Chapter 8

Open System Architecture (OSA)

The aim of Delta-4 *Open Systems Architecture* (OSA) is to offer the whole range of the Delta-4 distribution and dependability techniques in an environment in which heterogeneity must be accommodated. OSA is open to heterogeneous hardware and local executives. In particular, it is able to accommodate off-the-shelf host computers that have not been specialized for fault-tolerance and have *a priori* no built-in mechanisms for that purpose. The Delta-4 OSA infrastructure therefore provides error-processing techniques that do not require restrictive assumptions concerning the failure modes of individual nodes (OSA can accommodate *fail-uncontrolled* host computers). OSA implements all Delta-4 replication techniques (active, passive and semi-active replication).

The basic components of the Delta-4 Open Systems Architecture are:

- A communication system, the Multipoint Communication System (MCS), which was designed according to the ISO/OSI philosophy. MCS includes at any level of this model either new communication protocols, or extensions to existing ISO protocols, in order to achieve the communication requirements implied by the Delta-4 dependability models. MCS provides support for error-processing and basic reconfiguration mechanisms for fault-treatment.
- An administration system that provides a complementary set of mechanisms to handle the system's complexity from the administrative point of view, to support planning and system integration, to assist in daily operations and to help with fault treatment and maintenance. Complying with ISO management framework, it covers the areas of fault-, performance-, configuration- and security management, including all system components and the application itself.

In OSA, applications can be built either directly above these two basic components or preferably above an *Application Support Environment* that will provide the application programmer with a higher level of abstraction and with powerful tools for building his application. Deltase (see chapter 7) is of course a good candidate as such an environment and is available in OSA. Other computational models can be used; in particular, OSI application layer protocols such as MMS (Manufacturing Message Specification) are also available

8.1. Multipoint Communication System (MCS)

8.1.1. Introduction

This section provides a presentation of the Delta-4 *Multipoint Communication System* (MCS). The communication requirements specific to Delta-4 systems, both for fault-tolerance and distribution aspects, are expressed. A multipoint communication model, taking no account of consideration of fault-tolerance, is then defined. An implementation of this multipoint communication model, the Delta-4 MCS, is finally presented, with the addition of fault-tolerance aspects. The architecture of the MCS is described, and an overview of naming and mapping within the MCS is provided.

8.1.2. Delta-4 Communication Requirements

8.1.2.1. Requirements for Distribution. In the Delta-4 architecture, a distributed application is seen as a set of run-time components communicating amongst themselves by means of messages (and only by these means). Even without considering the problem of replication, it may be uncomfortable (although not impossible) to build such distributed applications above point-to-point communication using physical designation, as it is offered by OSI-based systems.

It is often the case within distributed applications that a function is performed by interactions among more than two application entities; these interactions involve communications among the application entities. Current communication standards only allow point-to-point communication, and such multi-peer interactions must be mapped onto a set of peer-to-peer interactions. Furthermore, if some of these interactions must be linked together to fulfil some global requirements (global order, atomicity, consistency,...), protocols must be added to restore this global view.

It is possible within a distributed application to group entities according to the function they execute or to the service they provide together. It would therefore be natural to map such application groups onto similar communication instances, such as associations allowing communication between more than two entities.

Here are some examples of communication between more than two entities within distributed applications:

- In client-server models (Deltase, the Delta-4 Application Support Environment, uses such a model, see chapter 7), a service can be used by many clients. Even if the client-server interactions are point-to-point interactions for every pair (client, server), it is useful to group all interactions related to the same service within a single communication instance associated to the service. This allows communication resources to be economised and decreases the overheads due to communication resource allocation and release every time a new client wants to use the service (this is required only on the client side). It allows the server to have a global perception of all communications related to the same service, and according to the quality of the communication service (order, atomicity,...), to have more facilities in the management of global events affecting the service (e.g., locks, stamps,...). In addition, with such a use of multi-peer communication, the dynamic aspects of communication group membership are important: clients must have the opportunity to join dynamically the communication instance associated with the service when they need to use the service, and to leave it when they do not need it any more.
- Distributed transactions and distributed database management are examples of applications where some operations need to be coordinated in a global way among all entities. The possibility of multicasting some information during certain phases in an atomic way (e.g., commit phase) is of major help, and is rather expensive to implement by means of point-to-point interactions.
- In many applications, data produced at one node can be used by several components residing on different nodes. Alarms and incidents that affect global distributed processing can be forwarded to several nodes for different purposes (archiving, processing, reporting, ...). In this case, the same information must be transmitted

towards multiple receivers and even in an OSI-based system that uses a physical medium allowing data multicasting, high level protocols are not able to take into account such requirements.

- Load balancing within distributed systems often requires the migration of software components from nodes to others and then messages to be delivered to a dynamic, time-varying set of receivers with no a-priori knowledge of their physical location.

All these considerations have led the Delta-4 project to work intensively in the field of multipoint communications, and to define some new communication models, services and protocols, which are described in this chapter and in chapter 10.

8.1.2.2. Requirements for Fault-Tolerance. The implementation of host fault-tolerance techniques based on software component replication leads to important requirements on the communication system. The communication system must provide services that are amenable to communication between replicated software components. The Delta-4 objective of providing transparent fault-tolerance management implies that the programmer of these components must be able to ignore the redundancy involved. This means that the visibility that is given to the application programmer when he programs a communication interaction — a source entity forwarding a message to a destination entity — is that of one single message being sent by the source entity and one single message being received by the destination entity (whatever the degree of replication of the interacting entities) (figure 1).

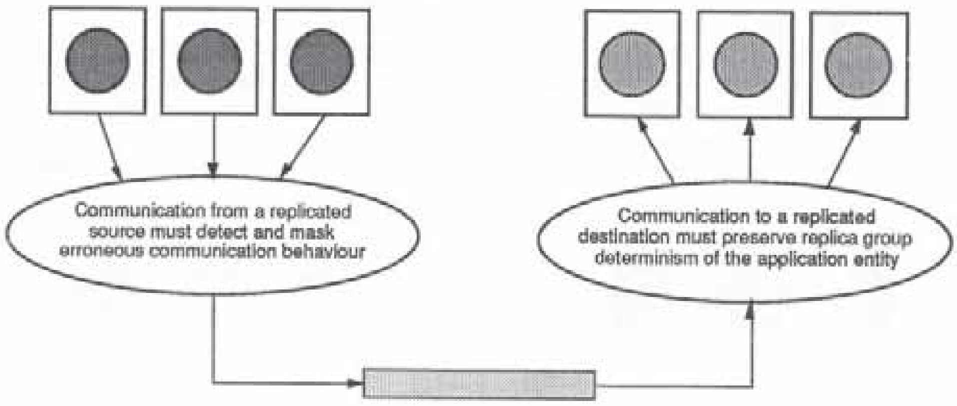

Communication from a replicated source must detect and mask erroneous communication behaviour
Communication to a replicated destination must preserve replica group determinism of the application entity

Fig. 1 - Communication between Replicated Application Entities

To preserve this mutual ignorance of replication by the code of interacting replicated entities, the communication system has therefore to take into account the replicated nature of these entities, and perform some actions that are made necessary by this replication:

- The communication system has to manage the redundancy of information sent by replicated sources. Every time the application programmer invokes a "send" primitive in a program, the replicated execution of this program results in the delivery of this message to the communication system on as many stations as there are replicas. According to the failure assumptions that are made on the concerned host computers, and to the error detection strategy that was selected for the concerned application entity, the communication entities located on these stations then have to reach an agreement to determine the message that will be forwarded to the destination. This agreement may include some mechanisms to detect erroneous communication behaviour resulting from faults in the host environment. Such detection should be associated with error reporting to administration entities, and as far as possible must achieve masking of errors to application entities. This agreement is a distributed agreement, and thus it involves some additional protocol as compared to standard communication protocols; this additional protocol is referred in this document as *Inter Replica protocol* (IRp) (figure 2).

To reach agreement, the IRp entities must exchange some protocol data units; a distributed agreement is considerably easier if these exchanges can rely on underlying transmission protocols that provide atomicity and order. For example, a distributed vote among three values that are sent onto the network by three different IRp entities is quickly performed if these three entities are assured to receive all three values in the same order. In the same way, when the agreement is reached, one IRp entity has to propagate the value to the destination (eventually replicated), together with an indication to the peer IRp entities that this propagation has taken place. This set of actions (propagate the value to the destination, inform the peer entities) must be atomic, and it is highly beneficial if it can be done by using a single atomic multicast primitive.

- The communication system has also to preserve the consistency of all copies of a replicated entity when it forwards some messages to this entity. As the application programmer must not be aware of replication, the provision of messages by the communication system to the component when "receive" primitives are invoked must not introduce any deviation in the behaviour of some replicas with respect to others. If the program excludes some constructs that may preclude replica determinism (i.e., the same inputs applied to the replicas produce the same outputs), a sufficient condition to ensure that consistency is maintained is to provide to all replicas with the same ordered set of messages (figure 2). This service is provided by a protocol referred to in this document as the *Atomic Multicast protocol* (AMp).

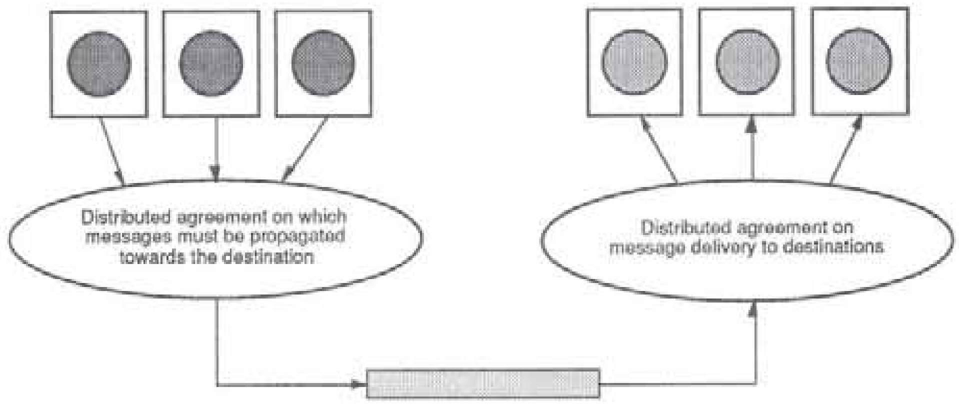

Distributed agreement on which messages must be propagated towards the destination
Distributed agreement on message delivery to destinations

Fig. 2 - Distributed Agreement Protocols.

It therefore appears that the communication requirements of interacting replicated components can be fulfilled by an *inter replica protocol*, which relies on the services provided by an underlying *atomic multicast protocol*.

8.1.3. Multipoint Communication Model

8.1.3.1. Introduction and Definitions. The aim of this section is to define a multipoint communication model that covers the requirements expressed above for multi-peer communication between different application entities within distributed applications. It refers to existing standards, in particular the Open Systems Interconnection Reference Model [ISO 7498-1] and related standards, and to the work that was carried out in ISO/TC97/SC21 about multi-peer data transmission before this item of study was suspended (this work had produced a working draft addendum to ISO 7498-1 on Multi-Peer Data Transmission [ISO N2031]).

Some terminology from the latter document that was found convenient to use in this chapter and following ones is repeated here:

instance of communication	a single instance of a connectionless transmission or a single instance of a connection
multi-peer data transmission	transmission of a data unit to one or more destinations
multi-peer group	a group of peer entities which are mutually willing and able to be senders or receivers of multi-peer data transmissions with other members of the group
defined group	a multi-peer group whose total membership is known to members of the group
partially defined group	a multi-peer group whose total membership is known to some of the members of the group
undefined group	a multi-peer group whose total membership is not known to any members of the group
static group	a multi-peer group whose membership can only be altered outside the operations of an instance of communication between the entities of the multi-peer group
dynamic group	a multi-peer group whose membership may be controlled and altered by some or all members of the group
invoked group	the sub-group of a multi-peer group with which communication is attempted in an instance of communication
active group	the sub-group of a multi-peer group which is participating in an instance of communication
sub-group	some or all of the members of a group
central entity	an entity that is able within an instance of communication to transmit data to all other members of the active group. An entity that is not a central entity can only send data to the central entity.
centralized communication	a type of communication where, within an instance of communication, there is only one central entity and it cannot change.

roving centralized communication	a type of communication where, within an instance of communication, there can be many central entities but there can only be one central entity at any one time.
multi-centred communication	a type of communication where, within an instance of communication, there is more than one central entity.
decentralized communication	a type of communication where, within an instance of communication, all entities of the invoked group are central entities.

8.1.3.2. Multipoint Communication Model. The objective is to allow groups of communications entities to communicate amongst themselves within a multi-peer communication instance. An instance of communication is defined as either a single instance of a connectionless transmission or a single instance of a communication.

Connectionless communication, as defined in [ISO 7498-1], allows the transmission of copies of a data unit to several (more than one) destination addresses. However in that case, every data unit is transferred independently from others by the layer (or layers) that provides the connectionless service. All information that is necessary to transmit a data unit (destination address, quality of service, options,...) is presented to the connectionless service provider together with the data unit to be transmitted, in a single access to the service. The service provider does not have to establish a relationship between this access and others. This means that some important characteristics that may be required by applications (e.g., order) are not provided. This chapter does not therefore consider the connectionless case and concentrates on the definition of a multipoint communication model that would provide a service similar to connection-oriented communication, as defined in the OSI Reference Model [ISO 7498-1].

According to [ISO 7498-1], a connection is an association that is established for transferring data between two or more peer entities. This association is established between the peer entities themselves and between each entity and the layer that provides the connection-oriented service. The use of the connection-oriented service by peer entities has three distinct phases:

1) connection establishment;
2) data transfer;
3) connection release.

An OSI connection has the following fundamental characteristics:

a) it implies the establishment and maintenance of a three or more party agreement concerning the data transfer between the concerned peer entities and the service provider;
b) it allows the negotiation, between all concerned parties, of the parameters and options that will be used during data transmission;
c) it provides the connection identifier which is the means to avoid, during transmission, the overhead due to address transmission;
d) it provides a context in which all successive data units transferred between peer entities are logically linked, and it allows sequence preservation and flow control for these transmissions.

These definitions, if strictly applied to multi-peer groups, would not give the sufficient flexibility to meet the requirements of some distributed applications:

- Respecting phases 1), 2) and 3) only allows communication to take place among members of static groups. The membership of a communication instance cannot be changed during the lifetime of the connection. All members that want to participate to the instance of communication must agree on the connection establishment during the same phase, and all members must release the connection together.
- Characteristics a) and b) imply that the negotiation to establish the communication instance (the connection) takes place among all the members of the group. According to the number of group members, this can be the cause of a considerable overhead.

To allow a more dynamic scheme in the case of multi-peer connection-oriented communication, it seems that these general definitions should be slightly extended; these extensions are proposed here.

8.1.3.3. The Multi-Peer Connection (or Association)

8.1.3.3.1. Main Features. A multi-peer connection is an association that is established for transferring data between two or more peer entities. This is exactly the OSI definition; but whereas in OSI a connection only exists if the peer entities have decided to establish it and have negotiated to do so, and ceases to exist as soon as one of these entities releases it (thus the three phases: establishment — data transfer — release), it is possible to define more phases in the lifetime of a multi-peer connection.

The more general case to be considered is the one of dynamic multi-peer connections. In such communication instances, members of the communication group are allowed to join and leave the connection dynamically during the life of the connection. The consequence is that a multi-peer connection exists as soon as one entity has decided to join it. In other terms, the connection establishment primitive, which in the OSI case means "I want to communicate with entity X", simply means in this multi-peer context "I want to join (to participate in) communication instance I".

This dynamic aspect avoids the necessity for a new negotiation to take place between all active members of the communication group every time a new member wants to join the group. Apart the fact that such a negotiation would be a non negligible performance penalty, it appears impossible to change communication parameters and options of an instance of communication during its life. However, there is still the need that all peer entities participating in the same communication instance use the same communication parameters and options. Several possibilities exist to ensure this:

1) All possible members of a multi-peer communication instance have means to a-priori agree on these parameters and options; this implies that these parameters and options are defined before the actual instantiation of the connection. In that case, the negotiation when joining the multi-peer connection can be reduced to a check that the new member wants to use the right parameters and options for the connection it wants to join. This check can be carried out either by the service provider, or by a particular entity that coordinates the multi-peer connection. The first case assumes that the provider has some means of knowing these parameters and options at the very beginning of the communication instance (when the first member joins it). The second case implies that the first member to join the connection must be the coordinator, that the provider knows that there is a coordinator and knows its identity, and then again has some means to get this knowledge at the very beginning of the connection life.
2) The new members get the knowledge of communication parameters and options when they join the connection. This knowledge can be obtained from the provider or

from a coordinating entity. It still implies, as in 1), that the provider has some means of knowing some options at the very beginning of the connection life. However this case does not seem very realistic, because it would make it rather difficult to program such entities (which get all communication parameters and options only when they want to start communicating).

In any case, it appears that dynamic multi-peer connections must have some existence before the actual instance of communication; the main operations that can occur during the lifetime of a multi-peer connection are then:

1) create a multi-peer connection;
2) open a multi-peer connection;
3) join a multi-peer connection;
4) multi-peer data transfer;
5) leave a multi-peer connection;
6) close a multi-peer connection;
7) delete a multi-peer connection.

8.1.3.3.2. Create a Multi-Peer Connection. This operation consists of creating an administrative view of a multi-peer connection before the actual use of the corresponding communication instance. This administrative view includes all parameters and options that will be used when communicating with this multi-peer connection. Some of these parameters can be used for membership control (access rights, restricted membership,...). A minimum membership can also be defined, that is to say a list of members that must join the connection before any data transfer service is available on the connection.

No assumption is made on how this administrative view is represented and where it is stored. It is simply assumed that application entities have means to access this information, and that the service provider is also able to get them when needed.

8.1.3.3.3. Open a Multi-Peer Connection. It is conceivable that several peer entities located in the same end system use the same multi-peer connection (an end system is a node of the distributed system where application entities execute and use the communication system). In that case, the service provider must not allocate several times the resources it needs to manage the multi-peer connection (connection context). Opening a multi-peer connection therefore consists in allocating the resources that are necessary to instantiate the connection in an end system. The open operation also consists of creating the necessary underlying means to allow communication between the end system and all other end systems where the connection is already open. These means need to be created only once in the lifetime of the connection in a particular end system, and this operation has not to be repeated every time a new entity joins the connection on this end system.

The open operation can be implicitly invoked when the first entity wants to join the connection on the end system, or can be explicitly invoked before any entity joins it.

During the opening of the connection, the service provider gets the parameters and options of the connection from the administrative view that was created by the create operation.

8.1.3.3.4. Join a Multi-Peer Connection. A peer entity can join a multi-peer connection at any time in its lifetime. The entity provides to the service provider the communication parameters and options, and the provider checks that these parameters and options correspond to those of the connection, or transmit them to some coordinator that will perform the checking.

If the parameters and options proposed by the entity do not fit with those of the connection, the join operation is rejected by the service provider.

If a minimum membership is necessary for the data transfer services to be available on the connection, the join confirmation from the provider to the joining entity can be delayed until all required members have joined the connection.

According to whether the multi-peer connection is a defined group, the join operation may be notified to the other active members of the group. If the group changes are not notified to other members, and if the connection is already open on the end system, the join operation can be a local operation that consists only in checking parameters and options and allocating some additional resources for the entity that joins the connection.

8.1.3.3.5. Multi-Peer Data Transfer. As soon as one entity has successfully joined a multi-peer connection, multi-peer data transfer services are available to this entity to communicate with other entities.

Several modes of multi-peer data transfer can be defined:

- *Broadcast transfer:* a data unit transferred on a multi-peer connection using this mode is delivered to all entities that have joined the connection. The sender of the data unit can even be included in the broadcast group.
- *Multicast transfer:* a data unit transferred on a multi-peer connection using this mode is delivered to all entities that are explicitly specified by the sending entity. This specification can take the form of a list of addresses of entities, or of the address of a sub-group of the active communicating group.

Data transfer can be decentralized, centralized, roving centralized or multi-centred. If the communication is roving centralized, some service primitives must be associated with the data transfer services to allow the communication centre to change.

The quality of service of multi-peer data transfer is a parameter of the multi-peer connection. Important properties can be defined in this parameter, such as order properties, priority criteria, etc.

8.1.3.3.6. Leave a Multi-Peer Connection. An entity can decide to leave a multi-peer connection at any moment of its lifetime. This departure can be notified to other members of the connection, according to connection options. If a minimum membership is required for the connection, the provider can decide to force a leaving of all other members and a closing of the connection on all end systems when this minimum membership is no longer available.

Closing the connection on an end system can be performed automatically by the provider if all entities have left it on this end system. Conversely, an option of the connection can be to keep it open, to avoid further re-opening when entities re-join it later on.

Two types of connection leave can be defined:

- *Normal leave:* this is requested by the members and must be done while preserving some properties of on-going data transfer.
- *Abnormal leave* (or abort): this can be requested by the members or provoked by the provider on some abnormal events, and which can be done with loss of some data transfer properties.

8.1.3.3.7. Close a Multi-Peer Connection. This operation deletes the instance of a multi-peer connection on an end system. According to options, if some entities had joined the connection on this end system, they can be forced to leave the connection, or the close operation can be rejected. The close operation also relinquishes the underlying means that were used to allow communication with other end systems where the connection is open.

8.1.3.3.8. Delete a Multi-Peer Connection. This operation deletes the administrative view of a multi-peer connection created by the create operation. This deletion is only possible when no communication instance of the multi-peer connection exists in the distributed system.

8.1.4. The Delta-4 Communication System

8.1.4.1. Communication models. The Delta-4 Multipoint Communication System (MCS) is a layered communication system designed to meet the Delta-4 communication requirements as expressed in section 8.1.2, for distributed applications running on host computers interconnected by Local Area Networks.

The Delta-4 MCS proposes an implementation of the multipoint communication model that is introduced above, by defining multipoint communication protocols at the appropriate levels of the OSI reference model.

The Delta-4 MCS meets the Delta-4 requirements for fault-tolerance by including at the bottom of the session layer an Inter-Replica protocol (IRp). This protocol is able to handle communication from and to replicated application entities and hides from them the management of this replication. The application entities communicate between themselves either by using the multipoint communication model, or by using the ISO bi-point communication model (figure 3).

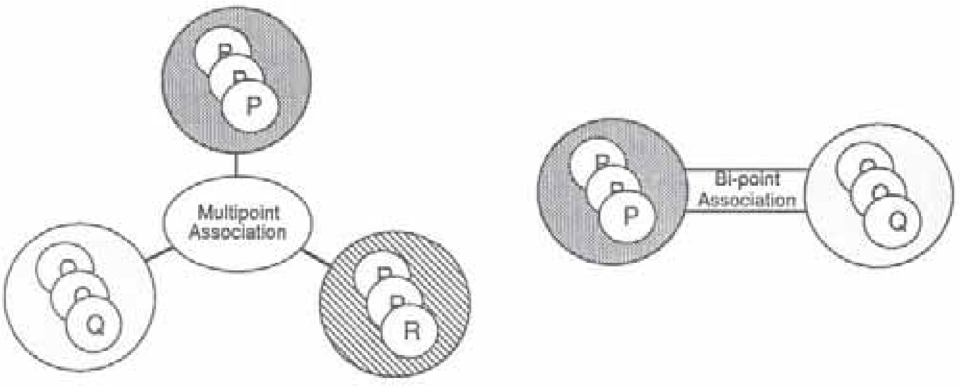


Fig. 3 - MCS Communication Models

Above services provided by IRp, high layers implement either the multipoint communication model or the more classical OSI bi-point communication model (thus allowing applications designed with this last model to take advantage of replication facilities offered by IRp).

The inter replica protocol is built upon multipoint communication services that are provided by multipoint communication protocols in layers 2 to 4 of the Reference Model. When no replication has to be managed, the IRp acts as an inactive protocol, and allows both high layers multipoint and bi-point model to use multipoint services of low layers.

The low layers multipoint protocols use multicast facilities provided by standard Local Area Networks, such as Token Ring [ISO 8802-5], Token Bus [ISO 8802-4] and FDDI [ISO 9314].

There are two functional parts to MCS: the Multipoint Communication Protocol stack (MCP) and the Network Administration System, or Multipoint Communication Management (MCM) which manages these protocols. The latter is dealt with as part of the more general notion of system administration that is covered in section 8.2. This section presents the MCP stack, the services it offers and its architecture.

8.1.4.2. MCS Multipoint Communication Model

8.1.4.2.1. Definitions. In the following, an end system within a Delta-4 distributed system is termed a *station*.

An *application process* is an element within a real open system that takes part in the execution of one or more distributed information processing tasks. An *application entity* is that part of an application process that deals with the communication system. An application entity represents a particular communication activity of the application process.

The application entities have access to the MCS through *M-SAPs* (MCS Service Access Points): the M-SAP is an individual access point to the higher-level services offered by the MCS.

At the high level (ACSE[1]/presentation) a multi-peer connection is termed a *multipoint association*. The application entities are linked by *associations* and communicate between them by using the data transfer services available on these associations. When an application entity joins an association through an M-SAP, MCS creates an *endpoint* to that association, which is fully identified by the couple (M-SAP name, association name) (figure 4).

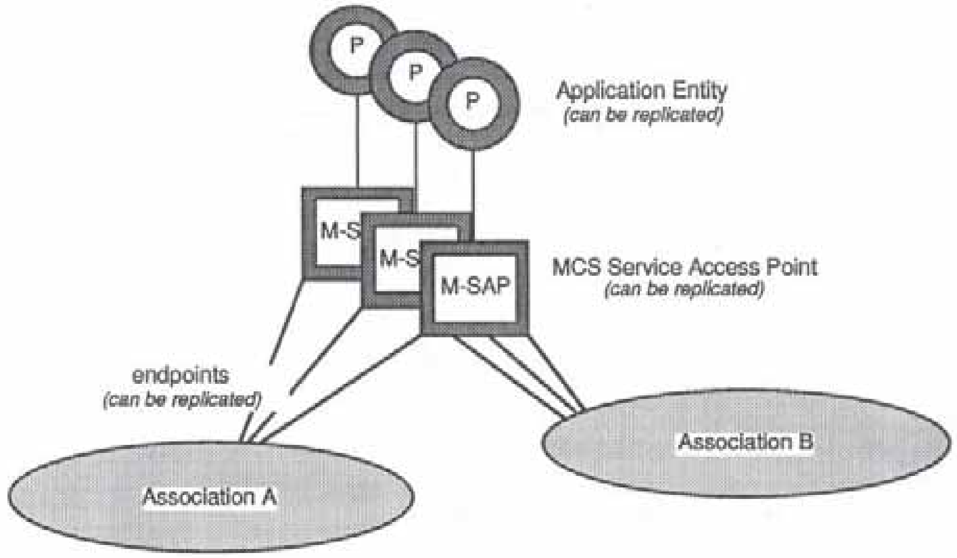


Fig. 4 - Main MCS Communication Objects

1 Assocation Control Service Element

8.1.4.2.2. Create and Delete an MCS Multipoint Association. A multipoint association represents a group of communicating application entities. The creation of such an association is an administrative operation that can be requested by a single application entity, or by a human operator. On creation, the characteristics of the association (name, available services (sequencer, token,...)) are defined and stored in a global administrative data base, the global Management Information Base (MIB). The name of the association can be obtained from a global name server, because association names must be unambiguously allocated in a single naming space within a Delta-4 system.

MCS multipoint associations have a dynamic membership, which is not controlled by any coordinating entity. There is no minimum membership for MCS associations. According to an association option, the changes to the association membership are notified to the association members.

8.1.4.2.3. Join and Leave, Open and Close an MCS Multipoint Association. Application entities can unilaterally join or leave MCS multipoint associations. When joining an association, the application entity must provide the parameters and options related to this association, which are checked by the ACSE provider (figure 5). If the association has not yet been opened on the station where the application entity wants to join it, the ACSE provider automatically opens the corresponding association and session connection. To do so, it obtains the administrative view of the association using support from the System Management Application Process (SMAP).

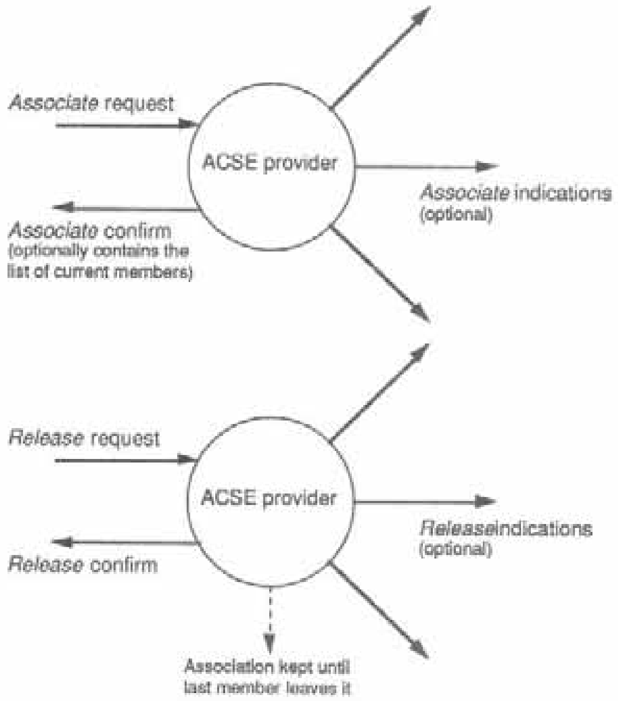


Fig. 5 - MCS Association Joining and Leaving

When all application entities have left an MCS multipoint association on a particular station, the association is automatically closed by the ACSE provider.

If the association has the defined membership option, join and leave operations are notified to all active members of the multipoint association. New members joining such an association also get the knowledge of the current active membership.

8.1.4.2.4. Data Transfer on Multipoint MCS Associations. There are three different modes for transferring information over an MCS multipoint association:

- *Inclusive multicast:* the information is sent to all members of the association including the sender.
- *Exclusive multicast:* the information is sent to all members of the association except the sender.
- *Selective multicast:* the information is sent to a sub-set of members of the association designated either by their logical names or by a common sub-group identifier.

Ordering of message delivery over a multipoint association is such that two application entities receiving two messages from an association receive them in the same order.

All entities connected to a multipoint association may both send and receive messages. A multipoint association is characterized by the type of dialogue that manages the interactions between the members of the association:

- *Decentralized dialogue:* all the entities are simultaneously able to initiate message transmissions.
- *Roving centralized dialogue:* only one entity at a time is able to send data (i.e., the one that possesses the data token of the association). Primitives are provided to allow the association token to circulate among association members.

Multipoint associations also provide a sequencer service that allows the members of the association to get consecutive positive ticket values. This enables the application to stamp and thus to order detected events that modify the state of the distributed application. Up to two sequencers can be used on an MCS multipoint association.

8.1.4.3. Concepts for Fault-Tolerance

8.1.4.3.1. Error Detection and Masking. The Delta-4 communication system meets the requirements created by the replication of application entities on different host computers of a distributed system. It is able to handle communication between replicated application entities in a way that is transparent to the programmer of these entities. This transparency is sometimes termed *invisible multicasting*.

The inter replica protocol (IRp), which acts as a bottom sub-layer of the MCS session layer, handles this type of communication. It includes error detection and masking mechanisms for replicated communication issued from application entities actively replicated on fail-uncontrolled hosts. These mechanisms also allow error masking for replicated application entities (either active replication, or passive, or semi-active, see chapter 6) running on fail-silent host computers; in this case, the error detection mechanisms are located in the host fail-silent environment.

The inter replica protocol relies on the services provided by the low layers of the MCS, essentially the atomic multicast service, and the notification of station failures.

8.1.4.3.2. Impact on Communication Objects. A replicated application entity is one that has been replicated on different stations for the purposes of fault-tolerance. When such a replicated application entity uses MCS services, it does so through an M-SAP (MCS Service Access Point). Generally, Service Access Points have only a local significance, in the station where the service is accessed. However, in the case of replicated entities, the service is accessed by the same application entity in several stations in the system. This is the reason that leads us to consider in MCS that M-SAPs used by replicated entities are themselves also replicated. In the same way, all endpoints that would be created by a replicated application entity through a replicated M-SAP are also considered as replicated endpoints (figure 4).

8.1.4.3.3. Services for Fault-Tolerance. As M-SAPs can have a global significance, they must be known and defined in a global way. M-SAPs are defined by administrative primitives before being used, and this administrative view of M-SAPs is stored in a "Global Management Information Base" or "Global-MIB". M-SAPs have some attributes that are used by IRp to coordinate the replicas of the M-SAP (and then to control the replicated behaviour of the corresponding AE). Some of these attributes are:

- the degree of replication of the M-SAP;
- the replication domain of the M-SAP (the list of stations where a replica of the M-SAP exists);
- the allowed desynchronization delay of data transfer request issued through this M-SAP;
- the type of error detection mechanism to be used;
- ...

The IRp provides to administration entities some means to manage fault treatment and reconfiguration:

- error reporting (voting disagreement between replicas, early or late timing errors, abnormal overflow of one replica,...);
- primitives allowing dynamic reconfiguration of M-SAPs (dynamic creation or deletion of M-SAP replicas), transparently to the replicated AE.

As explained previously, the IRp fully relies on the atomic multicast services that are offered by low layers, and which are built on the basic atomic multicast protocol (AMp). In particular, the AMp is able to detect and signal, both to upper layers (and thus, IRp) and to administrative entities (SMAP), the failure of stations in the Delta-4 distributed system. This property is intensively used by error recovery and fault-treatment mechanisms.

In addition, IRp uses the AMp facilities to provide to replicated AEs a global order for all data units received from several associations through the same replicated M-SAP. This property, termed "deterministic receiving", is an attribute of the M-SAP and can maintain determinism of a replicated AE communicating through the same M-SAP on several MCS associations.

8.1.4.4. Structure of the Multipoint Communication Protocol Stack. The MCP stack is a communication stack that follows the layering principle of the OSI Reference Model. No standardized multipoint protocols yet exist. Consequently, Delta-4 proposes a hierarchy of specific protocols for providing the multipoint communication services. Despite the specific nature of these protocols, there is a total compatibility with ISO standards for the physical layer and medium access control protocols, thus allowing the coexistence of the MCS and ISO protocols on the same LANs. High layer ISO standards can also work on top of IRp (with a logical addressing scheme). This allows applications using such standards to be ported on to the MCS stack and to benefit from the fault-tolerance techniques provided by Delta-4.

Figure 6 shows the layering of the Delta-4 MCS; those protocols that were designed and developed within the Delta-4 project are put in bold characters, those protocols that follow existing standards are put in italic characters.

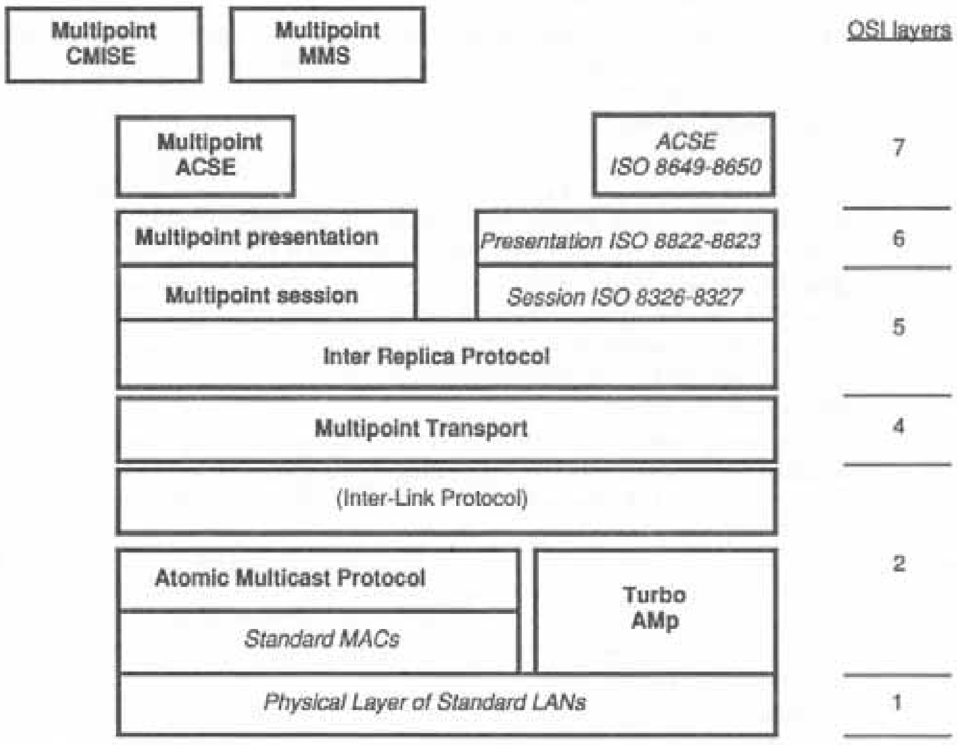


Fig. 6 - MCS Layers

The services and protocols for every layer are described in the MCS section of the Delta-4 Implementation Guide [Delta-4 1991]. The services and protocols are not described in detail when they conform to existing standards. The reader can refer to these standards. The communication profiles that are selected for these protocols are inherited from the ESPRIT project 2617 (CNMA), and are described in detail in the CNMA Implementation Guides.

The protocol and the service provided by the bi-point session conform to ISO standards 8326 and 8327 (FU kernel and FU duplex).

As an exercise to port application entities using ISO communication, and to let these AEs benefit from the Delta-4 fault-tolerance techniques, the use of the application level Manufacturing Message Specification [ISO 9506] by replicated Application Entities was prototyped during the Delta-4 project. A multipoint MMS, using MCS multipoint facilities, has also been specified and implemented [MP MMS].

The MCP stack uses standard communication mediums and the physical layer conforms fully with these standards. The atomic multicast protocol, which is one of the main Delta-4 specific protocols, constitutes the basis for the management of multipoint communication and communication between replicated entities. Two versions of this protocol have been specified within the Delta-4 project:

- The "generic" atomic multicast protocol is a software implementation of AMp. It is medium-independent, and is built upon standard implementations of MAC protocols for Token Ring [ISO 8802-5], Token Bus [ISO 8802-4] and FDDI [ISO 9314] Local Area Networks. It allows the use of commercially available chipsets for these LANs without any modification.

  The generic AMp provides a confirmed service of atomic multicasting of a single data unit between groups of stations interconnected in a single LAN medium. It manages the membership of these groups, and notifies upper layers of significant group changes (among them station failures).

  The data transfer service offered by generic AMp is similar to the acknowledged single frame transmission of LLC type 3, but extended to multiple destinations. The other difference is that generic AMp cannot be considered as connectionless, but follows the definition of multi-peer connections given in this document.

  Selection and grouping facilities are built upon the service provided by generic AMp, to provide efficient support to higher layers for mapping selective multicasting onto AMp groups. These facilities also allow IRp to propagate efficiently and atomically grouped data units to different sub-groups communicating in the same high-level communication instance.

- The enhanced atomic multicast protocol (or "Turbo-AMp") is a version of AMp specially targetted to the 8802-5 Token Ring. It uses special properties of the Token Ring structure and protocol (in particular, the ability of "on the fly" acknowledgements) to implement AMp in a very efficient way (atomic multicast in one round trip in the normal case). It involves development of hardware and firmware around commercially available chipsets. Nevertheless, Turbo-AMp preserves compatibility with the standard at the medium level and allows the coexistence on the same medium of stations using ISO stacks on standard implementations of [ISO 8802-5].

  Selection and grouping facilities are inherent to Turbo-AMp.

On top of AMp and selection and grouping facilities, the Inter-Link protocol extends the service to interconnections of LANs. The transport protocol is a light connection-oriented protocol that adds to underlying services functionalities for segmenting and reassembling, and data flow control.

8.1.4.5. Addressing and Mapping of Communication Objects. The MCP stack uses at any level a logical addressing: communication objects are identified by a logical address that is allocated in a space of 2 power 20 possible addresses.

The communication objects that are visible to the user have a logical address in that space:

- An M-SAP is identified by a unique logical address; if this M-SAP is replicated, all the copies of this M-SAP have the same address (even though they are located on several distinct physical stations).
- An association is identified by a unique logical name; when an association is opened on several stations, the identification of this association is the same in any one of these stations.
- An endpoint is fully identified by the couple (M-SAP logical address, association logical name) constituted by the logical name of the association to which this

endpoint is related and the logical address of the M-SAP through which it was created.

The atomic multicast protocol manages logical gates: a gate defines a group of stations within a single LAN segment.

The transport layer manages multipoint transport connections: there is at most one endpoint of a transport connection on one station. There is a one to one mapping of transport connection endpoints onto logical gates. Every time a transport connection endpoint is created on a station, the corresponding AMp logical gate is also opened.

There is a one to one mapping of associations onto transport connections. When an association is opened on a station, the corresponding transport endpoint is created on the station, and therefore the logical AMp gate is opened.

Because of this one to one mapping of associations to transport connections, and to AMp gates, all these communication objects have the same name.

When an M-SAP is replicated, IRp entities that manage this M-SAP need sometimes to communicate among themselves, out of the control of AEs. A replicated M-SAP is therefore mapped onto a transport connection, and then onto an AMp logical gate, which has the same name as the M-SAP address.

It therefore appears that association names and M-SAP addresses are allocated in a unique naming space.

The selection and grouping facilities provided above AMp are used to give to the low level the visibility of high level endpoints and sub-groups, to allow a more efficient implementation of selective multicasting (figure 7). This means that joins and leaves within MCS multipoint associations are made known to transport and to underlying facilities, which offer services to maintain consistent views of sub-groups of atomic multicast groups.

8.2. System Administration

The Delta-4 multipoint communication protocol system, described in section 8.1, which provides the basis for dependability, and the Deltase support of its high-level transparent usage by programmers, described in chapter 7, are supplemented by a third concept, the Delta-4 System Administration. The properties of the envisaged application areas and the fault tolerance approach of Delta-4 require powerful mechanisms to handle system complexity from the administrative point of view, to support planning and system integration, to assist in daily operations and to help with fault-treatment and maintenance. The set of such facilities is referred to here as system administration.

This section outlines the main functional requirements of system administration, the applied structuring concepts and the derived architecture. Within OSA, the Delta-4 approach to system administration follows the ISO/OSI related work on systems management.

8.2.1. Functional Requirements

The management of distributed systems is currently under intense discussion in the academic field as well as in the immense ISO standardization work, international multi-vendor initiatives (MAP, CNMA, OSI/Network Management Forum) and network management product developments (IBM's Netview, HP's OpenView, DEC's EMA — Enterprise Management Architecture).

The functional view of Delta-4 system administration goes beyond the scope of these activities as a consequence of the sophisticated fault tolerance approach. This consequence also implies that administration is not limited to the management of communication resources, but encompasses all kinds of data processing resources.

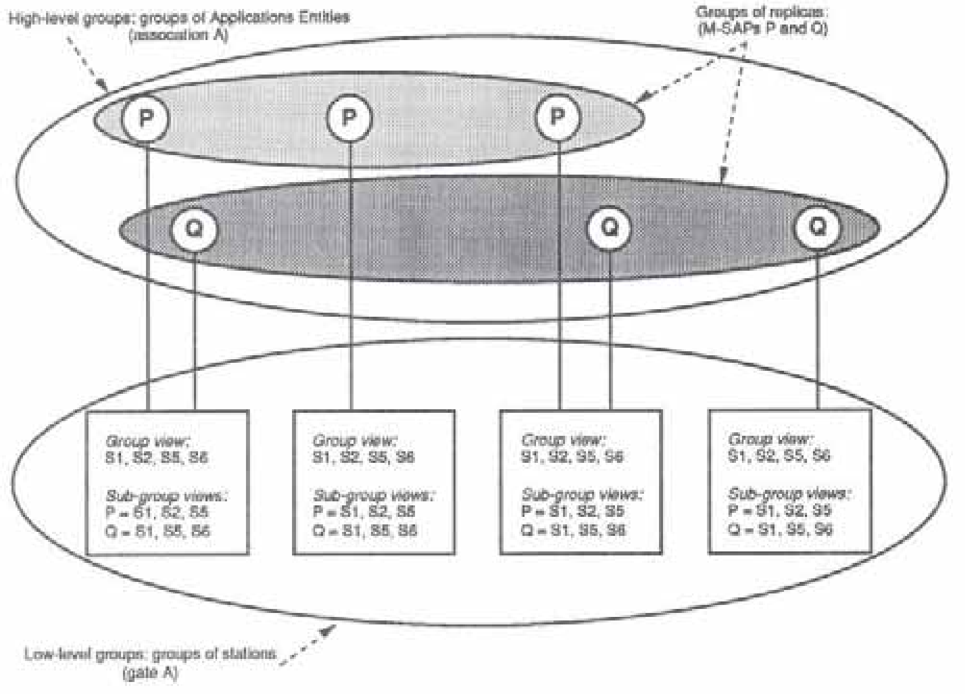

High-level groups: groups of Applications Entities (association A)
Groups of replicas: (M-SAPs P and Q)
P
P
P
Q
Q
Q
Group view: S1, S2, S5, S6
Sub-group views: P = S1, S2, S5 Q = S1, S5, S6
Group view: S1, S2, S5, S6
Sub-group views: P = S1, S2, S5 Q = S1, S5, S6
Group view: S1, S2, S5, S6
Sub-group views: P = S1, S2, S5 Q = S1, S5, S6
Group view: S1, S2, S5, S6
Sub-group views: P = S1, S2, S5 Q = S1, S5, S6
Low-level groups: groups of stations (gate A)

Fig. 7 - Communication Groups Mapping in MCS

8.2.1.1. The System Administration Space. This section aims to structure the functional view of system administration in a fault-tolerant open distributed system. Figure 8 shows the main dimensions that stretch the administration space and categorize the requirements for building an integrated administration system.

The dimension "Functions" corresponds to the management functional areas, which are used to structure the OSI-Management related ISO work. Of particular interest within the Delta-4 approach are fault and configuration management and to some extent the evaluation of the performance of a system, but the project also investigates specific topics on security management (see chapter 13).

A second dimension is dedicated to the components to which the management functions are applied. This is of course the MCP-stack itself. The fault tolerance approach based on replicated software components implies the need for fault and configuration management for *replicated* files and processes or general Deltase applications.

The different life cycle phases that a Delta-4 system may follow constitute a third dimension for categorising administrative requirements. There are design, planning, installation and commissioning phases, normal daily operation, and phases where the configuration changes with respect to station failures or system evolution.

8.2.1.2. Administrative Tasks. In the following a general view is given by defining three overall administrative tasks (ATs) along the life cycle phases of a Delta-4 system.

AT1: Planning and integration of redundancy and distribution;

AT2: Monitoring of system behaviour;

AT3: Fault treatment and maintenance

The illustration of the ATs is restricted here to the main topics of the Delta-4 approach. The overall management model and the implementation principles are however designed to provide support for all aspects of system administration.

The requirements of the Administrative Tasks identified below are concerned with the Delta-4 attributes "fault tolerance" and "distribution".

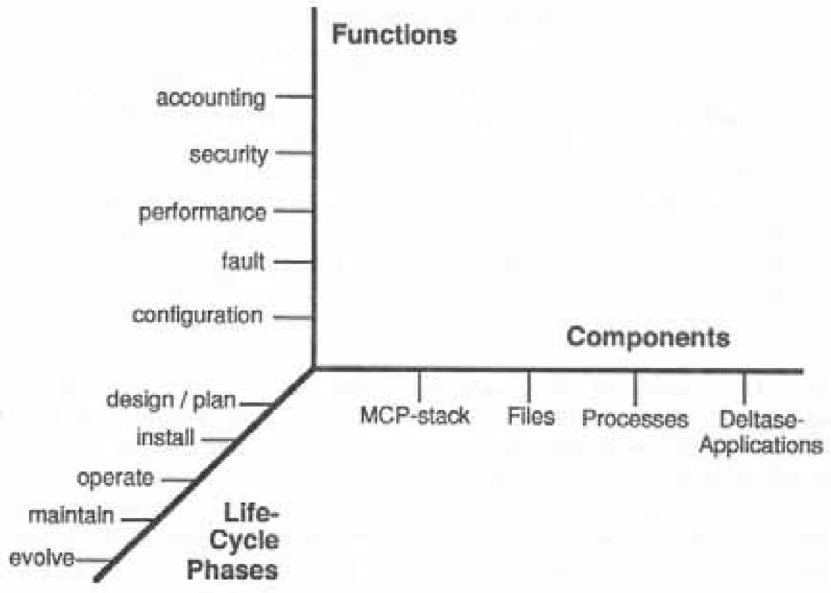


Fig. 8 - The System Administration Space

8.2.1.2.1. AT1: Planning and Integration of Redundancy and Distribution. The first step to configuring a distributed application is that of identifying application-specific characteristics such as:

- topological configuration (e.g., process control interface constraints);
- services to be made dependable (replicated software components);
- degree of possible redundancy (software component replication domains);
- error-processing modes dependent on host fault assumptions and required dependability attributes;
- contingency plans for application-specific reconfiguration strategies.

The system designer's experience is usually supported by simulation tools. Actual measured management information, derived from comparable configurations, can provide a quality of simulation significantly better than when using parameters derived only from the equipment's technical data.

The system integration phase in the real or a simulated application environment is used to validate the design decisions and to tune the system's operating parameters. In particular, administrative information and actions are required to support the following:

- Testing the fault tolerance mechanisms by using performable actions (on-line fault injection) to stimulate the use of available redundancy (passive replicas, cloning of further active replicas). Stimuli should cover all expected operational conditions such as failures of host and communication resources, buffer overflows and general overloading. This simulation of operational situations must also be carried out to demonstrate the system's capability for evolution.
- Performance evaluation of hosts and the communication system requires actions to initiate time-based measurement to obtain appropriate statistical data of a real or simulated workload under different operational (fault) conditions.
- Dimensioning the system parameters such as communication buffer and window sizes, priorities and schedule frequencies derived from the offered statistical data is an important and difficult task. Of particular interest is the first rough layout of the time-out parameters required by the various error-processing protocols. This must be done on a per replication-component basis (real-time constraints, workload conditions) and is an iterative process that can be performed with human interaction or automatically by systems management. In either case, during the system integration phase, "slow" hosts are treated as being non-faulty until time-outs have been tuned to an acceptable compromise between spurious error detection and acceptable real-time responsiveness.

8.2.1.2.2. AT2: Monitoring of System Behaviour. By the nature of the Delta-4 fault-tolerance approach, the required human interactions to preserve the expected system behaviour should be minimized. The error-processing protocols of the communication system must be supplemented by powerful system administration facilities for fault treatment.

AT2 provides the essential input for AT3 to carry out successful fault treatment and maintenance by gathering appropriate system information such as actual configuration data (redundancy used, current allocation of replicas, etc.), performance statistics and the numbers of (recovered) errors. The monitoring activity may be carried out by periodic polling, event-triggered polling, and/or by counting events such as error reports from the error-processing protocol entities.

Additional tests might be used to check the absence of faults within unused redundant system components. For example, a dual-ring LAN may require that the current inactive transmission direction be periodically tested in case a failure of the active one should occur.

8.2.1.2.3. AT3: Fault Treatment and Maintenance. The traditional "network management" view of AT3 is that of providing long-term management functions for system maintenance staff. Within a truly fault-tolerant distributed system environment this task is much more critical since it pertains to the prevention of serious consequences when faults accumulate. It should thus be automated as far as possible.

Fault treatment is achieved by supporting:

- fault diagnosis;
- fault passivation;
- system reconfiguration; and
- system maintenance.

Fault diagnosis is necessary to decide if a fault is permanent (e.g., judging the significance of time-out occurrence within the assumed and actual workload profile) and to assist in fault localization (e.g., by evaluation of error reports from error-processing protocol entities). If fault diagnosis should conclude that a permanent fault has occurred, then fault passivation must be carried out and system reconfiguration envisaged.

Fault passivation of fail-uncontrolled hosts must be done automatically by the integrated administration system, i.e., manual intervention should not be required. Note that fail-silent hosts, by definition, carry out automatic and autonomous fault passivation.

System reconfiguration can be envisaged if there are sufficient redundant resources. It entails re-allocation of the software component replicas that were resident on failed hosts to restore the level of redundancy required for the error-processing protocols to function correctly despite further faults. If re-allocation is not possible, then some software components may either have to be abandoned in favour of more critical ones. Alternatively, fault-tolerant operation is degraded to fail-safe operation to ensure safety and/or integrity of the distributed application(s).

Re-allocation of software component replicas is achieved by means of a cloning operation that creates a new replica on a specified node. Three sub-operations can be identified:

- Creation of a template of the software component at the new location. This can be done in advance of an actual cloning request according to application-specific contingency plans specified by AT1 (e.g., localization of passive replicas, designation of degraded modes,...).
- Creation of a copy of the component's persistent data or "state" at the new location.
- Activation of the new replica whilst ensuring replica-consistency. This involves the automatic management of the dynamic, configuration-dependent associations between replicated components.

Two techniques for cloning with different performance characteristics can be considered; they are termed *recursive transfer* and *snap-shot transfer*.

Recursive transfer requires the continuous identification and tagging of structures that are modified in the active replica(s) while a state transfer is attempted. The state transfer sub-operation is then repeated using only the tagged data. This is carried out recursively until no data structures are tagged during state transfer.

Snap-shot transfer involves the creation of a local copy (or "snap-shot") of the component's state on the host of the active replica(s). While this snap-shot is transferred to the new location, a log of messages sent to the software component is constructed (at the new location). When the snap-shot transfer is complete, the new replica processes the stored messages.

Other optimizations may be possible in the cloning operation; for instance the partitioning of the state of a software component into independently-lockable sub-structures.

The basic fault treatment administrative facilities outlined above are supplemented by facilities for configuration and maintenance management.

Configuration management facilities include remote initialization and loading of nodes, node passivation, version control, etc. These functions are necessary to assist in normal system evolution.

Maintenance management facilities are provided to minimize repair time. Such facilities include tools for post mortem analysis, remote access to the host's local operating system diagnostics and support for remote initialization or loading of off-line tests. In an ideal case, the maintenance staff is provided with information concerning failed boards (in stations) and failed transmission medium segments.

8.2.2. Structuring Concepts

This section presents a general structuring model for designing management systems (see §8.2.2.1) and its application to Delta-4. The model is supplemented by an enumeration of features of a formal management description language (see §8.2.2.2) that enables a specification of a management system structured according to the model's design guide-lines. The derivation of general architectural components from the model is given (see §8.2.2.3) and this forms the basis for the architecture of the Delta-4 management system (see §8.2.3). The relationship between the object model and the management of distributed applications is used as an example of the model's application (see §8.2.2.4).

8.2.2.1. Structuring Model. The inherent complexity of the various administrative tasks outlined in the previous section requires the development of a structuring model. This model should provide unified design principles and support a stepwise implementation strategy aimed to conform and coexist with related (draft) standards. This model should provide an abstraction of the system to be managed that is based on the viewpoint of a system administrator and which allows the system to be viewed coherently despite its complexity. The proposed model encompasses the following structuring principles:

1) manageable components;
2) management domains (or simply "domains").

8.2.2.1.1. Manageable Components. The basis for the model is to treat a system as consisting of a set of (interacting) typed components. A manageable system consists of a set of manageable components. These are an abstraction of normal components of some granularity with extensions relevant to management. In addition to the designed normal functionality, manageable components are characterized by their:

- static and dynamic management-related attributes such as version identification, operational state and parameters, error and performance-related statistical information, designed fault-tolerance behaviour, various management relationships to other manageable components etc.; and
- performable management operations such as create, delete, clone, reset, change state, get/set attributes or wait for events triggered by the manageable component.

Note: In the ISO/OSI Management standards, an abstraction of a physical or logical resource for managing this resource is called a managed object. An ISO managed object is defined by:

- the attributes visible at the managed object boundary;
- the management operations that can be applied to the attributes or the managed object itself;
- the behaviour exhibited by the managed object in response to management operations; and
- the notifications that can be emitted by the managed object.

A Delta-4 manageable component may be described as an ISO managed object. Thus, when the term managed object is used in the following to describe the management view of a Delta-4 resource, the ISO management view is applied with the above described properties.

Examples of manageable components in Delta-4 are stations, communication objects or software components. A survey of manageable components that are currently under implementation is given in §8.2.4. Note that software components (capsules) generated from Deltase objects form one type of manageable component (this is further illustrated in §8.2.2.4).

8.2.2.1.2. Domains. In the literature various definitions of a domain can be found. In [Sloman 1987] a domain identifies a sphere of influence of management. A more concrete definition can be found in [Robinson 1988]: here a domain identifies a "set of managed objects that share a common attribute; in particular, it is the set of managed objects to which the same management policy applies".

A common attribute of replicated objects in Delta-4 is that replicas can be dynamically placed on a given set of stations. Such a set of stations is termed a *replication domain*. Common management policies are applied to sets of replicated objects that have the same replication domain, for instance:

- restoration of the replication degree of objects which had a replica on a failed station by cloning them to stations offering spare redundancy; and
- migration of all objects from a station that is due for maintenance.

Various boundaries of domains can be considered, e.g., physical, organizational or security boundaries. For example, in a distributed system the components which conform to a communication standard form a domain. Manageable components encapsulating management information accessible through management operations may also be seen as a domain.

As illustrated in [Sloman 1987], it is possible to conceive of four relationships between domains: they can either be disjoint, interacting, overlapping or nested. These relationships indicate how the management entities that are responsible for the management of respective domains must cooperate to fulfil a common management policy.

8.2.2.2. Management Description Language (MDL). A management description language (MDL) can serve as a basis for a formal description of the structure and the functionality of a management system. In Delta-4, the development of such a language is under way; it should enable a system administration designer to specify formally the administration policy of a Delta-4 system.

The following language requirements have presently been identified:

- facilities to specify the domains and their interrelationships;
- facilities to specify the manageable components and their integrated management mechanisms;
- facilities to specify the transparency attributes of management services;
- facilities to specify decisions about the invocation of management services;
- facilities to specify possible system states — the system state is necessary to specify the conditions on which the above-mentioned decisions are based.

The current emphasis of our work on MDL is on the specification of the structure of the management system and the decision taking mechanisms. Structuring elements are derived from the structuring model given in the previous section; decisions are based on system monitoring.

The architectural components that make use of these specifications are not included in the language themselves. A classification of architectural components is given in the next section. A complete architecture includes more than this; it encompasses the set of architectural components as well as the set of relationships between these components according to a set of rules for their establishment [ANSA 1989]. The architecture of the Delta-4 management system is given in §8.2.3. Modelling the architecture in MDL is currently not an area of investigation.

8.2.2.3. Architectural Components. Having defined the structuring concepts and specification techniques, guide-lines for the development of the management system architecture can now be established.

Architectural components of a management system can be divided into two major classes:

1) domain managers;
2) (domain) manager applications.

8.2.2.3.1. Domain Managers. A domain manager can decide about, and carry out, management operations on the set of managed objects that constitute a domain.

A domain manager comprises a single domain manager process, or a set of peer domain manager processes that cooperate within the domain to carry out a domain-specific management task. A domain manager process may be replicated to meet the Delta-4 dependability requirements. However, the various interrelationships and interdependencies of domains mean that not all management tasks can be performed within a specific domain. Thus, inter-domain interaction is needed between domain manager processes across domain boundaries as well as intra-domain interaction between peer domain manager processes. Cooperation is carried out by a sequence of management interactions. For a particular interaction, a domain manager process may take either the role of an agent or a manager:

- A domain manager process taking the role of an *agent* (shortly: an agent) is responsible for performing the management operations upon managed objects within its domain, and for forwarding event notifications triggered by managed objects to a manager.
- A domain manager process taking the role of a manager (shortly: a *manager*) conveys management operations on managed objects to the responsible agent, and receives event notifications from agents.

Manager and agent roles are either statically or dynamically assigned to domain manager processes. In a static role assignment, one domain manager process takes the agent role and the other takes the manager role during the whole domain manager cooperation. In a dynamic role assignment, a domain manager process may take the agent role in one interaction and the manager role in a separate interaction during the period of the domain manager cooperation. The role assignment is dependent on the given relationship of the domains (figure 9) that the domain manager processes are part of:

a) For intra-domain cooperation between peer domain manager processes the roles are assigned dynamically (e.g., SMAP-SMAP cooperation, see §8.2.4.3.4).

b) There is no cooperation at all between the domain manager processes of two disjoint domains.

c) For cooperation between domain manager processes of two interacting domains the roles are assigned dynamically.

d) Two overlapping domains *A* and *B* have a domain intersection *AB* in which a common management task is to be performed. There are (at least) two approaches to manage the managed objects within *AB*:

   1. *AB* is itself regarded as a new domain. In this case, the cooperation between the domain manager processes that are responsible for the domain intersection is an intra-domain cooperation with dynamically assigned roles (cf. (a) above).
   2. Instead of regarding *AB* as a new domain, a new higher-level domain *C* is built that contains domain *A* as well as domain *B* (thus forming two nested domains, cf. (c) above). To synchronize management operations on the managed objects within *AB*, cooperation between the domain manager processes of domain *C* and those of *A* and *B* is necessary.

In such interactions, the domain manager processes of *C* always take the manager role, whereas those of *A* and *B* take the agent role. There is no direct cooperation between the domain manager processes of *A* and *B*.

e) Domain manager processes residing within nested domains interact using static role assignment where the domain manager processes of the containing domain take the manager role, and those of the contained domain take the agent role.

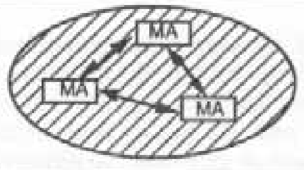
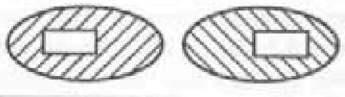
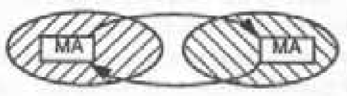
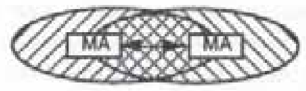
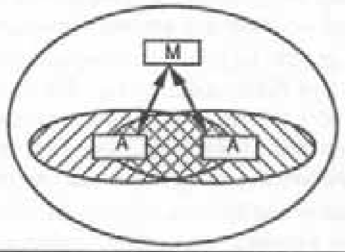
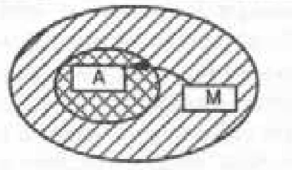
Domain Relation	Role Assignment	Illustration
Intra-domain		
...	Dynamic	
Inter-domain		
Disjoint	...	
Interacting	Dynamic	
Overlapping 1. with domain intersection = new domain	Dynamic	
2. with new containing domain	Static	
Nested	Static	
▭ domain manager process	[A] agent	⬭ domain
[M] manager	[MA] manager/agent	⟷ cooperation

Fig. 9 - Manager/agent Role Assignment Dependent on Domain-Relationships

8.2.2.3.2. (Domain) Manager Applications. As outlined in the previous section, domain managers may not be able to carry out a particular (higher-level) common management policy without cooperating with domain managers of other domains. If the domain relationship between two domains *M* and *A* is of type "nested", and *M* contains *A*, then the domain manager of *M* is said to be a *"(Domain) Manager Application"* of the domain manager *A*. In this case, the domain manager processes of *M* always take the manager role when interacting with domain manager processes of *A*. Thus, the management policy is guided and synchronized by the (domain) manager application with the support of its agents in a containing domain. Note that this definition of a Manager Application is a generalization of the basic architectural components defined in [MAP] ("agent", "manager", "manager application") or [CNMA].

8.2.2.4. Example: Management of Distributed Applications. Figure 10 integrates the structuring concepts, the specification techniques and the generic and invariant architectural components that apply to distributed applications and their support environment (Deltase) on one side, with the management of distributed systems on the other. In this section, the management of distributed applications serves as a case study for illustrating the concepts introduced in the previous sections. This will be continued on the architectural level in §8.2.3.

On the "administrator side", figure 10 represents an overview of what has been introduced in sections 8.2.2.1 - 8.2.2.3. On the modelling level, the object model and its application for distributed systems has been introduced in chapter 7. Object-oriented programming is characterized by the concepts of data typing, data encapsulation and the inheritance of certain properties of typed objects to particular instances of them. On the specification level, the types and instantiation mechanisms can be specified in an Object Description Language (ODL), along with the parallel entities (threads) and the interfaces of objects. One of these interfaces is the "management interface" (see §8.2.3.2) that exports the management operations applicable to an object. This is related to the management description language (MDL) where software components (capsules) generated from objects can be described as manageable components (note that hardware components can also be described as manageable components).

While ODL scripts represent the "normal" functionality of a distributed system by describing the distributed computation, MDL scripts represent its administrative view by describing the integrated automatic management functions.

The generic architectural components shown in figure 10 may be generated from the scripts in ODL and MDL respectively. They make use of the invariant components that form the nucleus of a system. The software components that form an application are divided into an object part and an object envelope part. One part of the envelope is the Object Manager Entity (OME) performing the operations of the management interface.

Creation and deletion of managed objects are performed by a *domain manager*. The create operation supplies management information about the object to the domain manager. The domain manager is assisted by local factories that generate new object instances from a template. Local factories are also responsible for terminating object instances.

The cooperation of all the architectural components shown in figure 10 is necessary to perform certain management functions. The architecture in section 8.2.3 describes how the management information contained in these components (referred to as the "Management Information Base" (MIB)) is exchanged to perform management functions.

Both of the models illustrated in figure 10 are generic enough to incorporate the other as one particular instance. As the management system itself is a distributed application, its architectural components may be realized as objects; they could also be integrated into particular software components that form domains for them.

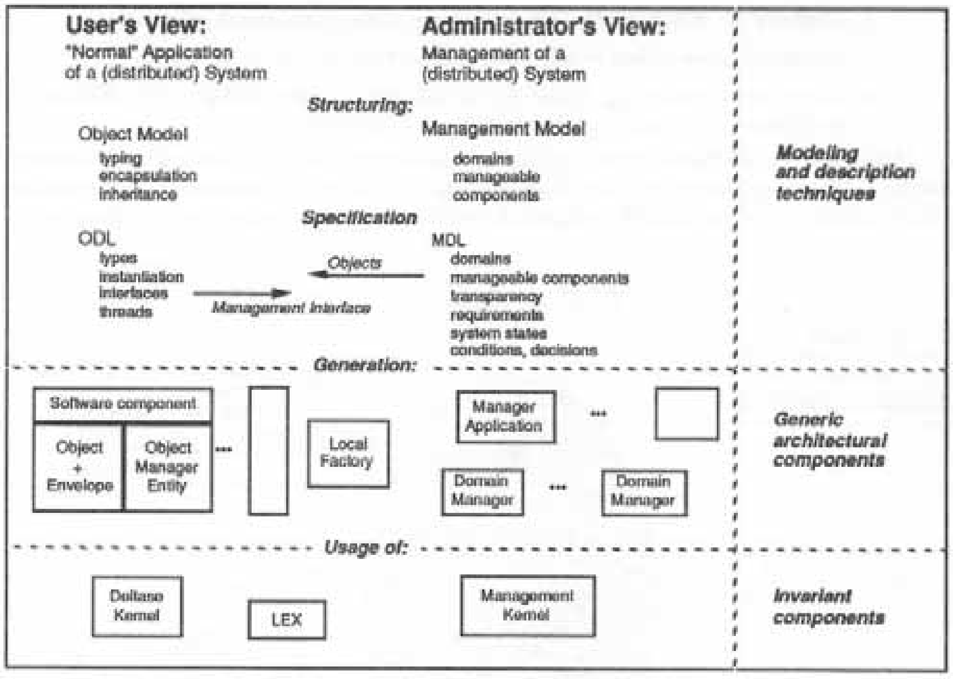


Fig. 10 - User's and Administrator's View of a Delta-4 System

8.2.3. Management System Architecture

The design of the Delta-4 OSA management system architecture in this section is described by:

- specifying the manageable components considered in Delta-4 OSA, including the management information, i.e., the representation of the attributes contained within the manageable components, and the management interface to the manageable components;
- specifying the domains considered in Delta-4 OSA, the respective domain managers, the management information contained in domain managers, the services and protocols for the exchange of management information, and the management services provided by domain managers;
- specifying the cooperation between manager applications, domain managers and manageable components that is necessary to perform management tasks.

8.2.3.1. Template Architecture. The generic architectural components of a management system as defined in section 8.2.2.3 must be correlated to get an architecture of the management system. These correlations are expressed in terms of service interfaces and cooperation types.

A manageable system consists of a set of *manageable components*. Manageable components are characterized by the fact that they possess integrated mechanisms for management purposes (figure 11). These must be specified and implemented in addition to the "normal" functionality of system components. These *integrated management mechanisms* comprise attributes, operations and events:

- *attributes* are represented by component integrated management information;
- *operations* allow access to and manipulation of component attributes;
- *events* are a means by which the component delivers management information asynchronously (events are a special form of attribute).

Operations are offered to domain managers by a *manager entity* at a dedicated *management interface*. This management interface forms the functional division of management mechanisms that are specific to the manageable component and domain-specific management mechanisms.

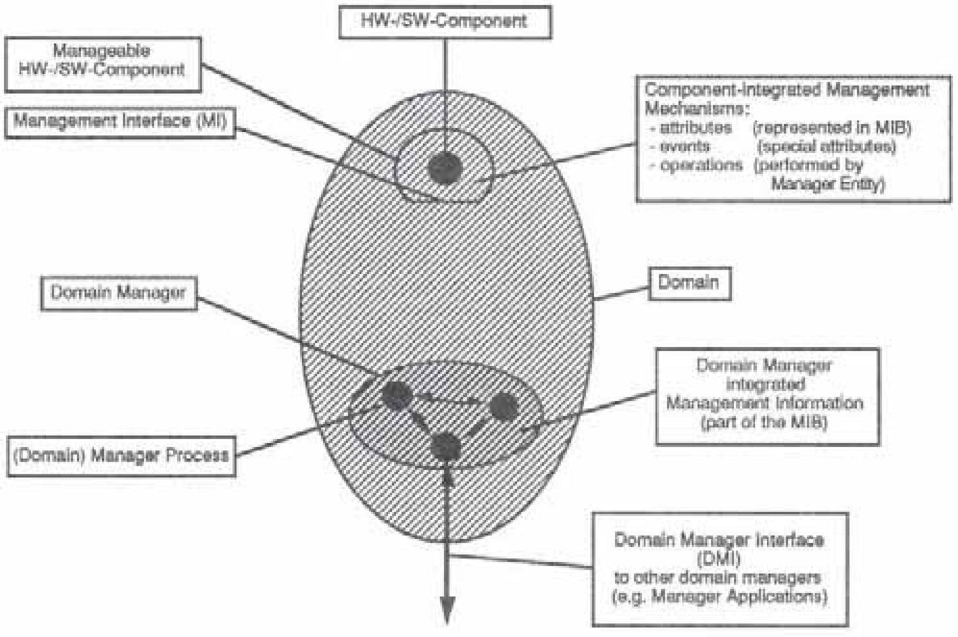

HW-/SW-Component
Manageable HW-/SW-Component
Management Interface (MI)
Component-Integrated Management Mechanisms:
- attributes (represented in MIB)
- events (special attributes)
- operations (performed by Manager Entity)
Domain Manager
Domain
Domain Manager integrated Management Information (part of the MIB)
(Domain) Manager Process
Domain Manager Interface (DMI) to other domain managers (e.g. Manager Applications)

Fig. 11 - Manageable Components, Domain Managers and their Interfaces

The management tasks that are dedicated to a management domain are performed by a domain manager and may be categorized according to the five functional management areas (cf. §8.2.1.1). These tasks are offered to other domain managers at dedicated Domain Manager Interfaces (DMI) in form of management services:

- DMI_FM: DMI for fault management services;
- DMI_CM: DMI for configuration management services;
- DMI_PM: DMI for performance management services;
- DMI_AM: DMI for accounting management services;
- DMI_SM: DMI for security management services.

MCS supports these functional-area-specific management services in form of the Multipoint Common Management Information Service Element (M-CMISE). M-CMISE is part of the MCS application layer and offers basic multi-point services for the exchange of all types of management information between domain managers. The M-CMISE services enable the access to management information that is either encapsulated within the domain managers themselves or within the manageable components. In addition, they allow a domain manager to request other domain managers to perform a management operation upon manageable components and to send event notifications triggered by manageable components to other domain managers. As the M-CMISE services are mapped upon the MCS Session services, an M-CMISE service-user may also benefit from the multicast facilities of MCS. In ISO Systems Management, corresponding management services for the ISO/OSI bi-point environment are standardized in the application service element CMISE [ISO 9595].

Management information residing within the domain manager is information about the manageable component, e.g., the minimum, the current and the maximum number of replicas of the manageable component. On the other hand, management information integrated within the manageable component itself is information that is closely related to the "normal" functionality of the component, e.g., the context of a file in the case of a file server. This type of management information can be automatically generated from a dedicated MDL description (cf. §8.2.2.2) Both kinds of management information are conceptually summarised under the term *Management Information Base* (MIB) which is thus per definition distributed.

A *domain manager* may consist of a set of cooperating *domain manager processes*. Cooperation is needed for instance:

- to exchange management information between domain manager processes;
- to fulfil a common management task that was requested at a domain manager interface by a manager application; or
- to execute a distributed control or consensus algorithm.

8.2.3.2. Manageable Components

8.2.3.2.1. Objects. Objects that behave according to the strong object model, as described in chapter 7, encapsulate abstract data structures that can only be accessed and affected by operations via defined object interface(s).

This section discusses general aspects about managed objects as one type of manageable component. In subsequent sections these are applied to certain objects investigated in Delta-4.

a) Objects as Manageable Components

To be manageable, an object must have an *Object Management Interface* (OMI) at which management specific operations can be invoked. These allow access to *management information* that is encapsulated as object data within the object. The functional part within the object that performs the management operations is called an *Object Manager Entity* (OME). Thus the OMI and the OME, together with the object-integrated management information, comprise the additional part of an object that is necessary for the object to be manageable. Under the management view this kind of object is called a *managed object*.

b) Object-Integrated Management Information

Items of object-integrated management information are attributes that define static or dynamic properties of the object. The "object version identifier" is one example of a static object attribute. There are various types of dynamic attributes, e.g., counters, gauges, state information, logs, events, etc. Counters can be used for instance to indicate the number of invocations of "normal" object operations. Events can be seen as "special" attributes that are asynchronously generated by the OME if a defined object-internal situation occurs. They might be reported to the domain manager automatically or only following an explicit enquiry.

c) Interface to Domain Managers

The variety and complexity of management tasks require uniform basic architectural management mechanisms and a clear mapping of management functions onto the components of the management architecture. This is the major motivation for the identification of the *Object Management Interface* (OMI). It is situated between the *domain manager* and the (replicated) *managed object*. The OMI should provide a clear division between those management functions that are integrated within the managed object (and performed by the *object manager entity*), and those that are carried out by the domain manager. This functional division should be made both independently of the object type to be managed and of the domain manager type. As an object may reside within more than one domain, the OMI may be imported by several domain managers.

The OMI offers a variety of management operations to be performed on the object. On the one hand, their semantics should be generic enough to keep the number of management operations small, on the other hand they should cover all object types and their associated management information. Management operations are of two kinds: those which can be sent to an object to be applied to its attributes, and those which apply directly to an object itself.

Examples of the former are "get attribute" to read the current value of an attribute, or "set attribute" to modify the value of an attribute.

Examples of the latter are "create object" to direct the object manager entity to create an object, or "test object" to direct the OME to perform a specified test operation on the object.

8.2.3.2.2. Objects at the Application Level. The architectural approach is applied to the following objects on the application level:

- *Processes:* Processes form the basis on which (existing) applications are built. In the present Delta-4 implementation, a Deltase capsule is represented by a UNIX process.
- *Files:* Global files may be useful in certain applications, so a server for managing replicated global files also been implemented.

For these components of different nature cloning techniques are investigated. Cloning of higher level objects, for instance file servers or databases, may be based on the cloning mechanisms of these components.

a) Application Objects as Manageable Components

For specifying the management of application objects the following approach is taken:

- the application objects are modelled as objects according to the strong object model;
- to describe these objects as managed objects, their integrated management information and the *object management interface* are specified.

The emphasis placed on the investigation of application objects as manageable components is to relinquish modifications in the *Local Executive Environment* (LEX). A precondition of this is that the object-integrated management information, which is obtained through services offered by the LEX, is sufficient to make the object manageable.

b) Application Object - Integrated Management Information

Management information related to application objects is represented in two ways:

- The *LEX* context: this is information that represents the state of the application object in a LEX-dependent way. This information may either be part of the application object (e.g., the loaded code of a process or the content of a file), or it may be established by the LEX when operations are applied to the application object (e.g., object descriptors). This information is not held within the application object, but is kept within the LEX, whence the term the "LEX context" of an application object.

- The *global* context: this is information that represents the state of the application object independent from the LEX. To generate this information, the LEX context must be transformed into an abstract notation. This is a precondition for domain managers to be able to interpret the context. If an application object is replicated, the global context is the same for all replicas, whereas the LEX context may have a different representation for different replicas. At cloning, the global context is transferred in voted packets over the network to the location of the new replica, where the LEX context is to be reconstructed from the global context.

c) Interface to Domain Managers

In Delta-4, domain managers are primarily investigated for the management of replication. A *replication domain manager* manages a set of objects; it determines the stations on which replicas of these objects should reside. To execute management tasks, it is assisted by *Object Manager Entities* (OMEs), which perform management operations on a particular object that is part of the domain. These operations are invoked by the domain manager. The assignment of management tasks to either the OME or the domain manager in Delta-4 is such that the OMEs:

- assist the domain manager in instantiation and termination of a replicated application object; and
- perform instantiation of new replicas (i.e., cloning) and termination of a replica on request from the domain manager.

Thus, the OMEs maintain both the LEX context and the global context of an application object; at cloning time, they transfer the global context to the location of the new replica.

8.2.3.2.3. Communication Object Management. The management of communication systems and their communication objects is subject to standardization by ISO under the term OSI management. The concepts and the terminology that are elaborated in the OSI Management Framework [ISO 7498-4] and the Systems Management Overview [ISO 10040] fit naturally into the generic Delta-4 management model.

a) Communication Objects as Manageable Components

The management view of a communication system abstracts from the "normal" communication facilities and concentrates on those communication objects that need management support, and/or can supply management information. These communication objects are, for example, layer entities, protocol state machines, connections, service access points, or representations of physical devices.

To become manageable, communication objects are extended as described in §8.2.3.2.1 (a), i.e., management operations are invoked at a dedicated management interface to affect the communication object and its integrated management information.

Applying the OSI management terminology, a communication object that is regarded with such a management view is called a *managed object*. Instances of managed MCP objects that share the same management operations, attributes and notifications are said to be of the same *managed object class* [ISO 10165-1]. The set of managed object classes that have been defined for the MCP management is hierarchically organized. This hierarchy results in a so-called *registration tree* that is used for the identification of managed object classes. Figure 12 shows the current Delta-4 registration tree. One particular managed object class is identified by a path within this tree, e.g., the "MCP Session Endpoint" is reached by the path "Delta-4 1 5 2 1".

```
Delta-4
 |----- 1 ---------Network Management
        |---------1 -------- Layer
        |         |-------- 1 ---------Physical
        |         |         |---------1---- ISO 8802.4  (Token Bus)
        |         |         |---------2---- ISO 8802.5  (Token Ring)
        |         |-------- 2 ---------Data link
        |         |         |---------1---------LLC
        |         |         |         |--------- 4 ----MCP LLC_xAMp
        |         |         |                    |--------- 1 ----Gate
        |         |         |---------2---- MAC sub-layer
        |         |         |         |--------- 4 ----ISO 8802.4  (Token Bus)
        |         |         |         |--------- 5 ----ISO 8802.5  (Token Ring)
        |         |         |         |--------- 7 ----Abstract Network
        |         |         |         |--------- 8 ----Turbo-AMp  (Token Ring)
        |         |-------- 4 ---------Transport
        |         |         |---------3---------MCP Transport
        |         |-------- 5 ---------Session
        |         |         |---------1---- MCP Session
        |         |         |---------2---- MCP Inter-Replica protocol
        |         |         |         |--------- 1 ----MCP Session Endpoint
        |         |         |         |--------- 2 ----MCP Session Connection
        |         |-------- 7 ---------Application
        |                   |---------4---- MCP ACSE
        |---------2 -------- System
                  |-------- 4 ---------Service Access Point
                            |---------7---- M-SAP
```

Fig. 12 - Delta-4 Managed Object Class Registration Tree

The current Delta-4 MCP managed object definitions, together with their attributes, events and actions, are specified within the MCP protocol specifications. Delta-4 *communication object management* has to manage MCP-specific communication objects to support fault-tolerant applications. Examples are:

- Delta-4 Multicast Service Access Points (M-SAPs), and
- Delta-4 Multipoint Associations.

These types of MCP communication objects are represented by two different managed objects. The first managed object represents the local-to-station management view of this MCP communication object. This managed object is locally kept on the station where the communication object exists. It contains, for example, management information that represents the local-to-station usage of the communication object. The second managed object represents pre-defined management information that is of global significance. The access to this managed object is essential for the instantiation of new instances of the communication object. It is therefore stored within the (replicated) "Global-MIB" that is itself a manageable component (cf. §8.2.3.2.4).

b) Communication Object Integrated Management Information

Management information that is integrated within managed communication objects is closely related to the communication activities of the communication objects. It consists of those types that are identified within §8.2.3.2.1 (b) and is specified in an abstract notation within a *managed object definition*. ISO is proposing such a notation in [ISO 10165-4].

Examples of Delta-4-specific management information within MCP are the attributes "number of session Protocol Data Units (PDUs) sent/received" of the managed object "MCP Session Entity", and the event "newStation" that is automatically generated by the SMAP (see below) when a new station has been attached to the network.

c) Interface to Domain Managers

All management operations upon layer-specific managed objects and their associated management information are performed by a *Layer Management Entity* (LME) that exists within each MCP layer. The interface type to this LME is the same for all layers and corresponds to a subset of the standardized object management interface as described in the management architecture. The domain manager process that accesses the communication object integrated management information through this interface is called the Systems Management Application Process (SMAP). The SMAP is described in section 8.2.4.3.1.

8.2.3.2.4. Global-MIB Management. The template architecture in figure 11 distinguishes two types of management information:

a) component integrated management information;

b) domain manager integrated management information.

There are several reasons why parts of the management information of type a) in real implementations are not encapsulated by the component; the LEX context of application objects is an example. In some cases, a practical solution can be to keep such management information within the domain manager.

As further explained in the implementation section (§8.2.4), there are also reasons for which parts of the management information of type b) are not encapsulated by the domain manager.

Management information that (in the implementation) is neither encapsulated by a component nor by a domain manager is assumed to be stored in an object called the *Global-MIB*. Conceptually, it is still seen as information of type a) or b) respectively.

a) The Global-MIB as a Manageable Component

As other objects, the Global-MIB is a type of manageable component and offers an *Object Management Interface* (OMI) to domain managers (see (c) below).

b) Global-MIB-Integrated Management Information

The statements of §8.2.3.2.1 (b) apply unchanged. As the Global-MIB needs management support (e.g., enhanced dependability), it has integrated management information. Two special situations may occur in practice:

1) The Global-MIB's integrated management information (i.e., information of type a)) is stored in the Global-MIB itself. It can thus be accessed both via OMI and via normal object interfaces.

2) The Global-MIB is managed by a domain manager that also has its integrated management information (i.e., information of type b)) stored in the Global-MIB.

c) Interface to Domain Managers

The statements of §8.2.3.2.1 (c) apply unchanged for the *Global-MIB* object. The functionality of the standardized OMI is sufficient to handle the two situations described above concerning the storage location of management information as well. However, there are further requirements for the design of domain managers managing the Global-MIB over and above those for domain managers concerned with ordinary objects. The domain primarily investigated is a replication domain for the Global-MIB (cf. §8.2.3.3).

8.2.3.3. Domains

8.2.3.3.1. Replication Domains. A *replication domain* of an instance of a software component is the set of stations on which replicas of that instance may reside. A common management policy is applied to a set of software components that have the same replication domain. The architectural component that executes this policy is called the *replication domain manager*. It reacts on changes of the *station* or *node group* (cf. §6.8) that constitutes the replication domain, for instance, due to station failure or shut-down. It reconfigures *replica groups*, for instance, by cloning software components that had a replica on a failed station to another station offering spare redundancy. The reconfiguration strategy is not only dependent on this particular software component, but also on the *group of software components* that execute in the replication domain.

a) Domain Manager - Integrated Management Information

Domain specific management information is mainly the reconfiguration strategy and information about the objects and the stations of the domain that is necessary to execute the strategy, e.g.:

- the minimum, the current and the maximum number of replicas of an object;
- the desired and current location of replicas;
- the assigned communication objects, etc.;
- the maximum and current load of stations;
- the current state of stations (up or down, attached to the network), etc.

The update of this management information is in the responsibility of the manager of the communication domain (see next section); as the communication domain manager keeps this information in the Global-MIB, it can be retrieved from there. Significant changes of management information (e.g., station failure) are reported to the domain manager by spontaneous events.

b) Interface to Manager Applications

The replication domain manager offers a set of operations to its manager applications. Two different kinds of management operations can be distinguished:

- generic domain operations, independent of the domain type, e.g.:
  - *create/delete managed objects:* these services create/delete entries for objects in the domain managers information base;
  - *instantiate/terminate managed objects:* for instance, process instantiation/termination;
  - *list managed objects:* this service allows information about managed objects of the domain to be retrieved;
- operations that are specific to a replication domain, e.g.:

- *reconfiguration strategy:* this service supplies the domain manager with the reconfiguration strategy it should adopt for a subset of objects, which is to be applied if a certain situation should occur (e.g., station failure, station shut-down, etc.).

8.2.3.3.2. Interaction Manageable Component — Domain Manager. To perform its services, the domain manager makes use of the operations offered at the OMI of each application object.

At instantiation/termination of a process, the domain manager may be supported by the OME, which for instance performs some initialization and notifies the domain manager that all replicas have been instantiated/terminated successfully.

During cloning, the domain manager and the OME interact in the following way:

- Having decided that an object is to be cloned, the domain manager first instantiates a template of the object (and an OME for it) at the new location.
- When the new OME indicates that the (template) instantiation is complete, the domain manager instructs the OME of the already existing replicas to build and transfer the global context to the new OME.
- The new OME substitutes the current local context of the object with the one resulting from the received global context. It synchronizes the input and output message streams of the new replica with those of the existing replicas, while ensuring replica consistency. Having performed this sub-operation, the instantiated template has become a new replica, and the OMEs are now also replicas of each other.
- The OME responds to the domain manager to indicate that the context transfer is complete.

8.2.4. Implementation

A management system whose design is described in §8.2.3 is currently under implementation. This section gives a description of the architectural components and the managed objects chosen for that implementation.

8.2.4.1. A Replication Domain for UNIX System V Processes. The implementation comprises a prototype for the management of UNIX System V processes. Figure 13 illustrates the architectural components involved and how they interact.

Two replicated processes A and B are shown, which interact via an MCS (multi- or bi-point) association (other interaction, e.g., local inter-process communication, is disallowed to ensure replica determinism). These processes may include the Deltase run-time support system (Deltase/XEQ) which makes use of the MCS services offered on the host.

To make processes manageable, they include an *Object Manager Entity* (OME) as a library. The invocations of the MCS services are passed through the OME, which is thus able to control the complete input and output message stream of the process. Domain management is implemented by a *Replication Domain Manager* (RDM), which consists of one replicated domain manager process. Cooperation between RDM and OME is via a specific association, the RDM association.

To perform the instantiation/termination service, the RDM invokes via the RDM association the services offered by the *factories*. A *factory* is a manager of a domain nested within the replication domain; it is implemented as a non-replicated entity present on each host on which a process of the replication domain may have a replica; it uses the local UNIX fork/exec directives to instantiate a process.

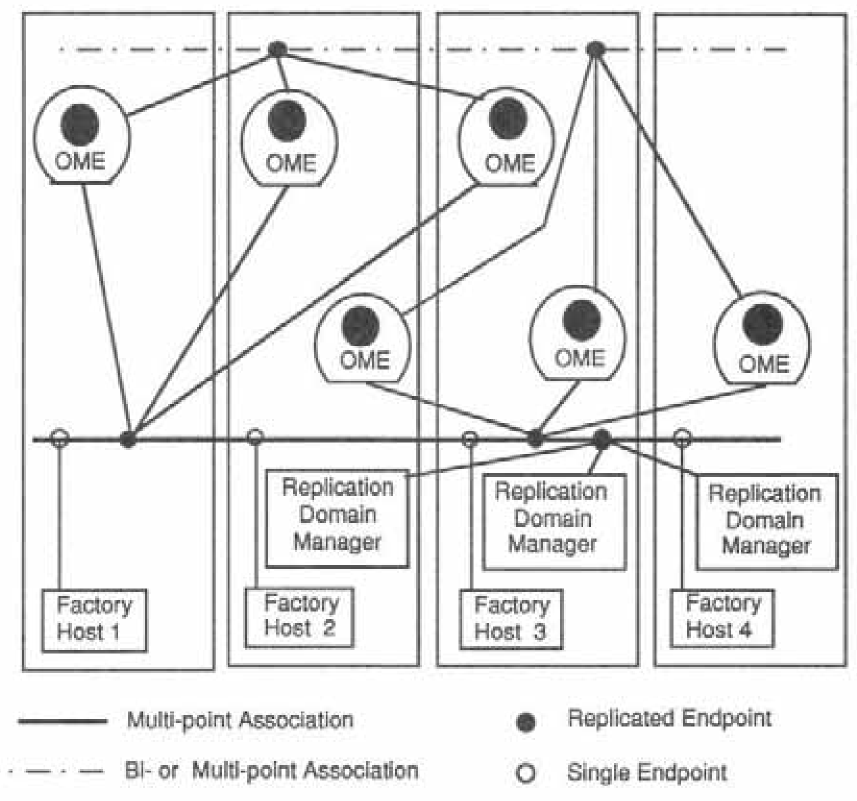


Fig. 13 - Architectural Components for Process Management

8.2.4.1.1. The Replication Domain Manager (RDM). The Replication Domain Manager offers the generic services as described in section 8.2.3.3.1:

- Create Managed Objects;
- Instantiate/Terminate Managed Objects; and
- List Managed Objects.

There are additional services that are specific to a replication domain, e.g., the reconfiguration strategy service mentioned in section 8.2.3.3.1. A precondition for such a service is the definition of a Management Description Language (MDL, cf. §8.2.2.2), which enables the description of reconfiguration strategies. Certain strategies are directly implemented within the RDM.

An example of such a strategy is the RDM's reaction to station failure and station reentrance. Recognition of station failure/reentrance is in the scope of the *Communication Domain Manager* (CDM, cf. §8.2.3.3.2). If such an event occurs, the replication domain manager is notified. At station failure, it searches in its information base for processes that had a replica on the failed station, and updates the value of the attribute that determines the actual replication degree of this process. If the desired replication degree of a process (an attribute set by the Create Managed Objects service) is higher than the actual replication degree, the RDM searches for a "free" station, i.e., a station that does not hold a replica of this process. It instructs the local factory on that station to instantiate a template of the process, including an OME, and starts the cloning service provided by the OME. At station reentrance, the RDM again searches for processes that have fewer replicas than desired and instantiates replicas on the reentered station. In addition, all replicas that had earlier been cloned to another station, are moved back to the reentered station.

As the RDM itself must also be dependable, it is implemented as a replicated object. When a station fails, not only the application objects' degrees of replication are to be reestablished, but also that of the RDM. Thus, the RDM has a built-in OME that it instructs to clone itself.

8.2.4.1.2. The Object Manager Entity (OME). The *Object Manager Entity* (OME) offers services at the *Object Manager Interface* (OMI) to the *Replication Domain Manager* (RDM):

- it assists the RDM in instantiation and termination of a replicated process;
- it performs instantiation of new replicas (i.e., cloning) and termination of a replica on request from the RDM.

The OME library not only supports cloning of actively replicated processes. Cloning is also useful for semi-active and passive replicas, for instance when a station is reinserted a new leader or primary may be required. However, the description here concentrates on cloning of active replicas.

As illustrated in figure 13, in a replicated process, each replica contains an OME library. The process and its OME use one replicated M-SAP. The process establishes endpoints on that M-SAP in order to communicate with other processes; in addition, the OME establishes an endpoint to the RDM association.

The OME acts as a replicated instance whose Finite State Machine (FSM) is always in the same state in each replica. Incoming events are generated:

a) when the process reaches an MCS communication statement (send/receive a message, establish/release an endpoint); and
b) when the OME receives a message, which is only possible when the process has reached a receive statement.

To preserve replica consistency, all incoming events are ordered identically at all replicas of the OME.

Outgoing events of the FSM depend on the current state. For instance, if the OME is executing a cloning protocol with several steps of context transfer, the process may reach a communication statement between two transfer steps, which may lead to a phase in which the process must be blocked.

The phases of the cloning protocol described below start from the assumption that the DM has created a template of the process with an OME library at the new location.

The new process's OME establishes the same M-SAP on the new station and receives all messages on the DM Association that the existing process/OME receives, but the new OME's FSM does not generate the same outgoing events. The RDM invokes the Instantiate Replica service of the OME.

Phase 1: *Reestablish Communication Context*

The context of all endpoints the process has currently established is transferred to the new location. The new OME re-connects to all associations on which the process has endpoints.

Finally, the input message streams of the existing replicas and the new replica are synchronized. Should the process try to establish/release an endpoint during further phases of the protocol, it is blocked.

Phase 2: *Take a Snapshot of the Computational Context:*
The LEX (UNIX) context of a process comprises:

- the data segment (initialized, uninitialized and dynamically allocated data);
- the stack segment, including the UNIX environment;
- the process registers.

The UNIX context is obtained by normal system calls. Modifications of the operating system kernel and libraries were not required.

Unlike checkpointing with passive replicas under a fail-silent regime (cf. section 6.6), cloning with active replicas requires that the snapshot be taken at the same execution checkpoint by each active replica. Furthermore, since the active replication technique is applied in a fail-uncontrolled environment, the context transfer messages must be able to be voted. This means that the context data in each active replica must be made bitwise identical; this is carried out in phase 3.

Phase 3: *Generate the Global Context:*
The system libraries (UNIX-, MCS-, Deltase- and OME libraries) may use local services that generate values that may be different between replicas. For example, local file descriptors, mailbox identifiers, context identifiers significant on the NAC, etc., can be found in the local context that are different for each replica. The OME builds the global context for which all local descriptors are equal in each replica, which is a precondition for starting phase 4.

Currently, application programmers must not use local resources for inter-process communication (signals, semaphores, pipes, shared memory, mailboxes). Only global files may be used (cf. §8.2.4.2). A reduction of these restrictions is planned such that local files may be used by the application.

Phase 4: *Transfer the Global Context:*
The global context is transferred in voted packets. During the transfer steps, the currently active replicas are allowed to continue execution; the new replica then avoids sending messages that have already been sent in the context transfer phase.

Phase 5: *Continuation at the Execution Checkpoint:*
Having received all context packets, the computational context of the new replica is substituted by the transferred context. The local resources needed by the communication and management entities (see Phase 3) are then established. The process now continues at the execution point at which the snapshot was taken. All replicas now synchronize their output message stream, i.e., the new replica is built and the RDM is informed.

8.2.4.2. A Replication Domain for UNIX System V Files. The architectural design outlined in section 8.2.3 has also been validated by a prototype for the management of replicated UNIX V Files. As in the case of the prototype for UNIX System V processes, the implementation architecture does not imply modifications to the UNIX kernel or libraries; the local UNIX file servers are used unchanged.

8.2.4.2.1. Management Services. As opposed to processes, files do not execute any computation and do not interact with each other. Operations for local file management (open/close, read/write, etc.) have defined semantics and are performed by a local file server.

To enable replication of files, the following support is provided:

- *Global file server:* this offers a subset of the UNIX System V file management operations. Applications using the global file server send requests for file operations as messages; if a requesting application is replicated, then its request-messages are voted.
- *Domain management:* The following generic domain management operations are provided:
  - *Create/delete managed object:* This delivers management attributes (e.g., degree of replication) to the RDM.
  - *List managed objects:* Instantiation/termination services are not provided; it is assumed that local copies of a file are present at each station as soon as the *create* service is invoked. The reconfiguration strategy for station failure/reentrance described in the previous section is also applied to files.
- *Object management:* Instantiation/termination services for new replicas of a file are provided.

8.2.4.2.2. Implementation. The functionality described above is implemented by a software component residing on each station of the replication domain. This software component is termed the *File Manager* (FM). The file manager offers service access by way of its *FM-Association* and comprises several sub-components, which are described below.

FM - Domain Manager (FM-DM)

This offers the domain management services listed above. It consists of one RDM Process with as many replicas as the replication domain has stations. A manager application of the FM-DM executes a reconfiguration strategy, which attempts to restore the replication degree of files in case of station failure and reestablishes the initial configuration when a station is reinserted. This manager application together with the FM-DM forms a *replication domain manager* (RDM).

FM - Application Entity (FM-AE)

The FM-AE offers the UNIX System V compatible remote access to replicated files. AEs are automatically instantiated by the replication domain manager when a new file is added to the domain (Create Service). This involves the generation of a file specific M-SAP and an endpoint on the FM-Association with the same number of replicas as the file.

FM - Object Manager Entities (FM-OME)

In a similar fashion to the AEs, OMEs are automatically instantiated by the DM when a new file is added to the domain (Create Service).

The object manager interface (OMI) between File-Manager-DM and the File-Manager-OMEs is an internal interface of the FM.

The file-cloning protocol starts with the generation of an OME and an AE (by the Domain Manager) and the establishment of the file's M-SAP and endpoint to the FM-Association on the new location. The new OME and AE from now on receive all messages on the FM-Association that the existing AE and OME receive. The OME then establishes the global context from the local context (as described in section 8.2.4.1).

The local context of a file comprises:

- the content of the file; and
- the file context maintained by the LEX, e.g., the file descriptors.

The global context is then transferred in voted packets. During the transfer steps, the services of the global file service continue to be provided. Having received all context packets, the content of the file is substituted and the file descriptors are reestablished at the new location (by local UNIX System V open calls). Finally, the output message stream of all file replicas is synchronized.

8.2.4.3. Management of the Multicast Communication Protocol Stack

8.2.4.3.1. The Systems Management Application Process (SMAP). The *Communication Domain Manager* (CDM) is the domain manager for the *Multicast Communication Protocol* (MCP) stack. The CDM is implemented as a set of cooperating domain manager processes, called *Systems Management Application Processes* (SMAP). As the set of MCP communication objects and their associated management information is not the same on each Delta-4 station, SMAPs are individual, non-replicated processes, and exist one per station. For dependability reasons, a SMAP mainly resides on the fail-silent *Network Attachment Controller* (NAC). The current structure and interfaces of a Delta-4 SMAP are shown in figure 14 and are further explained in the following sub-sections.

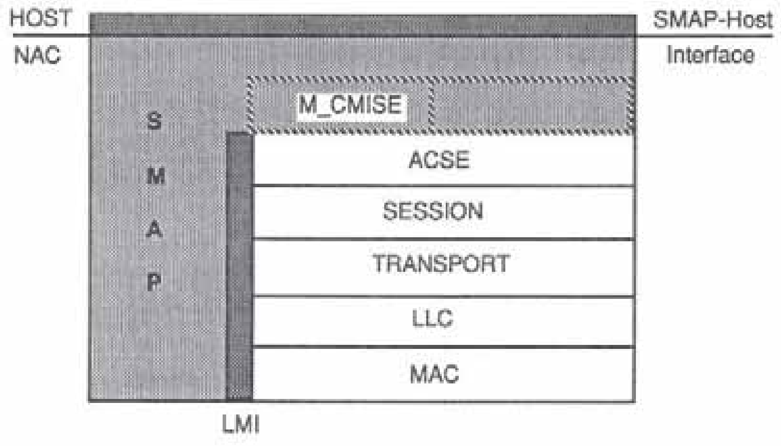


Fig. 14 - Structure and Interfaces of a SMAP

8.2.4.3.2. The SMAP-Host Interface. The SMAP-host interface allows management processes on the host to access the services of the SMAP. Whereas most of the SMAP software resides on the NAC, the implementation of the SMAP-host Interface also has some software components on the host. These are:

- two UNIX processes that handle the data transfer between host processes and the SMAP on the NAC (one for each direction);

- a set of C-language procedures that can be linked to any application process and allow the SMAP-host interface to appear as ordinary procedure calls to the application.

8.2.4.3.3. The Layer Management Interface. The *Layer Management Interface* (LMI) is the boundary between the SMAP and the MCP layers. The services that are offered at this interface are used by the SMAP to access the MCP managed objects. It currently offers the following services:

- LM_GET_VALUE to read management information;
- LM_SET_VALUE to write management information;
- LM_ACTION to require operations to be performed upon MCP managed objects;
- LM_EVENT_NOTIFY to report events spontaneously generated by MCP managed objects.

On the one hand, the LMI services are used by the SMAP to retrieve management information from the MCP layers. On the other hand, the MCP needs management support for "normal" MCP communication. This support is also given by the SMAP at the LMI. The following example illustrates the interactions between the SMAP and the MCP layers when an association endpoint is to be established:

When an application process issues an A_ASSOCIATE service request to the MCP ACSE sub-layer, the MCP ACSE generates an event at its local SMAP. This SMAP then checks if the association concerned is "open" ("open" means that all communication resources in layers 1-5 that enable the endpoint establishment on that association are allocated). If it is not, the concerned SMAP sends a request to the Global-MIB (cf. §8.2.4.4) to obtain the necessary parameters for the particular association (such as the services available on this association, the association context name or its amount of credit). Then the SMAP issues an action invocation to the LME of the MCP ACSE sub-layer to instruct it to continue with the A_ASSOCIATE service. The event and the action that follows it are transparent to the application (it is meaningless to perform the action alone, only the correct sequence leads to the desired result; generally, the correct sequence of such invocations is guaranteed by the SMAP).

8.2.4.3.4. Communication between SMAPs. SMAPs on different stations must communicate to perform their management task within the MCP communication domain. For this purpose they use the *Multipoint Common Management Information Service Element* (M-CMISE): this is an application service element within the application layer of MCP. M-CMISE is an extension of the ISO Common Management Information Service Element [ISO 9595] and provides generic multipoint services for the remote access to managed objects of all kinds.

As the SMAPs need dependable communication, M-CMISE is also implemented on the fail-silent NACs. It is mapped onto the MCP session services, and offers the MCP ACSE services or association control as pass-through services at the M-CMISE interface. Thus, management communication may also exploit the MCP multipoint communication facilities for its own purposes.

M-CMISE is used by the SMAP in the following way. The SMAPs use a single multipoint association that connects all SMAPs in the system. Each SMAP creates an endpoint on a dedicated multipoint association during its startup phase and keeps that endpoint throughout its lifetime. SMAP to SMAP communication then only requires a single M-CMISE service call in which the receiving SMAP is identified by its endpoint (or M-SAP address), thus saving the overhead of association endpoint establishment and release. Furthermore, by changing a parameter in the M-CMISE service call, a multicast or broadcast of a message from a SMAP to several or all SMAPs in the Delta-4 system is possible (which would be difficult using point-to-point connections).

8.2.4.3.5. Communication with the Global-MIB. Apart from the communication channel between the SMAPs, each SMAP communicates with the replicated Global-MIB. Communication between SMAPs and the Global-MIB is carried out by way of a dedicated multipoint association (the "Global-MIB-association") whose members are the Global-MIB and all Global-MIB users. The "Global-MIB-Manager" handles this communication and performs the required Global-MIB access protocol. See also §8.2.4.4.2 for a description of the Global-MIB contents and operations.

8.2.4.3.6. The SMAP during System Startup. The SMAPs play a special role during the startup of the communication software on a Delta-4 station. After the communication system and SMAP software has been loaded on the NAC and the NAC's processor has been started, the SMAP waits for a startup command from one of the SMAP-Host Interface processes on the host.

It then issues initialization action invocations to the communication layers, inserts the station into the network and allocates its own communication resources (and on "Global-MIB-stations" also allocates the communication resources of the Global-MIB).

Two different kinds of station startup are distinguished:

- startup during the initial startup of the Delta-4 system;
- startup of a station or restart of a repaired station when the rest of the Delta-4 system is already operational.

In the first case, the Global-MIB is started on a set of initial "Global-MIB-stations" during the startup phase. In the second case, the newly started station is always treated as a non-"Global-MIB-station" (however, the Global-MIB may be cloned to the new station after that station has become operational).

The Global-MIB is only accessible after completion of the startup phase of a station. Certain configuration parameters of a Delta-4 system must be known by the SMAP during this phase. They are generally specified at system generation time.

8.2.4.3.7. SMAP Support for Bi-Point Connections. Apart from Delta-4 multicast communication, the Delta-4 communication system also offers a standard ISO bi-point communication (with the extension of possibly replicated endpoints). Bi-point connections are mapped onto the Delta-4 communication resources of the lower layers (Transport to Physical) in the Session layer. This requires special support by the SMAP during connection establishment:

- Provide a unique global name for the communication resources "Transport Connection" and "Gate" per each bi-point connection. This is done by a request to a naming authority (a "global name server") which is integrated into the Global-MIB (cf. §8.2.4.4) and which manages a set of logical names.
- Ensure the allocation of the required communication resources (Transport connection, gate) on all requester and responder stations through a protocol between the involved SMAPs.
- Coordinate the replies from the responder stations (concerning the communication resource allocation) and instruct the local session entity to proceed with the bi-point connection establishment protocol, for which it needs the newly allocated communication resources (or to stop the protocol).

8.2.4.4. The Global-MIB. Management information that is stored in the Global-MIB is critical to the system's integrity and dependability; thus the Global-MIB must have a considerable degree of dependability. The implementation of the Global-MIB as a managed object comprises:

- an *Object Manager Entity* (OME) to manage information in the Global-MIB which is relevant to its replication and cloning; and
- a *replication domain manager* offering a service for the on-line cloning of new replicas of the Global-MIB.

The design of replication domain managers concerned with the Global-MIB has additional requirements compared with replication domain managers concerned with ordinary objects. In the current implementation, a dedicated domain for the replication of the Global-MIB is assumed.

8.2.4.4.1. The Global-MIB-Manager. The architecture for object management was outlined in §8.2.3.2.1. The *Object Manager Entity* (OME) of the Global-MIB is integrated into the *Global-MIB* object; the Global-MIB and OME software component is referred to as the *"Global-MIB-Manager"* (MIB-M). Similarly to the FM (cf. §8.2.4.2), the Global-MIB-Manager has an "application interface" and a "manager application interface". The Global-MIB Manager is implemented as a replicated manager process. Likewise, the replication domain manager for the Global-MIB is implemented as a set of replicated processes (the degree of replication and the location of these processes is the same as for the Global-MIB Manager).

8.2.4.4.2. Management Information Stored in the Global-MIB. This section gives an overview of the management information stored in the Global-MIB and how it is structured in the present implementation. Currently the Global-MIB contains "domain manager-integrated management information" for the CDM (cf. §8.2.4.3) and the FM (cf. §8.2.4.2).

1) CDM-integrated Management information

The "domain manager integrated management information" that is encapsulated by the CDM is described in §8.2.4.3. This information is local to a Delta-4 station and is accessed using the techniques described in section 8.2.4.3.4.

However, the Delta-4 fault-tolerance mechanisms require that parts of this management information be non-local to a Delta-4 station, i.e., the information that represents the attributes of the Delta-4-specific communication objects "multipoint association" and (replicated) "M-SAP". This information is held in the Global-MIB. The alternative having each SMAP on each station store its own copy of this information has been discarded because of:

- *Lack of memory space on the NAC:* The SMAPs reside on the NAC and would have to store the information in the NAC memory. The amount of information is not known beforehand but may be large. The Global-MIB is a replicated object on the host and can store its information on disc.
- *Consistency problems:* The SMAPs are "individuals" and do not act as replicas. It would have been necessary to implement a consensus protocol between the SMAPs without being able to use the Delta-4 Multicast Communication System facilities to their full extent. Using the replicated Global-MIB, which uses the multicast property and order guarantee of replicated Delta-4 endpoints, consistency among the Global-MIB replicas is preserved automatically.

2) Domain Manager information not stored in the Domain Managers

A domain manager may decide to use the services of the Global-MIB to store (part of) its global information. As an example the File Manager for replicated UNIX System V files stores information about file replicas in the Global-MIB. Manager applications of the FM make use of this information.

The information in the Global-MIB is organized as an abstract data type. The applicable operations are invoked at the application interface and are controlled by the MIB-M. The data is structured according to an identified set of manageable components, which are in the domains of the CDM and the FM, their attributes and the operations that can be performed on the attributes.

Currently, the Global-MIB defines 10 subtypes:

- delta_4_system;
- lan_segment;
- station;
- NAC;
- host;
- Multicast Service Access Point (M-SAP);
- multipoint association;
- replication_unit;
- file;
- replication_domain.

The first five data structures contain information about the configuration of the Delta-4 system and its major components, stations and LAN-segments, together with the primary communication objects that are to be managed: the multipoint associations and Multicast Service Access Points (M-SAPs).

The subtypes "replication_unit", "file" and "replication_domain" represent File Manager management information.

In general, attributes of any subtype may be:

- structural, reflecting the configuration of a Delta-4 System; or
- descriptive, holding information about manageable components or domains.

As an example, certain structural attributes show the decomposition of a Delta-4 system (figure 15).

A "delta_4_system" is composed of sets of "lan_segments", "stations", "replication_units", "M-SAPs" and "multipoint associations". A "lan_segment" is connected to a set of "stations". A "station" is composed of a "NAC" and a "host" and connected to a "lan_segment", etc.

An example of a descriptive attribute is the "lan_type" of a "lan_segment" (Token Ring, Token Bus, ...).

Note: Replicated M-SAPs and Multipoint Associations, as seen from Delta-4 management, incorporate two categories of management information: information pertaining to the managed object as a whole (e.g., its logical address), which is dealt with here, and information local to a Delta-4 station (e.g., the number of messages sent from a given Delta-4 station onto a multipoint association). This latter information corresponds to the ISO/OSI management information model, it is accessed using standard techniques: information exchange through M-CMISE, a service that uses an ISO/OSI addressing schema (object class registration tree and object instance containment tree [ISO 10165-1]). Figure 12 shows the object class registration tree used in Delta-4 for addressing local management information (which is specific for a given

Delta-4 station). Figure 15, on the other hand, illustrates attributes of the non-local management information that reflect structural relations between the managed objects. However, this is not used for addressing the information in the Global-MIB (an ISO-like addressing schema would be much too inefficient for the performance of the Global-MIB). As M-SAPs and Multipoint Associations have both categories of management information, local and non-local, they appear in both figures 12 and 15.

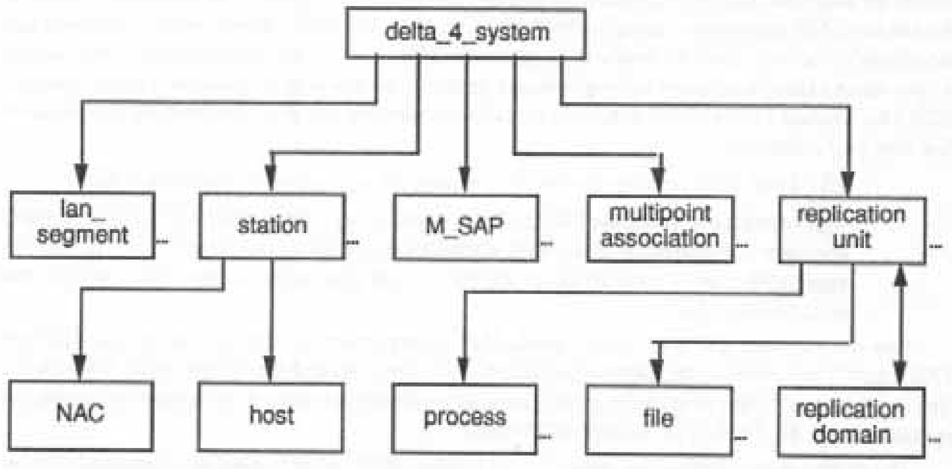


Fig. 15 - Structural Information about Manageable Components and Domains in the Global-MIB

8.2.4.4.3. Service Interface of the Global-MIB-Manager. As stated in §8.2.4.4.1, the *Global-MIB-Manager* offers services at an application interface and at a manager application interface. The application interface offers the normal object operations of the *Global-MIB* object. General operations offered at the application interface are:

- insertion or removal of Global-MIB entries, i.e., instances of data types representing manageable components;
- getting or setting values of single attributes in an instance of a manageable component;
- several operations for handling list type attributes;
- searching for a MIB entry of a given kind with a given name.

Special operations that may be used only by a *Systems Management Application Process* (SMAP) are:

- retrieval of all parameters needed for the opening of a given multipoint (or bi-point) association;
- retrieval of all parameters needed for the establishment of a given M-SAP;
- update of all concerned information in the case where a given multipoint (or bi-point) association has been opened or closed on a Delta-4 station;
- update of all concerned information in the case where a given M-SAP has been established or de-established on a Delta-4 station;
- preserve the Global-MIB database consistency and change NAC and host states in the case of a station failure or insertion of a new station into the system.

The manager application interface offers the services used by the Global-MIB domain manger for the on-line cloning of the *Global-MIB* object.

8.2.4.4.4. Service Protocol. Requests for operations on the Global-MIB and the results of such requests are transferred as messages via the Delta-4 multicast communication system. For this purpose, there is a dedicated multipoint association on which the Global-MIB owns an endpoint (characterized by the logical names of the association and the Global-MIB's M-SAP, which are both configuration parameters of a Delta-4 system). A Global-MIB client connects to the Global-MIB association using an M-SAP of its own. Both the Global-MIB's endpoint and the client's endpoint may be replicated, the only difference is that for replicated endpoints a voting mechanism is invoked during message sending. In this way, consistency of the Global-MIB data (across Global-MIB replicas) is simply preserved because the Delta-4 data transfer facilities guarantee that:

- all Global-MIB replicas receive the requests for operations in the same order;
- reply messages from the Global-MIB replicas are voted upon and only one reply message is transferred back to the corresponding client (the converse is also true for replicated clients: their requests are voted upon and only one message is transferred to the Global-MIB).

Any client wanting to access the Global-MIB incorporates an interface module (a "Global-MIB agent") that handles the data exchange between the client and the Global-MIB. SMAPs are special clients and therefore incorporate a special interface that gives them access to the special operations (cf. §8.2.4.4.3) on Global-MIB data.

The retrieval of information stored in the Global-MIB requires special considerations for system bootstrapping. No information stored in the Global-MIB is obtainable before the network is operational and the connections between clients and Global-MIB are established. Thus all information needed before that stage is configuration information of a Delta-4 network that must be fixed before network installation. Once the Delta-4 system is operational, its initial configuration may be changed dynamically.