that the system is completely cut off from the users. The aggregate concept can help to reach local consistency points. The backup can then be made without a system cutoff. This reduces the nuisance for the users.

A third motivation is to provide efficient disaster recoveries. A well-tried disaster recovery plan is the best insurance against a major computer center disaster. The process of recovery at a remote site is complex. A successful recovery must be done very quickly. Some applications must be restarted as soon as possible, whereas others are less important. Therefore the administrators must provide regular and independent backups of strategical applications. The aggregate concept can help them in this task.

Finally, the aggregate concept could be very useful in distributed environments. For a storage administrator, distributed applications are quite difficult to manage, and workstation users don't wish to do it. A LAN-based backup server is a promising solution. Distributed aggregates could make part of the solution.

2.1.3. Requirements

We introduce a general list of user-related and HSMS-related requirements.
The 'aggregate object' terminology designates the implementation of the aggregate concept.

A) User Requirements

A1. Any user should be able to create, modify and delete aggregate objects.
A2. The user who created an aggregate object must be the owner of the object and its administrator.
A3. The aggregate object should be able to meet any application (or any relation) topology. In particular, the aggregate concept must be independent of the user-ID concept. Though an aggregate must have an owner (i.e., a user-ID), it can represent a set of data objects created under several user-IDs. The owner is the administrator of the aggregate.

A4. The aggregate object must ensure the confidentiality of the aggregated data objects (access security).
A5. The aggregate object must offer efficient availability functions (backup and restore operations), and provide for disaster recoveries.

B) HSM Requirements

B1. The aggregate concept must comply with the HSM concepts.
B2. A maximum of compatibility and flexibility must be ensured between aggregate-level management and the traditional management at the level of the data objects. The aggregate object should not enforce new restrictions on the use of the current functions of the HSM system:
- data objects that belong to an aggregate object should remain accessible as before for management at the data object level;
- a backup copy of a set of aggregated data objects should be restorable at the aggregate level and at the data object level.

B3. The aggregate object must fit in the SMS-Pool topology. It must be local to an SMS-Pool.
B4. Aggregate objects must be safe against device crashes, to ensure recovery.

2.1.4. Concept Definition

The aggregate concept aims to collect under a single name a set of related data objects of the Data Manager system, and to provide for aggregate-consistent availability management functions. Figure 2.1 illustrates the concept of "aggregate". It collects under a single name:

- a set of related data objects;
- a storage management policy (management class);
- a set of backup and/or archive copies of the aggregate.

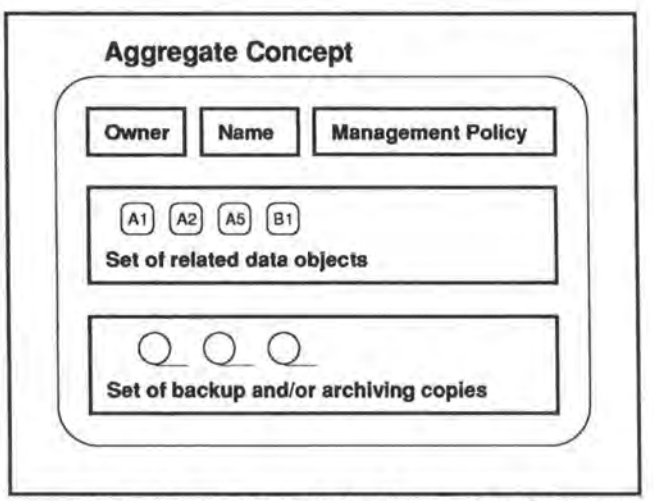


Fig 2.1 Definition of the Aggregate Concept

At a lower level, the aggregate object (see figure 2.2) is a set of data structures including:

- a special catalog entry;
- a set of management attributes;
- a link describing the related (aggregated) data objects;
- a set of archive catalog entries and tape volumes.

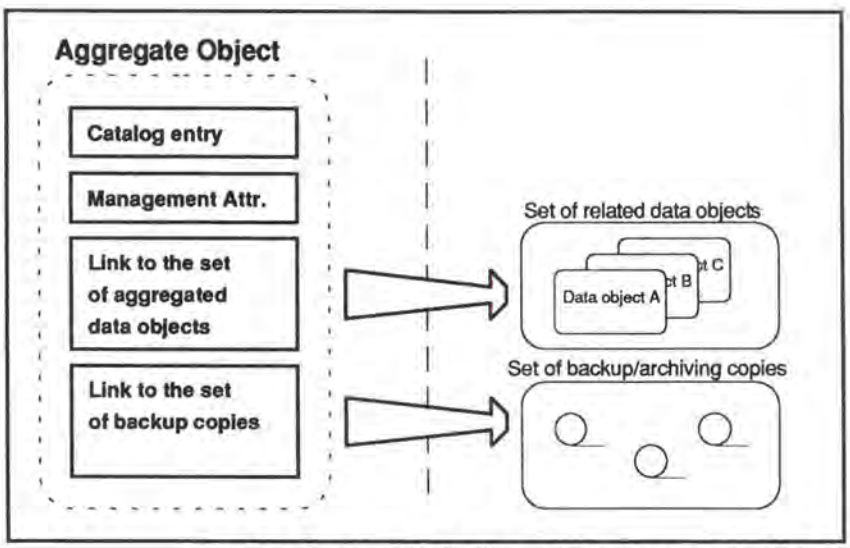


Fig 2.2. General Definition of the Aggregate Object

The aggregate object will be developed in the next chapters.

Two classes of operations belong to the aggregate object:

- aggregate *management functions* involve the processing of the aggregate object itself, like create, modify, delete an aggregate;
- aggregate *commands* involve the processing of the set of data objects that are collected under the aggregate, like backup, restore, archiving of an aggregate.

2.2. The Aggregate Object - Feasibility issues

2.2.1. Introduction

This chapter presents some of the main feasibility issues for the introduction of the aggregate concept. The study covers the following topics:

- the place of the aggregate object in a typical Operating System;
- the system privileges required for aggregates;
- the relations between data and aggregate objects;
- the relations between aggregate objects and background storage units (archives);

General solutions are reviewed, taking into account the advantages, disadvantages and possible problems. Some topics are quite general to any operating system. Others are quite related with the HSMS product of the BS2000 environment.

2.2.2. The Aggregate sub-system

2.2.2.1. Overview

In this first issue, we will discuss how to integrate the aggregate object in the Operating System. A typical Operating System is usually made of a set of independent sub-systems, where some are mandatory and others optional.

Let's consider two sub-systems: the Data Manager system (DMS) and the Hierarchical Storage Management system (HSMS). The first system is mandatory, whereas the second one is usually optional. The question is which system will manage the aggregate object.

This is fundamental because this will determine what will be effectively available for aggregate operations.

We will study

- the general prospects considering HSM aggregates or DM aggregates;
- the BS2000 environment with the ARCHIVE product.

2.2.2.2. General considerations

Two alternatives are possible.

1) Aggregates are only objects of the HSM system. The Data Manager system is not aware of the existence of aggregates This means that the HSM system provides aggregate management functions and aggregate commands.

Main consequences are that:

- extensive backup/recovery support is directly available through the HSMS;
- aggregate operations will be limited to those offered by the HSM system (backup/archiving/migration activities), unless the HSM system provides a programming interface for other products. The availability of aggregates would then depend of the availability of the HSM system, which is optional;
- aggregate requirements will mainly concern the HSM system.

2) Aggregates are new objects of the DM system. This means that the DM system provides aggregate management functions, and any product can directly make use of this new object.

Main consequences are that:

- backup and recovery support are available through the HSM system;
- aggregate operations may cover usual DM operations, or other sub-system operations;
- aggregate requirements will concern the DM system and every sub-system providing aggregate support.

The second alternative is surely the most attractive. The DM system offers the aggregate object management functions, while taking advantage of useful functions already provided by the HSM system. Although the usefulness may sometimes be unclear, any other sub-system could actually provide its own set of aggregate commands. This solution is more opened for future developments.

2.2.2.3. BS2000 environment

In the BS2000 environment, the HSM system relies directly on a kernel implementing a set of basic storage management functions. This kernel is part of a stand-alone product called ARCHIVE (see figure 2.3).

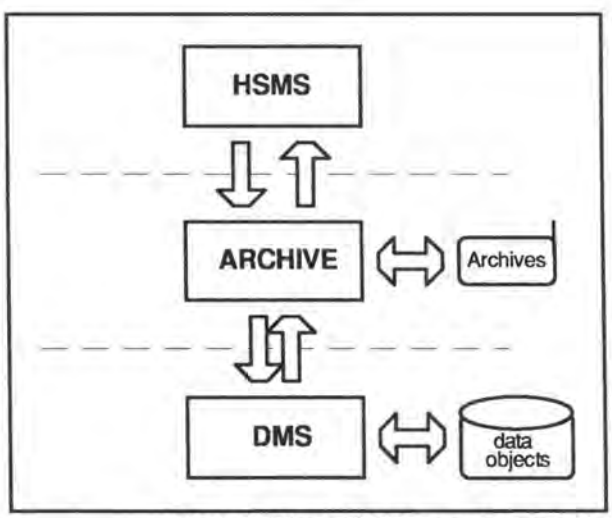


Fig 2.3. The BS2000 environment

The ARCHIVE kernel must be taken into account. So three sub-systems are considered: the DM system, the HSM system and the ARCHIVE kernel.

1) First alternative is (see figure 2.4)
- aggregate objects are managed by the HSM system;
- the ARCHIVE kernel doesn't know about aggregates;
- the DM system doesn't know about aggregates.

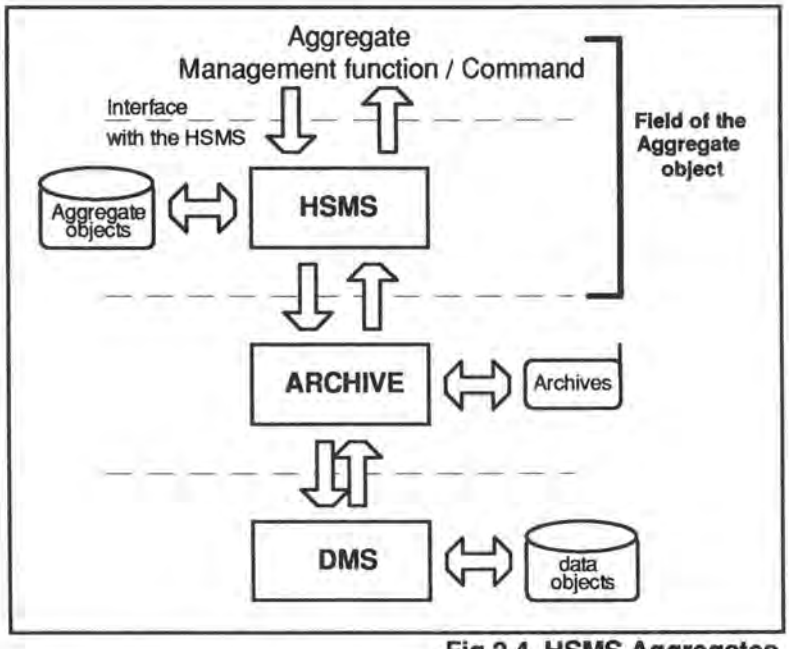


Fig 2.4. HSMS Aggregates

The main consequences are that
- the HSM system has to provide aggregate management functions and commands;
- the HSM system has to provide a transparent processing of aggregated data objects relying on the present functions of the ARCHIVE kernel;
- new requirements for the ARCHIVE kernel are very limited;
- if other sub-systems would like to provide aggregate processing, the HSM system must offer a special programming interface.

2) Second solution is (see figure 2.5):
- aggregate objects are managed by the ARCHIVE kernel;
- the HSMS product offers aggregated processing;
- the DM system doesn't know about aggregates.

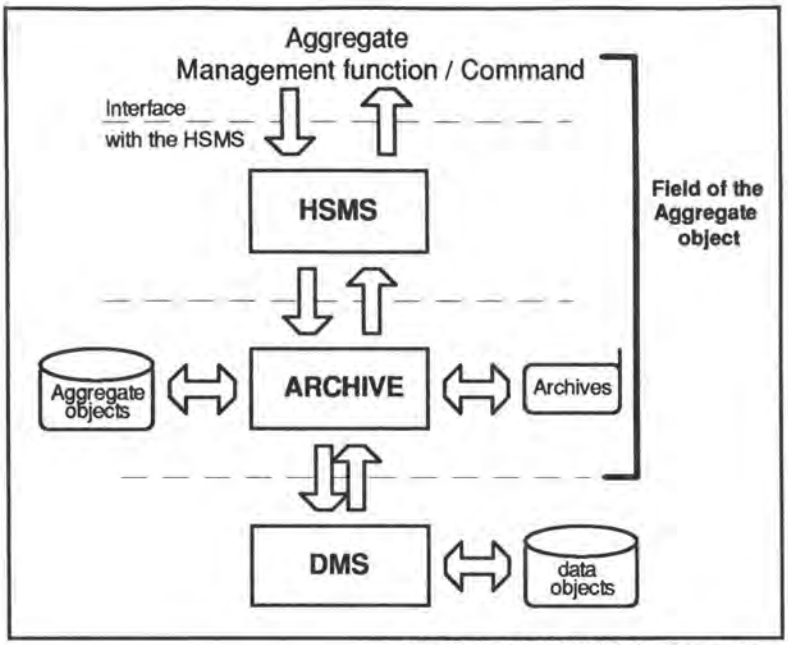


Fig 2.5. ARCHIVE Aggregates

The main consequences are that

- the HSM system can call the ARCHIVE kernel to save or retrieve an aggregate;
- the ARCHIVE kernel has to provide for a transparent processing of aggregated data objects relying on a data object DM system;
- aggregate requirements concern the HSM system and the ARCHIVE kernel;
- if other systems would like to provide aggregate-level processing, the ARCHIVE kernel must provide a special programming interface.

3) Third solution is (see figure 2.6)

- aggregate objects are managed by the DM system;
- the HSM system and the ARCHIVE kernel provide support for all DMS objects, including the new aggregate object.

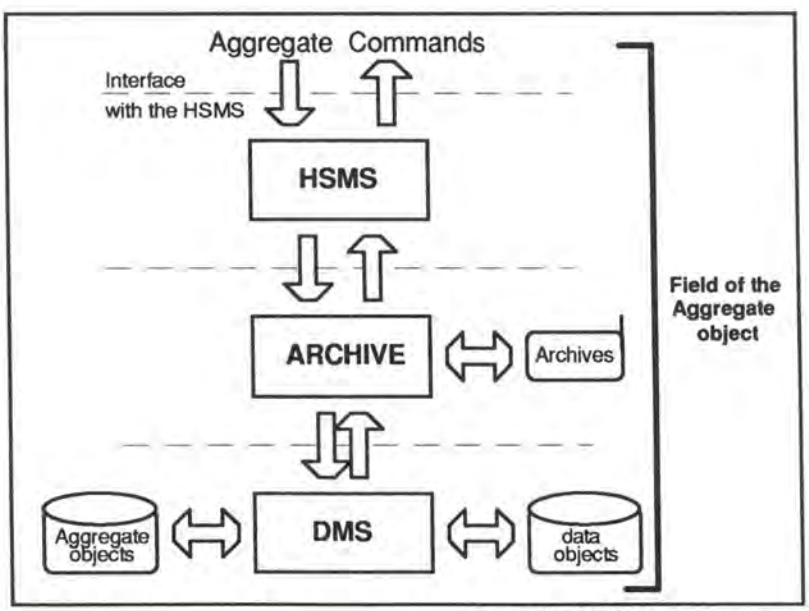


Fig 2.6. DMS Aggregates

Figure 2.6 shows that the DM system manages directly aggregates. The processing of aggregated data objects is transparent for any sub-system. The ARCHIVE kernel and the HSM system must know how to handle the aggregate as an object.

2.2.2.4. Summary

The best solution seems to include aggregate objects in the DM system.

In a stepwise approach, a first implementation could be limited to the HSM system. This will reduce external requirements and reduce costs. A first solution could be:

- Aggregates are managed by the HSM system;
- the ARCHIVE kernel knows a minimum about aggregates;
- Migration towards DMS-managed aggregates is foreseen.

We assume this solution for the following issues.

2.2.3. Aggregate Responsibilities

2.2.3.1. Overview

Security in the mainframe area is a major concern. In large systems, each user has a user-ID and a private workspace. Users are cut off from each other, and cannot peek in someone else's data objects. Beyond these users, some user-ID's share a set of privileges. These privileges give to the system administrators comprehensive access to the system.

The SRPM product (System Resources and Privileges Management) provides for privileges in the BS2000 environment. The privilege with regard to storage management is the HSMS-Administration privilege. The user-ID granted with this privilege is able to fully control the HSM system. In addition he is able to take full backups of the DM system. Therefore the HSMS-Administration has the rights to access all the objects of the Data Manager.

The aggregate concept has new requirements. Only the owner has the full knowledge to manage an aggregate. He best knows:

- which data objects should take part in the aggregate;
- which objects are critical for the aggregate and need special care;
- when and how often should the aggregate be backed up;
- which migration policy should be attributed to the aggregate;
- which archiving policy should be defined.

Therefore the administrator of an application (or of any set of related data objects) must be able to create an aggregate object for it, and then take part in its management. We can distinguish two kinds of aggregates:

- the aggregates created by an unprivileged user, to manage his private data objects;
- the aggregates created by an application administrator, to manage a set of related data objects saved under several user-ID's.

The security concern appears for the second kind of aggregates.

2.2.3.2. Requirements

We can summarize aggregates requirements as follow:

1. Users require the right to create aggregate objects.
2. Owners require the right to issue aggregate commands for their aggregate objects.
3. As aggregates should meet any application topology, an application administrator requires the right to access data objects saved under other user-IDs.
4. Security requires restricted access to objects saved by other users.

2.2.3.3. Considered solutions

Two solutions are considered here:

1) Restrict the use of aggregates to the Storage-Administrator.

The idea is to perform all the aggregate operations via the storage administrator.

When a user responsible for an application wants to make use of aggregates, he must contact the storage administrator. The user can give a description of what he wants, and the storage administrator can create the corresponding aggregate object. Aggregate operations are issued by the storage administrator, but under some kind of user supervision.

All the requirements are the responsibility of the storage administrator, notably requirements concerning the confidentiality of data objects.

Advantages:

- nothing new is needed;
- the storage administrator has great control over aggregate operations and over data object access. He can coordinate operations like backups, and optimize the tape sessions.

Disadvantages:

- the aggregate concept is poorly used. Users don't have direct access to the aggregate system, so they might not use it at all;
- it implies the availability of the storage administrator.

Pending problems:

None.

2) Create a new privilege.

The solution is to create a true Aggregate-Administration privilege under the privilege manager (SRPM in BS2000). The system administrator could give this privilege to selected users, giving them the rights to issue aggregate operations with aggregates including data objects saved under other user-IDs.

So the HSM system could have two levels of privilege:

1) HSMS-Administration;
2) HSMS-Aggregate-Administration.

When an ordinary user-ID is granted with the HSMS-Aggregate-Administration privilege by the system administrator, the following functions become available:

- executing normal HSMS statements, like backup or restore;
- processing objects of other users by means of HSMS statements.

Advantages:

- the aggregate concept is used at its best. An aggregate administrator granted with the privilege can fully take part in the management task.

Disadvantages:

- there is an external (SRPM) requirement.

Pending problems:

- requirement 4 is not met. Aggregate administrators would have uncontrolled access to all objects of the system.
  To partially solve this problem, we could add to the Aggregate-Administrator privilege a concept of "user group". An application administrator only has access to objects saved under user-ID's registered in his group. But requirement 4 is still not met. An aggregate administrator would have uncontrolled access to all the data objects of the users of his group.
  Another way to solve this problem is to involve the users (i.e. the data object owners) in the construction of the aggregate. This will be discussed later on.

2.2.3.4. Summary

The best solution is surely to create a new privilege under the privilege manager. The system administrator would be able to grant the Aggregate-Administrator privilege to some user-ID's. This privilege would give them the rights to manage aggregated sets of data objects beyond the user-ID concept. The user group concept can help to enhance data security.

2.2.4. Data objects and aggregates

2.2.4.1. Overview

We will discuss of the relation between data objects and aggregates. Let's consider the following figure:

Fig 2.7. World of Data Objects

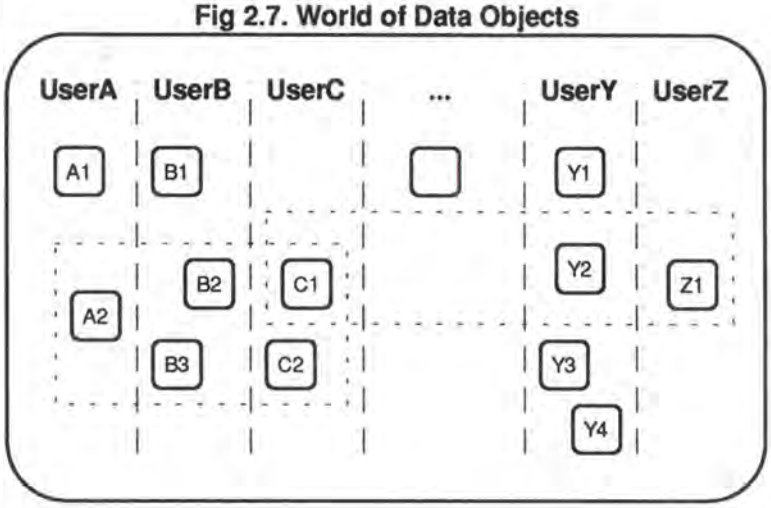


Figure 2.7 show a user-oriented model of the data object system. Each user owns a set of data objects (objects A1 and A2 for the user userA, objects B1, B2 and B3 for the user userB, etc.) Objects surrounded by dotted lines are related.

So issues to discuss are

- the connectivity between data objects and aggregates;
- the privacy of user objects;
- the method of assigning data objects to aggregates.

2.2.4.2. Connectivity

The question of the connectivity between data objects and aggregates is linked with the underlying relation between data objects.

In the application-level backup/recovery concept, an aggregate is created to collect under a single name all the objects belonging to an application. Backup and recovery are performed at once at the aggregate level, thus ensuring global consistency. In this idea, an object shared by several aggregates is a potential danger for each application. Consistency cannot be guaranteed for all the applications after the first recovery.

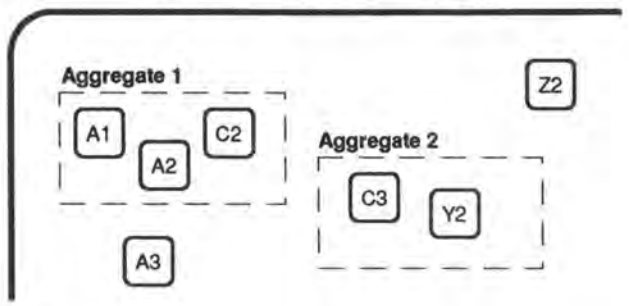
On the other hand, the relation between the objects may be of any kind. A general many-to-one connectivity may be too restrictive.

Three solutions are considered:

1) Enforce a many-to-one connectivity between data objects and aggregates.

A data object can be assigned to only one aggregate. The sets described by aggregate objects are separate (see figure 2.8).

Fig 2.8. World of Data Objects



Advantages:

- there is no danger of inconsistencies for shared data objects;
- it prevents user's mistakes;
- the connectivity is easy to manage.

Disadvantages:

- it is a very restrictive solution, because it cannot reflect the use of shared objects.

2) Allow a free connectivity, with no other control.

A data object may be freely assigned to several aggregates. The sets described by aggregate objects are not separate.

Fig 2.9. World of Data Objects

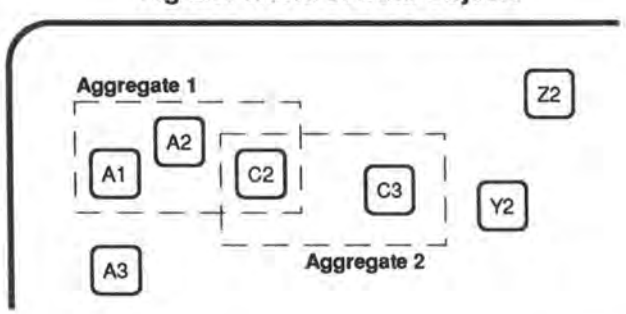


Figure 2.9 shows that data object C2 is shared by two aggregates. The aim is to ensure global consistency for each aggregate:

- the shared objects are read-only: there is no problem;
- the shared objects are Read/Write: there is a potential danger of inconsistency.

Advantages of this solution:

- aggregates may cover all the situations;
- there is no connectivity management problems, with no costs.

Disadvantages:

- the system offers poor support;
- there must be coordination among the aggregate administrators to avoid dangerous shared data objects;
- it can lead to serious inconsistencies.

3) Leave the connectivity to the owner's choice, with system control.

System control is based on two rules:

a) for each data object, the owner decides which aggregate(s) his object may belong to;
b) the owner of an aggregate may decide whether a data object belonging to his aggregate is authorized or not to become member of other aggregates.

Advantages:

- aggregates may cover all the situations;
- the system provides support for consistency control.

Disadvantages:

- The connectivity controls are complex. For example there must exist a way to ensure that a data object is assigned to only one aggregate (i.e. verify the many-to-one constraint).

2.2.4.3. Privacy

There is also a privacy aspect in the relations between data objects and aggregates. The privacy is related with the topic concerning the Aggregate-Administrator privilege.

The problem is to conciliate two opposite points:

a) give the rights to the aggregate administrators to access objects created under other user-ID's for aggregate operations;
b) ensure the privacy of user's data objects, whether the objects belong or not to an aggregate.

The privacy of a data object may be endangered when the aggregate administrator selects objects without the owner's permission. This can happen if a true privilege is used. Two cases may be distinguished:

1) the owner doesn't want his object to belong to aggregates (a many-to-none connectivity), but it is selected by an aggregate.
2) the owner wants his object to belong to one (or several) aggregate(s), but it is selected by another aggregate. This is a many-to-many, where "many" doesn't mean "any", but a "list of authorized" aggregates.

The privacy will be an important issue in the next section.

2.2.4.4. Assignments

The question is how to describe the set of data objects that belong to an aggregate. Recalling the aggregate object definition, this involves the link to the set of aggregated data objects (figure 2.10).

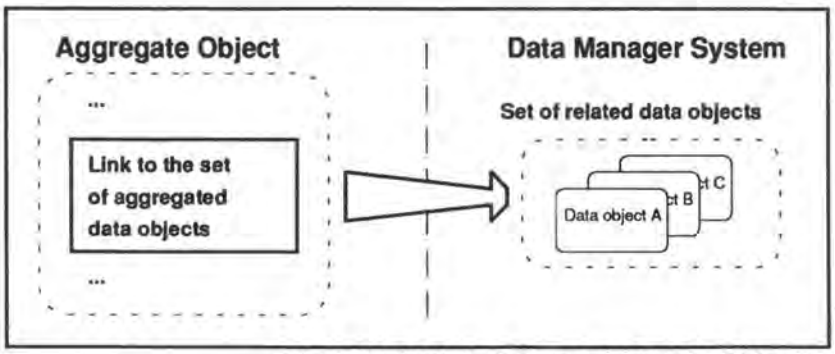


Fig 2.10. Link to Aggregated Data Objects

Important issues are:

- whether the aggregate object is a stand-alone object (self-sufficient) or depends of the availability of other metadata like DM catalogs. After a major disaster (loss of data objects and system catalogs), the HSM system must know for the recovery process which data objects belong to the aggregates.

- who is responsible for the setting up of the aggregated set of data objects (either the administrator of the aggregate, either the owners of the data objects). This issue is related with the control of connectivity between data objects and aggregates, and with the privacy of data objects.

- the update capabilities of the aggregates. Data objects which are members of an aggregate may still be directly processed at the data object level by commands from the DM system. Normal DMS operations may be carried out as usual, even on objects which are aggregate members. This is a problem because normal DMS operations may create inconsistencies between an aggregate object and the underlying data objects. Either the system has to update the aggregate objects, or someone has to do it.

Three solutions are considered for describing an aggregate set of objects:

Solution 1

- The link included in the aggregate object stores an exhaustive list of fully qualified names of data objects.
- The owner (administrator) of the aggregate writes this list.
- The users (data objects owners) don't do anything.

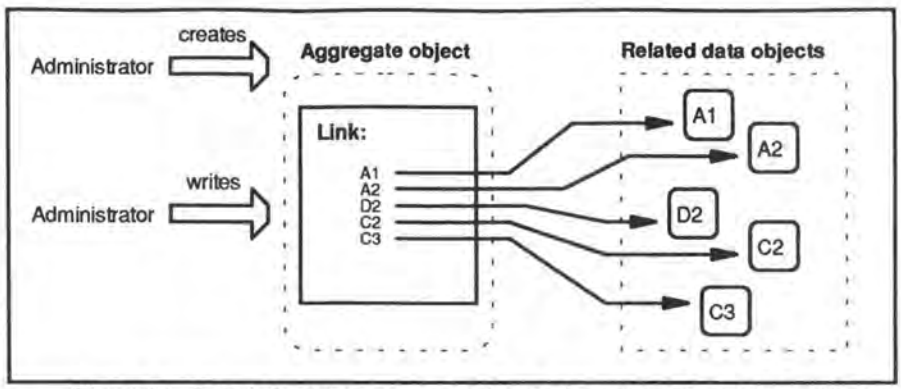


Fig 2.11. Administrator-managed list of aggregated data objects

Figure 2.11 illustrates the solution. When an aggregate command is issued by the administrator the system reads the list, and processes each data object. The system must verify that the administrator has the right to access the objects.

Advantages:

- the aggregate object stores a self-sufficient list of names. In case of loss of a device and the related catalogs, the aggregate object remains available for recovery.

Disadvantages:

- the list of names is quite static. An update is needed to add any data object to the aggregate;
- big aggregates are difficult and tedious to manage;
- a connectivity constraint would be difficult to control.

Pending problems:

- more means are needed to control a connectivity constraint;
- the privacy of user files is not guaranteed in the case of a privilege for aggregate administrators, because users are not involved in the selection of their data objects.

Solution 2

- The link included in the aggregate object is made of a sequence of Include/Exclude statements with full or partially qualified names.
- The application administrator writes this sequence.
- The users (file owners) don't do anything.

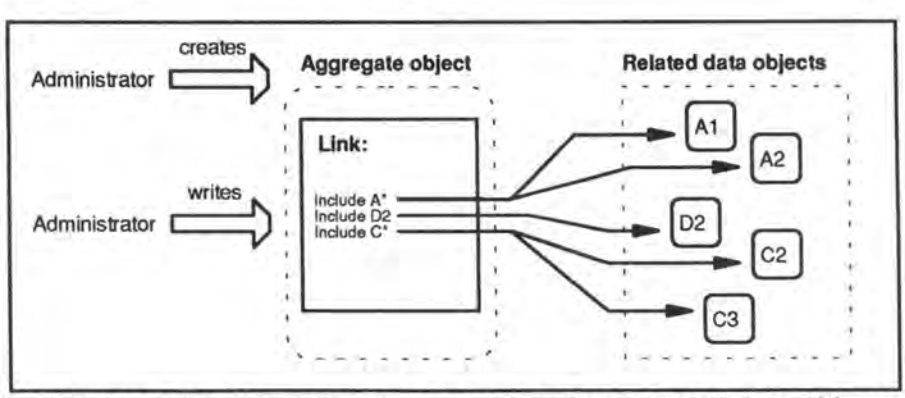


Fig 2.12. Administrator-managed partial list of aggregated data objects

Figure 2.12 illustrates the use of partially qualified names. When an aggregate command is issued by the administrator the system reads the sequence of Include/Exclude statements to build the exhaustive list of names in comparison with the current state of the catalogs, and processes each data object.

Advantages:

- the list of names is less static, because a good use of wild cards may automatically include or exclude new data objects. This is important if for example the application creates new data objects;
- it is less tedious to manage.

Disadvantages:

- the list is not self-sufficient, because it is dependent of the availability of the DM catalogs.

Pending problems:

- the list is not self-sufficient. Other means have to be provided to know which data objects have to be recovered after a disk crash;
- more means are needed to control a connectivity constraint;
- the privacy of user data objects is not guaranteed in case of an Aggregate-Administration privilege, because users are not involved in the selection of their data objects.

Solution 3

- The link included in the aggregate object is made of a list of partially or fully qualified names of data objects.
- The aggregate administrator creates the aggregate.
- The users (file owners) have special commands to add or remove data object names from the aggregate object link.

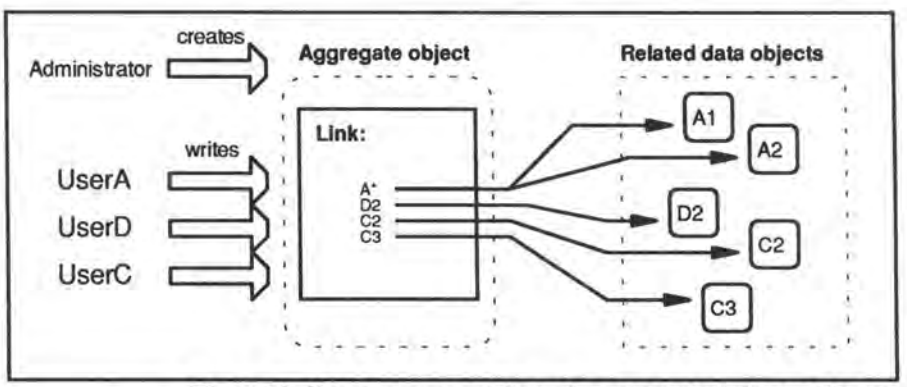


Fig 2.13. User-managed list of aggregated data objects

Figure 2.13 illustrates this solution. When an aggregate command is issued by the administrator, the system reads the list, builds the exhaustive list of names by comparing with the current state of the catalogs, and processes each data object.

Advantages:

- the list of data object names is very dynamic, because it is updated by the users themselves;
- it is very easy to manage by the application administrator;
- the privacy problem is partly solved, because the users are involved in the selection of the their data objects.

Disadvantages:

- the administrator has less control over the aggregate. He is not directly advised of modifications of the aggregate object;
- new commands must be introduced for the users.

Pending problems:

- the list is not self-sufficient. Other means have to be provided to know which data objects have to be recovered after a disk crash;
- more means are needed to control a connectivity constraint.

2.2.4.5. Summary

We think that a free (many-to-many) relation between data objects and aggregates is a good solution. The aggregate administrators are responsible for a good use of aggregates. Some diagnostic tools could be provided to check if data objects are selected by several aggregates.

For the description of the set of data objects, we propose that the link of the aggregate object includes a sequence of Include/Exclude statements (solution 2), because it is a dynamic solution. The self-sufficiency of the aggregate has to be reached by other means.

2.2.5. Aggregates and archives

2.2.5.1. Overview

This section deals with the relations between aggregate objects and the archive files of the HSM system. This topic is dependent of the BS2000 environment, so we will first introduce a simplified and generalized archive concept.

Our general archives are the basic HSM background storage units. A copy of a data object (produced by a backup, archiving or migration function) is stored in an archive. Figure 2.14 gives a summary of the general archive structure.

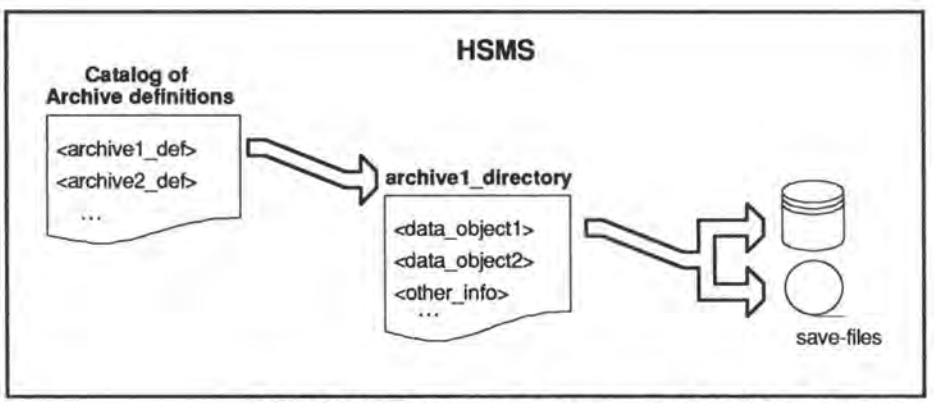


Fig 2.14. Summary of our general archive structure

An archive consists of:

- an archive definition, internally stored by the HSM system in the catalog for archive definitions;
- an associated archive directory, managed by the HSM system;
- a set of files called save-files which contain the copies of the saved data objects. A save-file contains the data objects saved during one backup or archiving session.

An archive definition entry <archive_def> stores the name of the archive, its type (for backup, for archiving or for migration), its owner, and the name of the associated archive directory.

An entry <data_object> in the archive directory stores the name of the data object, and a list of names identifying the save-files which contain a copy of the object. Other types of entries are possible, and are summarized by the <other_info> entry. A new save-file is created for each backup or archival session. A data object can be saved only once in a save-file.

The HSMS administrator can create public archives for backup, archiving and migration. Besides, any user can create his own private archives for backup or archiving. Private migration archives are not allowed. A private archive is restricted to the owner.

Let's also recall that aggregates are HSMS-managed objects, and that a maximum of compatibility must be ensured between aggregate-level storage management and data object-level storage management (requirement B2).

Therefore issues to discuss are

- the connectivity between aggregates and archives;
- the coexistence of aggregate objects and data objects in save-files.

2.2.5.2. Connectivity

The issue here is the connectivity between aggregates and archives. Note that an aggregate backup copy should never be split among several archives. Several connectivities are possible:

1) Enforce a mandatory one-to-one relation. Each aggregate has its own unique and dedicated archive. Conceptually speaking, this is the simplest relation.

Advantages:

- it complies with application reconfigurations and disaster recoveries, because the archive may be moved around with the application. Nothing is mixed up;
- a one-to-one constraint is easy to manage.

Disadvantages:

- it is very restrictive and will not suit many situations;
- it increases the number of archives.

2) Enforce a mandatory one-to-many relation. Each aggregate has its own set of dedicated archives. This relation may be justified if an aggregate is backed up for several distinct purposes, each archive being private and assigned to a different purpose. Remind that access to a private archive is restricted to the owner.

Advantages:

- it complies with application reconfigurations and disaster recoveries, because the archives may be moved around with the application;
- a one-to-many constraint is not too difficult to manage.

Disadvantages:

- it is still very restrictive;
- it increases the number of archives.

3) Enforce a mandatory many-to-one relation. Several aggregates share a unique archive. This relation is conceivable if one user is responsible for several aggregates and doesn't want to manage several archives, or if the archive is a unique public archive.

Advantages:

- less archives are needed.

Disadvantages:

- shared archives make application reconfigurations and disaster recoveries quite difficult;
- it is still restrictive.

4) Allow a free (many-to-many) connectivity. Several aggregates share several archives. This fourth relation may be justified by any combination of the above. Remind that access to a private archive is restricted to the owner.

Advantages:
- it can suit any situation. It is the user's responsibility to manage correctly his archives;
- less archives are needed;
- no special management is required from the system.

Disadvantages:
- there are no means for a specific control;
- shared archives make application reconfigurations and disaster recoveries quite difficult.

5) Allow the user to choose a system-managed connectivity. The owner of an archive can decide which aggregates are allowed to use it. The owner of an aggregate can decide which archives his aggregate can use. Both permissions have to be verified prior to any operation.

Advantages:
- it can suit any situation;
- complete access control may be done, so aggregate dedicated archives may be ensured when needed.

Disadvantages:
- it implies complex connectivity management, with high costs.

2.2.5.3. Archive sharing

We have to talk about the coexistence of aggregate objects and data objects in a same archive. From the HSMS point of view, a save-file contains a sequence of saved objects. Beside the fact that the objects were saved within the same backup or archiving session, there is no special relation between the objects. They are all independently accessible. An aggregate being a collection of data objects, their status inside a save-file has to be discussed.

General questions are:

(1) whether aggregated data objects and independent data objects may coexist in a same save-file or not;

(2) if they may coexist, how to later recognize aggregated data objects from independent data objects within the save-file, for aggregate restores.

Let's recall that the HSM system performs aggregate operations by calling several times data object level functions of the ARCHIVE kernel. The ARCHIVE product is not aware of aggregates. The HSM system has to organize the way aggregated data objects are saved, to be able to restore them later as aggregates.

Let's consider a set of data objects as in figure 2.15. Data objects A1, A2, A3 and B1 are related by Aggregate1, data objects B1, B2 and B3 are related by Aggregate2. C1, C2 and C3 are independent data objects.

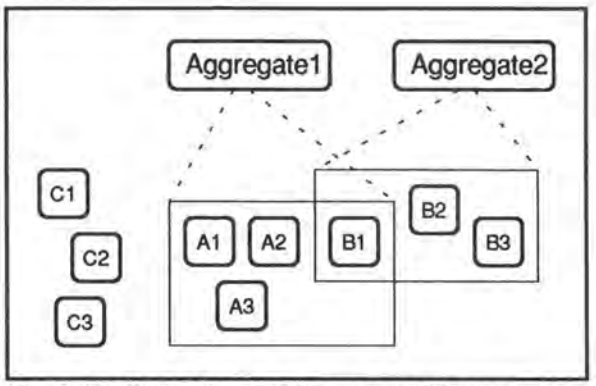


Fig 2.15. Set of Data Objects and Aggregates

Aggregates being a set of data objects, the sharing of the archive can be done at the save-file level or within the save-file.

1) Sharing the archive at the save-file level.

The save-file is a unit for archive sharing. A save-file can contain either
- a set of individual data objects; or
- one aggregated set of data objects.

For the example of figure 2.15, the administrator would have to start three backup sessions to backup Aggregate1, Aggregate2 and independent objects C1, C2 and C3. The resulting archive is shown in figure 2.16. It contains three new save-files.

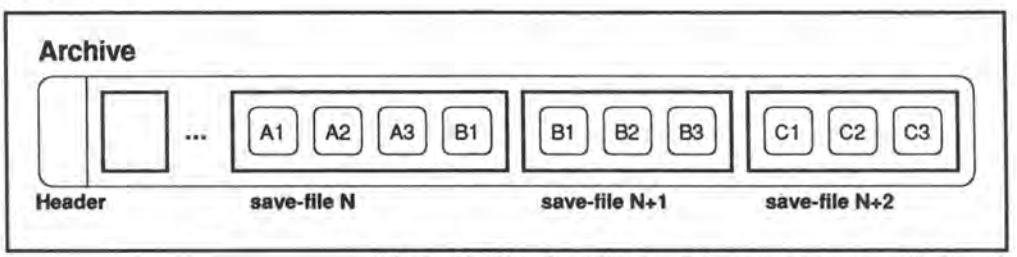


Fig 2.16. Sharing the archive at the save-file level

Advantages:
- as the save-file is a unit for archive sharing, the distinction between data objects saved as members of an aggregate and independent objects is done at the save-file level.

Disadvantages:
- aggregates have to be saved in separate backup sessions, because they need separate save-files;
- shared data objects (like object B1) is saved several times.

Aggregate restore requirement:

- the HSM system has to be able to find the save-files of the archive containing the aggregate. An aggregate flag is required at the save-file level;
- if the save-file stores an aggregated set of data objects, all the objects have to be restored. Therefore no special flagging is needed at the object level.

2) Sharing the archive within the save-file.

A save-file may contain individual and aggregated data objects. The data object is a unit for archive sharing. For the example of figure 2.15, the HSM system can save all the objects in a single save-file (see figure 2.17).

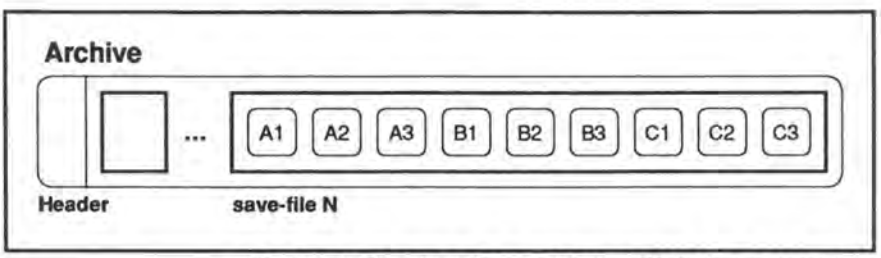


Fig 2.17. Sharing the archive within the save-file

Advantages:

- individual and aggregated data objects may be saved at once. There is no need for separate backup sessions;
- shared data objects can be saved only once.

Disadvantages:

- the distinction between objects saved as members of aggregates and other objects has to be done at the object level. Data objects must be flagged.

Aggregate restore requirements:

- the HSM system has to be able to find the save-file of the archive containing the aggregate. An aggregate flag is required at the save-file level;
- the HSM system has to know which data objects of the save-file have to be restored as aggregates. Aggregate flags are also required at the data object level.

2.2.5.4. Summary

We think that a free connectivity between archives and aggregates is a good solution. If a dedicated archive is needed, it should be created private to restrict access to one responsible user.

To meet the compatibility requirement (B2), archives are sharable within the save-files. A save-file may contain data objects and aggregated data objects. Any set of objects can be backed up in a single backup session.

Special metadata is needed to flag data objects saved as aggregates.

2.3. The aggregate object in the SMS-Pool

2.3.1. Overview

This chapter presents an approach to integrate the aggregate object in the SMS-Pool topology. Figure 2.18 recalls the definition of the aggregate object.

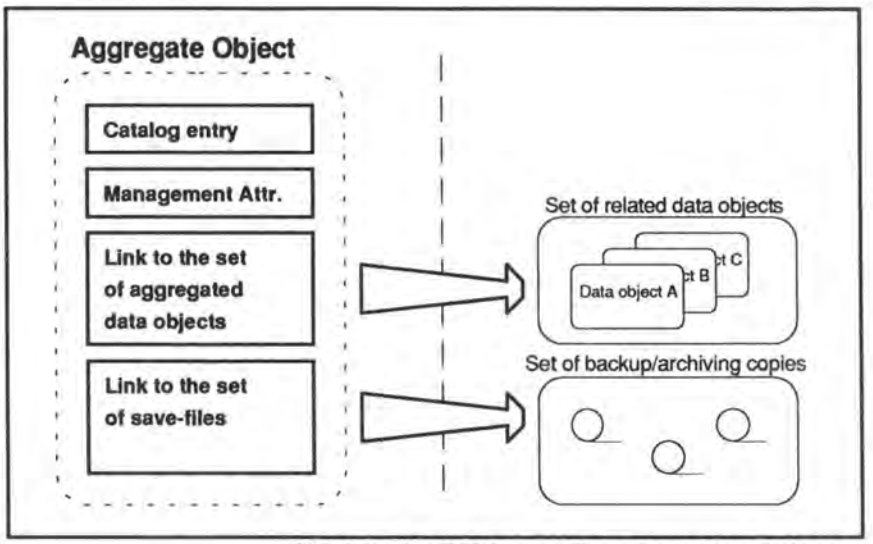


Fig 2.18 Definition of the aggregate object

The HSM system already manages a vast amount of metadata for other management functions. The four components (catalog entry, management attributes, link to the set of aggregated data objects and link to the set of save-files) of the aggregate object have to be integrated in the HSMS metadata and in the SMS-Pool. In addition, aggregate operations have to be defined.

2.3.2. Cataloging aggregates

The HSM system must keep track of all the aggregate objects that are defined within the SMS-Pool. Therefore the HSM system must manage a catalog.

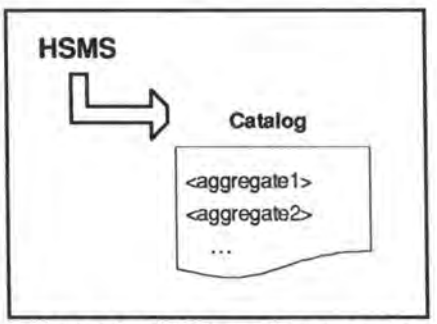


Fig 2.19. Catalog of aggregates

Fig 2.19 shows a general catalog of aggregates.

A catalog entry contains

- a field to store the name of the aggregate;
- a field to store the user-ID of the owner;
- a field to store a set of attributes (date of creation ...).

The catalog must be local to a SMS-Pool (requirement B3). It requires to be stored on a high reliability device, because it is important metadata.

The HSM system already manages a catalog for archive definitions (see the section 2.2.5 Aggregates and Archives) which meets these two requirements. Therefore two solutions are reviewed:

1) Extend the current catalog of archive definitions.

A new type of entry is defined to catalog aggregate objects. Figure 2.20 illustrates this solution.

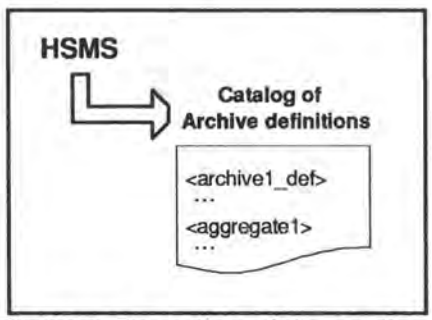


Fig 2.20. Extension of the catalog of Archive definitions

Advantage:

- there is no need to create and manage a new catalog.

Disadvantage:

- the aggregate object is very linked to the HSM system. This is not good for future developments like a migration to DMS-managed aggregates;
- the catalog for archive definitions is misused.

2) Create a new catalog for aggregate objects.

A new catalog is managed for aggregates by the HSM system. Figure 2.21 illustrates this solution.

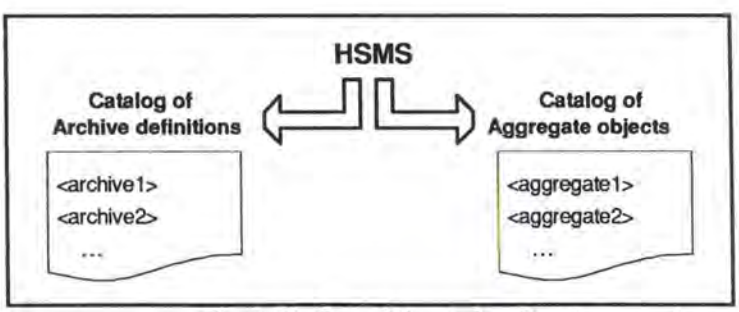


Fig 2.21. Special catalog for Aggregates

Advantage:

- the aggregate catalog is independent of other HSMS metadata. An upgrade to DMS-managed aggregates would be easier.

Disadvantage:

- the HSM system must manage a new catalog.

At this point, an independent catalog for aggregates is preferable. The aggregate metadata are not mixed up with other metadata.

2.3.3. Availability management for aggregates

The main motivation for introducing the aggregate concept is to enhance the space and availability management functions for related data objects. Questions are

1) which management attributes are needed for aggregates;
2) whether the management class concept is applicable to aggregates or not.

Space and availability functions are migration, backup and archiving. Automated backup and archiving of aggregates are very interesting, whereas automated migration has no sense because the migration criterion (i.e. inactivity delay) only works for data objects. Manual migration is useful to send a whole application to a background level.

Interesting backup and archiving attributes are the same as those for data objects.

Backup attributes:

- automated backup of the aggregate: enable / disable switch;
- backup frequency: controls the selection for backup;
- guaranteed backup: forces a backup before manual migration;
- number of backups retained while the aggregate object exists;
- number of backups retained while the aggregate object is deleted;
- retention period of a backup.

Archiving attributes:

- automated archiving of the aggregate: enable / disable switch;
- expiration date or expiration period: date of archiving;
- expiration period after archiving;
- maximum retention period.

The management class concept was introduced in section 1.2.3.2. A management class is a pre-defined management policy within a SMS-Pool. It enables the storage administrator to control the management policies allocated to data objects.

An extended management class concept is useful for aggregates. The differences are:

- The migration attributes are not needed for aggregates.
- Only the administrator of an aggregate really knows the management requirements. Therefore the aggregate administrators should be able to create management classes for aggregates.
- Aggregates are likely to be backed up towards private archives. The management class has to mention which archive(s) should be used.

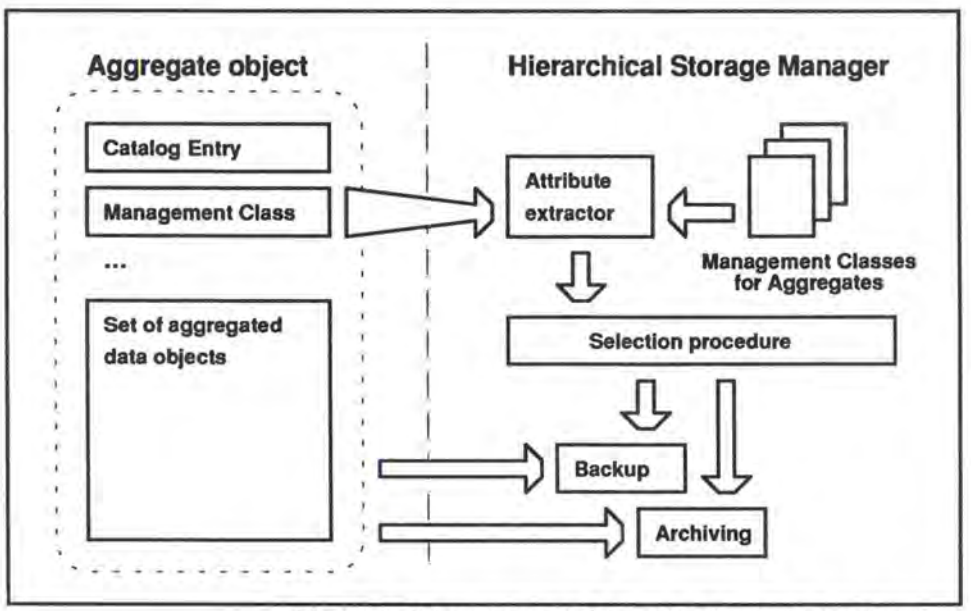


Fig 2.22. Management Class for Aggregate objects

The figure 2.22 shows a system-managed aggregate object. The management class for aggregates describes its requirements in terms of backup and archiving.

For each aggregate object, the name of the allocated management class can be stored in its catalog entry. So the catalog entries are extended, and contain:

- a field to store the name of the aggregate;
- a field to store the user-ID of the owner;
- a field to store a set of attributes (date of creation ...);
- a field to store the name of the management class for aggregates.

2.3.4. Linking data objects to an aggregate

The link to the aggregated data objects describes a collection of data objects. The question of how to describe the collection is discussed in section 2.2.4.4 (Assignments). Several solutions are reviewed. The proposed solution is to describe the set of data objects with a list of include and exclude statements with full or partially qualified names, because it defines a dynamic collection of objects.

Now the problem is how to store this information. Storage requirements are:

- the information must be stored in the SMS-Pool of the aggregate;
- there is no specific availability requirement. The link information is not critical, because it can be saved in the backup session.

Two solutions are considered:

1) Extend the entries of the catalog of aggregates.

Considering the catalog of aggregates, the catalog entries are extended, and contain:

- the name of the aggregate;
- the user-ID of the owner;
- a set of attributes (date of creation ...);
- the name of the management class for aggregates;
- the list of Include and Exclude statements with full or partially qualified data object names.

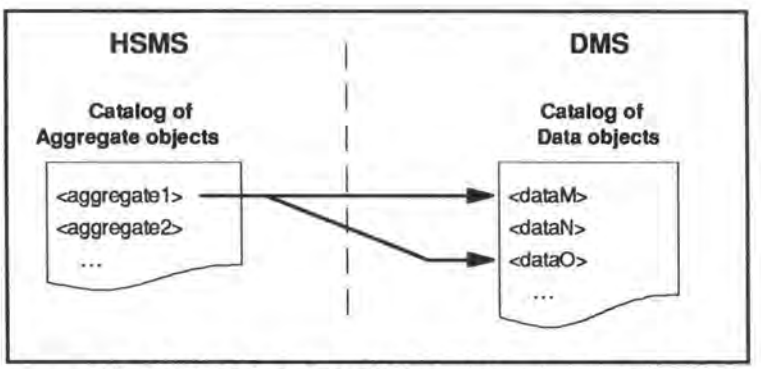


Fig 2.23. Extension of the catalog to store the link

Figure 2.23 illustrates the extension of the catalog entries. An aggregate catalog entry contains the description of the set of aggregated data objects.

Advantages:

- the HSM system manages all aggregate metadata in a single catalog.

Disadvantages:

- the catalog of aggregates is likely to grow very fast;
- the catalog of aggregates is located on a highly reliable device. The link doesn't require this service level.

2) Use a file of the DM system.

An easy solution is to use the concepts already managed by the system. The information can be stored in a data object (a sequential file), and cataloged in the Data Manager catalog under the same name as the aggregate name.

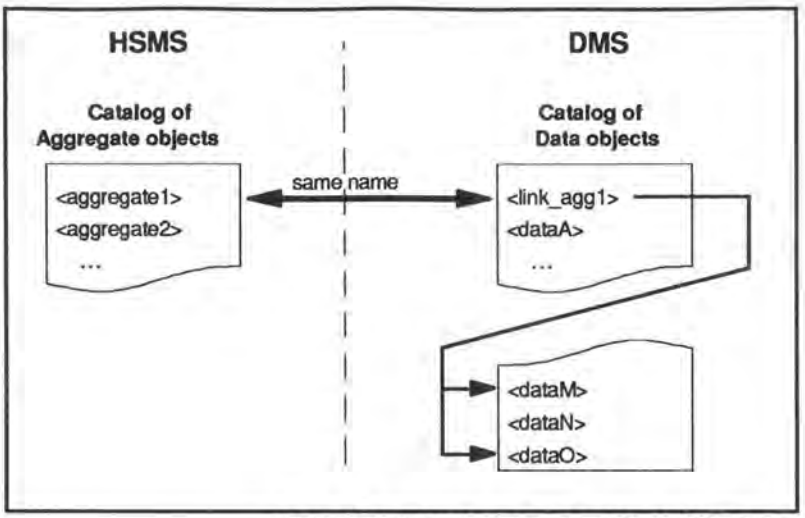


Fig 2.24. Link via an associated DMS file

Figure 2.24 illustrates this solution. Each aggregate has an associated DMS file. The aggregate and the file have the same name. The link is indirectly available through the DMS file.

Advantage:

- the catalog of aggregates is not extended;
- the file storing the link can be allocated on a more suitable device.

Disadvantage:

- the aggregate metadata is indirectly available.

At this point, the link stored in an associated file has more advantages. The catalog of aggregates is not overloaded, and a sequential file is easy to manage.

2.3.5. Linking save-files to an aggregate

The backup and archiving copies (i.e., the save-files) of an aggregate must be related to the aggregate.

For data objects, this is done via the archive directories. For each archive, the associated archive directory contains the list of names of data objects which are saved in the archive. An entry in the directory stores the following fields:

- name of data object;
- list of [save-file-ID].

For a given data object, the archive directory gives the list of save-files that contain a copy of the object. This information is on-line.

Previous topics imply new requirements for aggregates:

- Archives are sharable, and save-files can contain a collection of individual and aggregated data objects. The problem is to recognize aggregated data objects from all the objects saved in a save-file. Aggregate objects only include a list of partially qualified names of data objects. The link to the set of aggregated data objects is not sufficient to recognize the objects.
- The aggregate object could be unknown in the target SMS-Pool, either because of deletion or because of a disaster recovery on a new system. In addition, an aggregate object is very likely to be modified as time goes by. Therefore
  - the archive directory must be able to store several versions of the same aggregate;
  - enough components of the aggregate object must be included in a backup or archiving copy, to allow disaster recovery. The link file must be included in the copy.

Proposed solution:

A new type of entry is introduced in the archive directory. The content of the entry is:

- A-type : type of entry (Aggregate);
- <aggregate-ID> : owner and name of the aggregate;
- list of [save-file-ID; list of [data object name]].

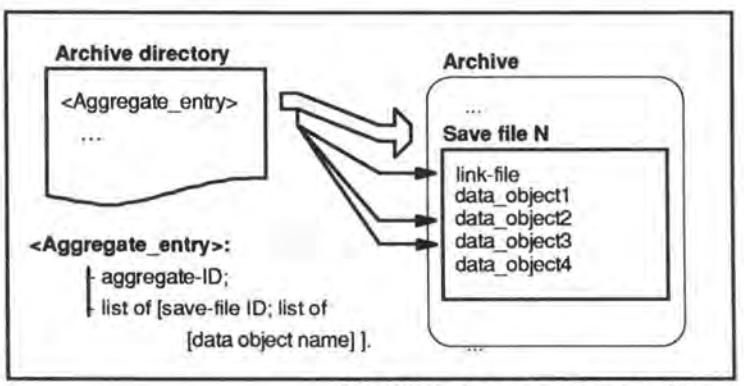


Fig 2.25. Aggregate metadata

Figure 2.25 illustrates the solution. For a given aggregate object, the archive directory stores the list of save-files that contain a copy of the aggregate. With each save-file, it also gives the list of data objects saved under the aggregate. To comply with the disaster recovery requirement, the associated DMS file storing the link is added to the save-file.

2.3.6. Aggregate operations

Two types of aggregate operations have to be defined: aggregate management functions and aggregate commands.

1) Aggregate management functions

Aggregate management functions deal with the processing of the aggregate object itself, without the aggregated data objects. Operations include:

- Create an aggregate object: the user must supply
  - the name of the new object;
  - the name of a management class;
  - a sequence of include and exclude statements to describe the set of aggregated data objects.

  This operation creates a new entry in the catalog of aggregates and a new DMS file, both stored under the same name. The user's user-ID, the aggregate name and the management class name are stored in the catalog entry. The sequence of statements is stored in the DMS file.

- Modify an aggregate: the owner can modify any component of the aggregate.

- Delete an aggregate: the owner can delete an aggregate. This deletes:
  - the entry in the catalog of aggregates;
  - the associated DMS file.

  This operation doesn't delete the aggregated data objects.

- Test an aggregate: the owner can test an aggregate to ensure that all the related data objects are really included in the aggregate.

2) Aggregate commands

Aggregate commands deal with the processing of the aggregated data objects. Operations are supported in the frame of the HSM system, namely:

- Backup of an aggregate: the owner can issue an aggregate backup command, or the system can start an automated backup task (defined by the management class). The backup task receives as parameters the name of the aggregate and the name of the archive, and performs the following operations:

  (1) read the aggregate's entry of the catalog of aggregates;
  (2) verify that the owner of the aggregate is the user who issued the command; if not, stop the backup task.
  (3) read the link file of the aggregate;
  (4) build the complete list of data objects, by consulting the DMS catalogs and the link statements;

(5) add the associated link file to the list;
(6) test if the aggregate has an entry in the archive directory. If not, create one;
(7) create a new save-file, and add the save-file-ID to the archive directory entry of the aggregate;
(8) For each data object on the list, do
    (1) if the user-ID has no Aggregate-Administration privilege, verify that the user is the owner of the data object. If not, skip this object;
    (2) perform a normal backup of the data object;
    (3) add the name of the objects to the archive directory entry.

The normal backup of data objects (operation 8.2) ensure that they can be restored individually at the data object level (compatibility requirement). Backups made for disaster recoveries must include a copy of the archive directory. The directory is needed for the restore process.

- Restore of an aggregate: the owner can issue an aggregate restore command. The restore task receives as parameters the name of the aggregate, the name of the archive and the save-file-ID, and performs the following operations:

    (1) open the archive directory entry of the aggregate, and read until the requested save-file-ID is reached;
    (2) continue reading the archive directory entry, and for each name, do
        (1) if the user-ID has no Aggregate-Administration privilege, verify that the user is the owner of the data object. If not, skip this object;
        (2) restore the data object;
    (3) if the aggregate is not cataloged in the catalog of aggregates, create an entry with the user-ID of the user who issued the command, the name of the aggregate, and a default management class.

    Note that the link file is automatically restored with the aggregated data objects. In case of disaster recovery, the archive directory must be restored before the aggregate restore process can take place.

- Archiving of an aggregate: the owner can issue an aggregate archiving command, or the system can start an automated archiving task (defined by the management class). The archiving task performs the same operations than the backup task, except that the data objects are archived.

- Migration of an aggregate: the owner can issue an aggregate migration command. The migration task performs the same operations than the backup task, except that the data objects are migrated to a background level.

- Recall of an aggregate: the owner can issue an aggregate recall command. The recall task moves all the migrated data objects to the processing level of the HSM system.

- Show archive contents: the owner can issue a query to display archive informations over the aggregate, like the save-file-ID's or the list of aggregated data object, etc.

2.4. Conclusion

The system-managed storage concepts introduced in Part One don't guarantee the consistency of related data objects. Data objects are viewed and managed as independent objects.

The aggregate concept aims to collect under a single name any set of semantically related data objects and their associated management metadata.

The aggregate object is an implementation of the aggregate concept. It is a new object managed by the HSM system, and has four components:

- an entry in the catalog of aggregates, which is managed by the HSM system;
- a management class;
- a link to the set of aggregated data objects;
- a link to the set of associated save-files.

Two classes of operations belong to the aggregate object:

- aggregate management functions involve the processing of the aggregate object itself, like create, modify, delete an aggregate;
- aggregate commands involve the processing of the set of data objects that are collected under the aggregate, like backup, restore, archiving of an aggregate.

The aggregate object as integrated in chapter 2.3 complies with the user-related and HSMS-related requirements:

A1. Any user must be able to create aggregate objects. The owner of an aggregate must be able to modify or delete it.

The creation function is available to any user. The owner-ID is stored in the catalog entry of the aggregate. The HSM system can verify if a user is the owner or not.

A2. The user who created an aggregate object must be the owner of the object and its administrator.

The owner can allocate a management class to the aggregate.

A3. The aggregate object must be able to meet any application topology.

The Aggregate-Administration privilege gives to selected users the rights to access data objects saved under other user-ID's. These users are called Aggregate Administrators.

A4. The aggregate object must ensure the confidentiality of the aggregated data objects.

The Aggregate-Administration privilege is only granted to selected user-ID's. Ordinary users can't have access to data objects of other users.

A5. The aggregate object must offer efficient availability functions, and provide for disaster recoveries.

The owner can allocate a management class to the aggregate. A new entry type is introduced for aggregates in the archive directories, and the save-files are able to store aggregated data objects. Enough metadata can be included in the save-files to provide for a disaster recovery.

B1. The aggregate concept must comply with the HSM concepts.

The aggregate concept doesn't enforce any restriction on the HSM concepts.

B2. A maximum of compatibility and flexibility must be ensured between aggregate-level management and the traditional management at the level of the data objects.

Aggregate-level management implies a traditional management of all the aggregated data objects. Additional metadata is included to be able to restore the data objects as one aggregate. For example, the backup process of an aggregate implies the normal backup of all the aggregated data objects, includes the backup of the link file, and updates the aggregate entry in the archive directory. A backup session can include aggregate objects and individual data objects.

B3. The aggregate must fit in the SMS-Pool topology.

The aggregate object and its related metadata are local to a SMS-Pool.

B4. Aggregates must be safe against device crashes, to ensure recovery.

The catalog of aggregates is stored on a high availability device.

The aggregate object provides system-managed and aggregate-level storage management functions.

Future developments:

The aggregate concept could be introduced in the Data Management System. First, the DM system could provide aggregate commands like copy of an aggregate or delete of an aggregate. Then any other sub-system could provide aggregate commands.

The aggregate concept could be very useful in distributed environments. Trends for distributed applications make consistent backups quite difficult. A first step solution consists in the backup server concept: a machine acting as a server is able to perform backup and restore functions for all the LAN-based workstations. An extended aggregate concept could collect under a single name a set of related data objects residing on several distributed workstations, and offer aggregate-consistent backup and restore functions.

Bibliography

(1) Siemens Nixdorf Informationssysteme, Hierarchical Storage Management System, internal documents of Siemens Nixdorf Software.

(2) Siemens Nixdorf Informationssysteme, Data Manager System, internal documents of Siemens Nixdorf Software.

(3) Siemens Nixdorf Informationssysteme, System Resources and Privilege Management, internal documents of Siemens Nixdorf Software.

(4) "Getting to Grips with Archives", Siemens magazine COM, June 1989.

(5) IBM, MVS/DFP V3 R3 General Information, GC26-4552-3.

(6) J.P. Gelb, "System-managed Storage", IBM Systems Journal, 1989 Vol 28, No 1, pp 77-103, G321-5349.

(7) W.B. Harding, C.M. Clark, C.L. Gallo, H. Tang,
"Object Storage Hierarchy Management", IBM Systems Journal, Vol 29, No 3, 1990, pp 384-397, G321-5407.

(8) R.C. Alford, "Disk Arrays Explained", BYTE magazine, October 1992, pp 259-266.

(9) J.P. Cardinael, course on "Gestion des Ressources Informatiques", FUNDP Namur, 1993.

(10) Richard E. Matick, "Computer Storage Systems and Technology", Wiley-Interscience publication, New York (USA), 1977, ISBN 0-471-57629-8.

Appendix A :

The BS2000 environment

This appendix presents an overview of the SNI BS2000 mainframe environment. This overview is not an exhaustive description of the BS2000 system. It only introduces the concepts that could help the reader to understand the issues of this paper.

BS2000 is a large operating system that includes a set of mandatory systems and a set of optional sub-systems. Two systems are concerned by this study: the Data Management System and the Hierarchical Storage Management System.

Data Management System (DMS)

The Data Management System is a mandatory system. It deals with the task of storing, organizing, identifying and retrieving data objects. Data objects are mainly files and job variables. Data objects are listed in DMS-Catalogs. The storage space is implemented by disk storage organized in pubsets. A pubset groups a set of disks.

Hierarchical Storage Management System (HSMS)

The HSMS product is an optional system. It provides space and availability management functions in a BS2000 system. The HSMS relies on a set of management functions provided by the ARCHIVE product. The main concepts are reviewed.

Logical storage levels:

The HSMS supports the definition of three logical storage levels:

- storage level 0 (S0) is the normal processing level. It is implemented by fast, on-line pubsets. S0 is managed directly by the Data Management System.
- storage level 1 (S1) is the first background level. It is implemented by on-line pubsets. Data stored on storage level 1 is managed by the HSMS.
- storage level 2 (S2) is the second background level. It consists of magnetic tapes or magnetic tape cartridges. Data stored on storage level 2 is managed by the HSMS.

HSMS archive

The archive is the basic HSMS management unit. HSMS stores and manages all data saved by either backup, archiving or migration in archives. An archive has a type and is dedicated to one of the three functions.

Each HSMS archive consists of

- the archive definition: stored in the control file, contains
  - the owner of the archive;
  - the type;
  - the name of the associated archive directory;
  - other attributes.
- an associated archive directory: contains informations about the data managed in the archive, like:
  - names of data objects;
  - save-file-ID's;
  - save-versions;
- a set of save-files: A save-file is a cataloged DMS file, and contains the saved data. A save-file contains one or more save-versions. A save-version contains all the data and related metadata saved by one request. A data object can be saved only once in a save-version.

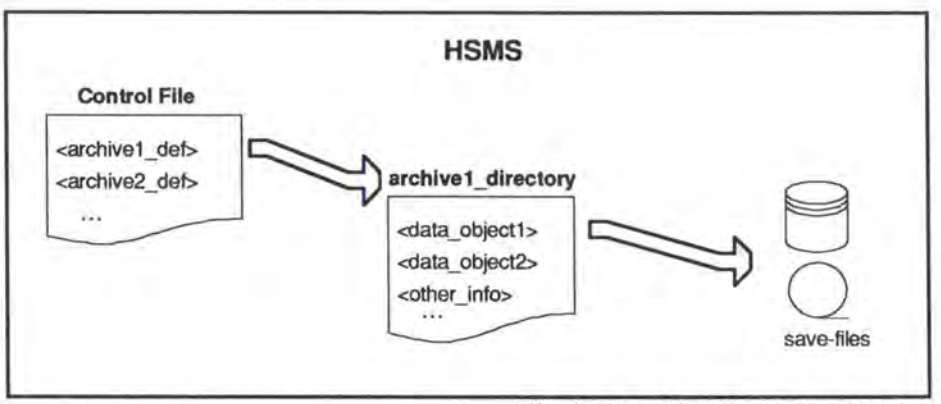


Fig A.1 Archive Data Structure

Figure A.1 illustrates the archive data structure. Figure A.2 illustrates the structure of a save-file and save-versions.

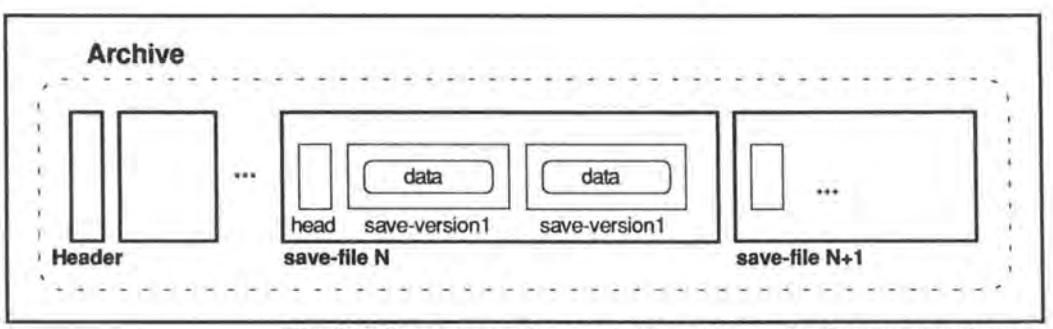


Fig A.2 Structure of a save-file with several save-versions

In addition, the HSMS provides default system archives for each of the three functions.

Functions

The HSMS supports migration, backup, archiving within a BS2000 system, and data transfer between different BS2000 installations.

Migration is the process of moving inactive data from the processing level to a background level. The migrated data is moved to a migration archive and the storage space it occupied at the processing level is released. However, the catalog entries of migrated files are retained at the processing level. When an attempt to read a migrated file is made by the DMS, the file is automatically recalled to the processing level.

Backup is the task of making or updating copies of the data inventory. In case of failure, the data can be recovered from the latest backup copy. The HSMS supports full backup, incremental backup and partial backup.

Archival is the long-term saving of files that are no longer required to cost-effective tape cartridges at level S2 of the storage hierarchy.

Data transfer allows to copy data objects on magnetic support and to read them on another BS2000 installation.

System Privilege

The System Resource and Privilege Management (SRPM) provides a special HSMS privilege: the HSMS-Administration privilege. The users granted with this privilege have the following rights:

- They have complete access to the HSM system. They can tune all the HSMS parameters.
- They have comprehensive access to all the data objects stored by the DMS. This allows them to run complete system backups.

Ordinary users have restricted access to the HSMS. For example, they can issue a backup command over their own data objects.

Appendix B :

Example of a HSMS-BS2000 installation

Computer installations and storage capacities are confidential data. However, thanks to their kind permission, here are some figures from the computer center of SNI Namur.

The BS2000 environment includes the HSMS product. Only two logical storage levels are defined:

- storage level 0: the processing level implements about 19 Gigabytes of fast on-line disk devices.
- storage level 2: the background level implements an automated library of magnetic cartridges (robot tower). Each cartridge has a capacity of about 800 Megabytes.

Storage level 1 is not defined. More interesting are the backup policies:

- every day: incremental backup of all the modified files
  - retention period: 14 days kept in the robot tower;
  - time needed: 1h00;
  - tapes needed: 10 cartridges.

- every week: full backup of all files
  - retention period: 14 days kept in the fireproof cellar;
  - time needed: 2h30;
  - tapes needed: 35 cartridges.

- every 2 weeks: full backup of all files
  - retention period: 150 days kept in the robot tower;
  - time needed: 2h30;
  - tapes needed: 35 cartridges.

- every 2 weeks: physical save of all the disks;
  - retention period: 30 days kept in the fireproof cellar;
  - time needed: 2h30;
  - tapes needed: 45 cartridges.

The need for the migration function is not high. Therefore migration is poorly used.