A Supporting Platform for Heterogeneous Networks Management Systems

Fouad GEORGES
email: fg@masi.ibp.fr

José NEUMAN (*)
email: jn@masi.ibp.fr

S. LALANNE
email: sl@masi.ibp.fr

J.P. CLAUDE
email: jpc@masi.ibp.fr

Laboratoire MASI
45 avenue des Etats Unis
78000 VERSAILLES
FRANCE

Abstract

Network management is characterised from other information processing systems by their constant evolution and the load they represent on their hosting infrastructure. These requirements lead to the definition of certain supporting functions, offering network managers a suitable environment for the development and execution of their automated management tasks. The TMN (Telecommunications Management Networks) standards describe a framework for the management applications' implementation and integration into the network administration process along with the neccesary utilities for the realization of such technique. Run-time libraries, Precompilers, special interfaces and software systems are all examples of such utilities. This paper treats the TMN needed environment and proposes a solution based on a "supporting-platform" approach. In the following sections, the specifications for such a platform are briefly presented. These specifications resulted from a requirements analysis for the implementation of a performance management system for integrated networks. However, it should be noted that they are equally applicable to other management domains.

Key Words: *Network Management, TMN, MIB, Supporting Platform*

Introduction

"It is clearly vital to all involved in the production of a TMN to understand the uses to which it will be put" [1] i.e. the requirements imposed on such a system. In fact, there are three factors that participate in a platform specification: functional requirements; non-functional and technology requirements; and finally environmental constraints. The main goal of such classification is to facilitate the design process, by the separation of crutial and less-crutial issues. Thus, in contrast to the functional requirements, the non-functional ones are those which do not (or are considered not to) have an impact on the main functionality of the system (e.g. cost, argonomy, etc..). Hence, these requirements are not taken into account during the design phase. However, they intervene later by playing an important role as criteria for the quality of a certain implementation of the design. It should be noted that requirements classification is relative to the design context, e.g. cost might be considered non-fuctional w.r.t. performance management but is a main issue for accounting management.

The role of TMN supporting platforms is mainly to provide for the engineering mechanisms of network management systems. In fact, for a given system, we distinguish between two categories of functionalities, namely computational and engineering ones, where as the first concerns applications implementing the intended management activities, the second is only about infrastructure mechanisms (i.e. those recognized and offered as basic, common or

* Supported by CNPq and BNB S/A

public to all management applications). This distinction leads to regarding the same management system from two perspectives (or viewpoints). Thus, from an engineering perspective, a management system view (fig. 1) is that of a set of cooperating applications running on top of a supporting platform and controlling underlying network elements. Management applications depend on the platform for supporting their access to the network and their interactions. This view gives the functional requirements detailed in the following section and which form the basis for the platform's specifications. The design of the platform structure is based on an architecture proposed by the IDEA[2] methodology which was developed in the MASI laboratory. Validation of the current specifications has resulted several prototypes in the experimental stage in the same laboratory.

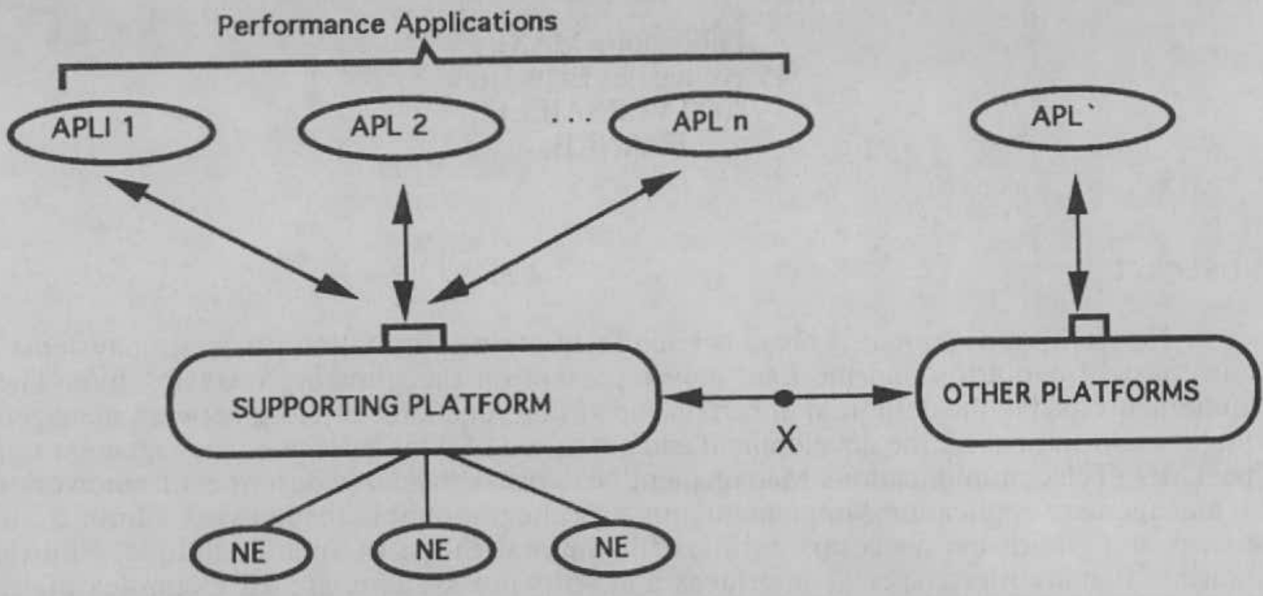


fig 1 : The Platform's Engineering View of Network Management Systems

Functional Requirements on the Platform

The following requirements concern the infrastructure mechanisms identified by management applications designers. At the present stage, only basic requirements were treated and the respective specifications represent the result of a first iteration in the platform's design process. Further iterations allow the introduction of new requirements (e.g. security, second order management, etc.), and the refinement of current specifications after their validation and analysis. It should also be noted that requirements, at this level, are not necessarily specific to performance management but are equally applicable to other management domains. Thus, the resulting specifications might be considered general to network management as a whole.

Network Access Requirements

The diversity of network elements liable to be managed makes it necessary for the management system (MS) to take into consideration the heterogeneity of these managed objects. In fact, the problem of heterogeneity manifests itself when a set of networks are connected together to form an infrastructure for communications. These networks along with their components have different designs and technologies. Therefore, the supporting platform must allow for the co-existance of these incompatible elements and must be capable of handling such a situation. It should be noted that heterogeneity is not only about communications but also about the different management solutions and capabilities provided by each element which inhibits the possibility of managing all the network in a uniform manner. The main aim is to hide the complexity due to heterogeneity from the developer i.e. to relieve him from dealing with details related to the diversity of the systems within his environment.

In fact, some management systems exist and are efficient in managing locally their own network components. But this management is inefficient when the network is connected to the external world. A kind of unification mechanism is required that offers the user an abstract view (where certain irrelevant details are masked out) of his underlying environment. In this manner, this unification mechanism may be considered as one of the transparency facilities offered to a user as it provides him with a virtually homogenous view of a heterogenous system. On applying such a mechanism, the user is detached from the real system but deals with it through the intermediation of the transparency support.

Storage Requirements

This is about the infrastructure mechanisms needed for information accessibility across the system. The system must support the management information acquisition, storage, retrieval and manipulation. In this specification, a conceptually single access point to the system's information is given via the visible interface. Management information is recognized to have two different natures, namely active and passive. Active data essentially signifies any network parameters that must be provided on-line. Passive data, on the other hand, include any information that can be kept in the platform information-bases. However, there is a unique representation for these two data natures given by the platform. Applications are thus enabled to handle all the system's information indifferently. The platform must include the necessary mechanisms for the aquisition, storage, retrieval and manipulation of this information. It must also assure the consistency and integrity of its data representation.

Communication Requirements

This is about the infrastructure mechanisms needed for information transfer across the system. There are three recognized categories (types) of information transfer for a management center:

Applications Communications

Performance management applications will need to inter-communicate with each other as well as with the platform. The platform must provide an interface visible to these applications and which they can use to transfer their management information. This **visible interface** must support several communications modes e.g. connected (association), non-connected synchronous/asynchronous.

Infrastructure Communications

As the platform itself will be implemented in a distributed manner, there is a need for mechanisms enabling the physical separation of its building blocks. Since, the possible internal messages are predefined, there will be need for a single communications mode. In fact, this **hidden interface** is considered to be a sub-set of the visible one (to keep level of genericity for the supporting mechanisms regarding internal/external communications). However, there is also need to follow up the chaining of messages inside the system i.e. when an external request results in multiple internal message. This is done for the sake of the operations atomicity.

Interworking Communications

Interworking provides for two systems exchanging information in a meaningful way. This type of communication essentially addresses the interoperability problem. The performance management center should be able, via its platform, to interwork with external management centers. In order to keep the openess objective of the platform, its "interoperable" interface must support CCITT's X-interface using the CMIP [3] protocol. Context negociation and authentification mechanisms also take part in the functionality of the interface provider.

Interoperability Requirements

Management centers, built on top of different platforms will co-exist and must be capable to interact i.e. they must mutually provide for information exchange while at the same time maintaining a high degree of autonomy. The OSI/NM Forum defines the framework for the interoperation of different management systems to manage the overall communications network. The cooperation strategy is based on a Manager/Agent relationship where the Manager role is to initiate cooperation by sending messages to an Agent of another management system which in turn is capable of performing management activities on behalf of the requesting manager.

It should be noted that the above interoperations do not represent any supporting mechanisms but are computational activities of the management system itself i.e. are performed by management applications implementing the above Manager/Agent roles. However, they imply that, for a conformant management system, its supporting platform must provide for:

- Interoperable Interface: set of protocols, procedures, message formats and semantics used to communicate management information .
- Conformant Management Entity (CME): A CME is a real open system which supports the OSI/NM Forum defined interoperable interface. Thus, two CMEs communicate across the interoperable interface.

Inter-Processing Requirements

Management applications will have to cooperate together in order to coordinate and complement their management activities. Meanwhile, applications should not have to bother with where and how information or functionalities are implemented. The different transparencies determine the extent to which programmers need to be concerned with, and have control over certain system details. In a non transparent program the programmer has taken full responsability for all aspects of distribution. On the other extremity, he has delegated all responsibility for inter-processing to the support environment i.e. in a completely transparent system the user "sees" only one abstract (or virtual) machine.

Some well known transparency mechanisms are location, access and replication transparencies. The kind of a transparency defines its criterion of abstraction i.e. the type of system information which is hidden from the user :

- Access transparency: provides identical invocation semantics for both local and remote components.
- Location tranparency: hides the exact location of a program component from any other.
- Concurrency transparency: hides the existence of concurrent users of a service.
- Failure transparency: hides the effects of partially completed interactions that fail for what ever reason.
- Replication transparency: hides the effect of having multiple copies of program components.
- Migration transparency: hides the effect of a program component being moved from one location to another.

The fact that transparency mechanisms hide some of the systems aspects from applications essentially means that these applications do not have control over the system's behaviour concerning this aspect. Thus, although transparency is desirable for most cases, it should not be imposed by the system. A programmer must be given the capability to select the kinds of transparency required when declaring an interface between two application components. Two different styles of invoking a transperancy support can be distinguished:

- Imperative - the programmer must call lower-level libraries from the program source at runtime to explicitly drive transparency mechanisms.
- Declarative - the programmer states distribution requirements in the program source; tools can then be applied to the source to yield components which meet the requirements.

The Supporting Platform Structure

The platform (fig. 2) is composed of several modules generated on top of a group of TMN nodes. These modules interact by exchanging messages via the DISPATCHER. There is a conceptually unique access point to this platform which is realised by the visible interface. In fact the interface shown in the figure as well as the other modules might be replicated for availability or reliability reasons. During the platform installation, one or more instances of the different modules are generated and configured across the system e.g. the shown INDB maybe actually implemented using distributed or federated databases. Moreover, the whole support is not intended to be implemented on one node. This implementation freedom is needed for the optimization of the development efforts by maximally exploiting the capabilities of the underlying systems (the INDB might profit from a specialised object oriented data base machine). The module responsible for keeping the virtual image of one platform together is the DISPATCHER, which is considered as the kernel. The description of the support's main components is as follows:

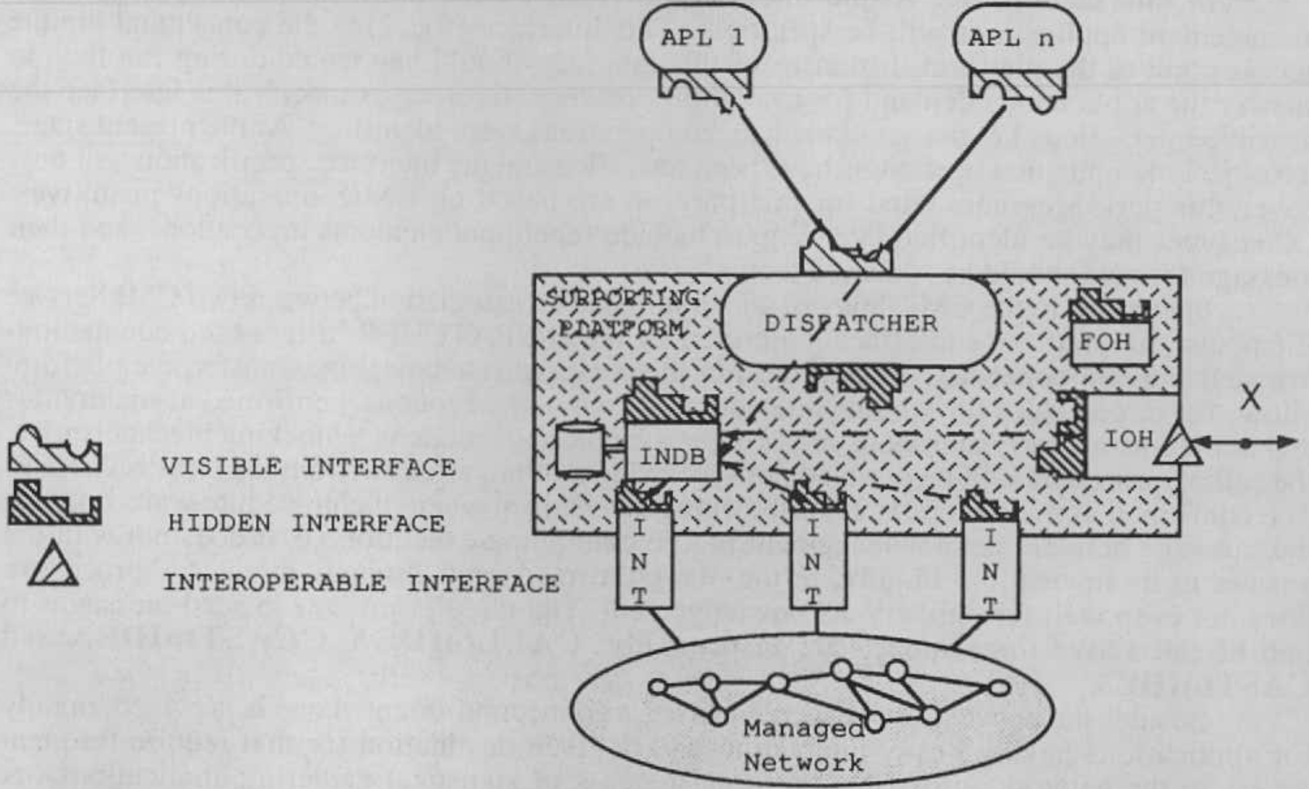


fig 2 : The Platform Structure

DISPATCHER: It represents the platform kernel. It controls the communications and the exchange of messages between the platform's modules as well as between applications and the platform.
INTEGRATOR: This component handles the functionality of interfacing to the manufactures access points and representing them as a common virtual access point. In order to do this, it provides a syntactic unification of the access methods (e.g. the translation of an incoming request to the respective manufactures management commands) and a semantic unification of the manufacturers management operations.
INDB: This component supports the integration and the manipulation of the platform's information (those of the existing INTEGRATORs as well as the data bases). Any interogation concerning management information must be directed to this module.
IOH: The **I**nter-**O**perability **H**andler is the component providing the possibility for the platform's applications to cooperate with external management centers. In contrast to the other components, it must support one or more interoperable interfaces (e.g. CMIP for ISO-based and CMOT for internet-based management centers). It also controls the visiblity of management information and maintains their correct semantics on top of the different interfaces.
FOH: This is the **F**unctional **O**bject **H**andler. It provides a repository of the management services available on the platform. This allows applications to fulfill their needed activities by invoking actions on shared (public) or imported functional elements [4].
Visible Interface: This is a set of primitives (offering 3 communication modes) used by the platform's users (e.g. performance management applications) to communicate with it. Management messages (which is CMIS [5] based) are encapsulated within a given structure before sending it to the platform. Messages are first collected by the DISPATCHER and then delivered to the proper module (or to another application).

It should be noted that such an architecture is suitable for network administration in general. The orientation of the platform towards performance management concerns the management information i.e. during the specification of the models of both the INDB and the INTEGRATOR and also by the generic services that are represented in the FOH model.

The Platform's Interface Specification

In this section, the visible interface provided by the platform to the performance management applications will be specified. This interface (fig. 2) is the conceptual unique access point to the platform. Instances of this interface should be created during run time to answer the applications demand for availability reasons. In order to specify this interface the possible interactions i.e. the set of exchanged operations were identified. At the present stage, the object manipulation operations have been identified and the interface specification will only cover this part. Messages used for this purpose are based on CMIS operations primitives. Other types may be identified later, e.g. to include functional elements invocations, and their message formats should be specified then.

In contrast to the CMI Protocol which imposes an association between two CMI Service Elements, the platform's interface, controled by the DISPATCHER, offers both connection-oriented and non-connected possiblities. For the non-connected message transfer, the platform allows for three modes for module interaction, namely, synchronous, confirmed asynchronous and non-confirmed asynchronous modes. The synchronous mode is a blocking mechanism i.e. the calling procedure will be disabled until receiving a return message from the DISPATCHER. The confirmed asynchronous is a non-blocking mechanism where the procedure waits only for the message delivery acknowledgement before continuing execution i.e. it does not wait the answer to its invocation. Finally, in the non-confirmed asynchronous mode, the procedure does not even wait for delivery acknowledgement. The three primitives to send messages in one of the above three modes are respectively: **CALLtoIDEA**, **CCASTtoIDEA** and **CASTtoIDEA**.

Besides the above non-connected modes, a connection-oriented one is provided, mainly for applications having heavy interaction with a given destination (or that require frequent access to the network across the platform, the case of statistical gathering applications). A **BIND** primitive allows such applications to work in association with a designated block of the platform. In this case, the DISPATCHER will initiate a special instance of the visible interface, notifies the destinator of the new session creation and returns the interface reference to the application. As from this time, the DISPATCHER no longer intervenes until the initiation of an **UNBIND** primitive to demand the closure of the association. In this way, the application enters in direct communication with the destination (normally a platform module) and will no longer need to address via the DISPATCHER i.e. it will not need to pack its messages in the IDEA_Frame. This also permits the platform to answer a request with multiple responses as soon as they arrive (case of notification) or at determined intervals (case of data collect).

This diversity of modes, permits a developer to choose the appropriate one according to the application's activities needs. This should not be confused with the operations confirmations to invocations (discussed later) which can be collected with or without blocking. It is evident that the actual inter-process synchronisation is up to the client applications i.e. these applications must be able to handle the proper protocol (a connection or a rendez-vous) with another application, the platform's role being restricted to the transfer of the proper protocol primitives.

The Message Format

The parameter to the communications primitives is the message to be delivered to one of the platform's blocks e.g. the INDB, or to another application. This message (called IDEA_Frame), shown in fig. (3), consists of a header (called IDEA_PDU), for the DISPATCHER usage and the CMIS-based message body (called IDEA_Message) that should be directed to the concerned block. A formal specification of the interface i.e. the primitives along with there parameters is given in annex 1 using IDL (Interface Definition Language) [6].

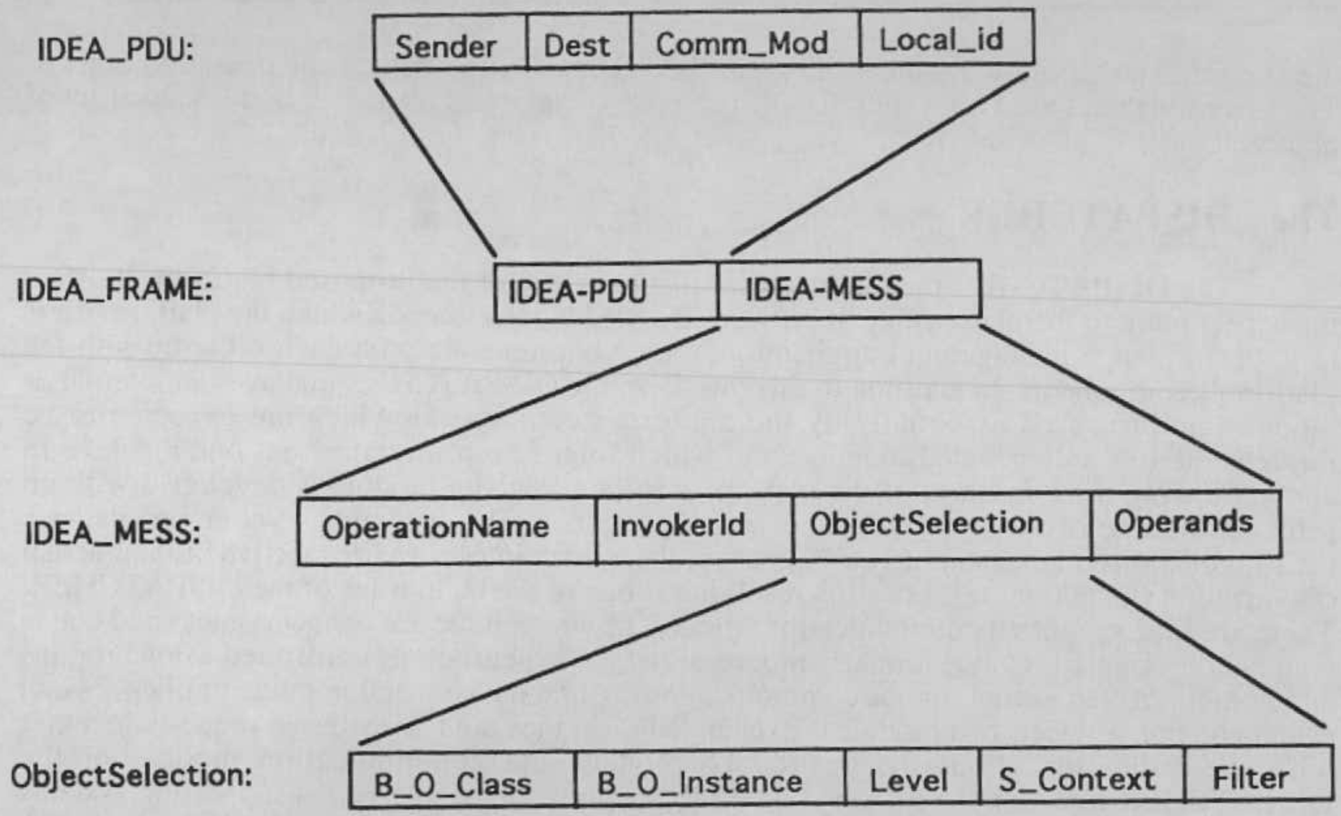


Fig. 3: The Structure of an IDEA_FRAME

In the following, an enumuration of the basic operations provided by the platform's modules is given, these are essentially the CMIS invocations plus a special one called COLLECT. This is equivalent to sending a same GET request regularly i.e. it invokes the platform to return the attribute(s) of one or more object instances every fixed interval and during a given period. As this given period maybe infinite i.e. continuous monitoring, a CANCELCOLLECT is also provided to abort this action. It should be noted that this invocation is used in association with the **BIND** primitive i.e. it works only with the connected mode. The description of the platform's set of messages (possible entries for the field "OperationName" is as follows:

CREATE	To add a new managed object instance to the members of its class.
DELETE	To remove an existing managed object instance from the set of its class members.
GET	To retrieve management information from the INDB. CANCEL-GET is used to stop the retreival process.
SET	To request the modification of managment information from the INDB
COLLECT	To gather the values of a given set of management information regularly and during a given period. CANCEL-COLLECT is used to stop the data gathering process.
ACTION	To request the platform to perform an action.
EVENT_REPORT	To express interest in a certain class of notifications. CANCEL-EVENT_REPORT is used to end the interest in the notifications.

Once the invocation's operation is specified, a set of instances should be identified as the targets of this operation. The names of instances of managed objects are arranged hierarchically in a management information tree. A specific object is selected for the execution of a given operation by a pair of selecting mechanisms namely: scoping and filtering. Scoping defines a context for scanning an ensemble of objects while filtering gives the criteria (conditions) of the object selection from among the ensemble. The scanning of objects starts at a base object which is the root of a MIT subtree and continues to a given n^{th} level. The simplest scoping is that of the base object alone. Otherwise, the filter maybe applied to either a part of the subtree root (i.e. all subordinates down to, or at, a given level) or to the whole subtree (all the subordinates). Following the scoping, a filtering mechanism selects the operation objects. The assertions are based on testing for the presence of values of attributes in the scoped set of objects.

The DISPATCHER

The DISPATCHER may be regarded as the kernel of the proposed platform. It offers the access point to the platform i.e. it provides the visible interface via which the platform users (e.g. performance management applications) intercommunicate with each other or with the platform's components. In addition to this interface, the DISPATCHER manages an internal or hidden interface used essentially by the platform's components which are expected to be implemented in a distributed manner (and which might even migrate from one machine to another during the execution of the platform). In fact, a given module is developed with no prior knowledge of the whereabouts of its partner ones. The ensemble of components base their interactions on exchanging messages (via the hidden interface) irrespective of their actual organization (local/remote). Location resolving is one of the main roles of the DISPATCHER. There are four supported communications modes, of which three are connectionless and one is connection oriented. Connectionless modes allow for synchronous, confirmed asynchronous and non-confirmed asynchronous communications, while the connection mode implies session establishment between two modules. Exchanged messages are CMIS-based requests together with the indication about the desired component and communication mode. For the DISPATCHER to undertake its role, it interfaces with the local operations supporting systems and exploits their offered facilities (e.g. queues, sockets, rpc, etc.).

It should be noted that the information needed for the DISPATCHER operation is added as a header to the sent request. To facilitate the packing of the request into an IDEA message, a compilation-time library is provided. The applications have to fill a data-structure then call the library function that takes care of communicating the request to the DISPATCHER (a process commonly called marshalling/unmarshalling).

The INTEGRATOR

The INTEGRATOR is responsible for interfacing to the manufactures acces points and representing them as a common virtual access point. In order to do so, the INTEGRATOR groups whatever underlying local schemas into one set of objects defining the view of the network elements for the management functions. In fact, the unification procedure is started by partitioning the overall network into homogeneous domains, each having an access point (provided by a proprietary system) through which management of the internal objects may be achieved. The mapping between the INTEGRATOR model and the different domains is done individually for each domain. That is, for each object's property (attribute or method) in the INTEGRATOR's model there exists a per domain syntax and semantic interpretation functions. The above partitioning is logical and is done with respect to a certain management aspect (geographic, functional or other). That is several organizations for the same network may arise, each consisting of domains to which an INTEGRATOR instance is assigned. As such it is expected that more than one INTEGRATOR are needed to provide access to the network information through one reference point. This reference point is realized by an interface that the INTEGRATOR should offer to its clients and that defines the set of permissible operations on the objects of its model.

In this section a brief description of the INTEGRATOR's internal structure is given. Figure (4) shows the different building blocks needed by the INTEGRATOR to fullfill its role. The INTEGRATOR has two kinds of interfaces :

The "ADMINISTRATION" interface

This interface is provided for the network expert that will feed the INTEGRATOR with its knowledge of the underlying network. It is used for the instanciation of a specific INTEGRATOR by creating its objects. The knowledge also includes all the operational information such as the mapping procedures of each attribute in the model to the specific instances in the different domains. This block should undertake the necessary checks on its input (e.g. lexical and syntax checks, model integrity checks, etc..) before representing it in the form usable by the INTEGRATOR's shell. Presenting the knowledge includes stocking the schema of the model, its population (instances) as well as the encapsulation of the mapping procedures for the different attributes.

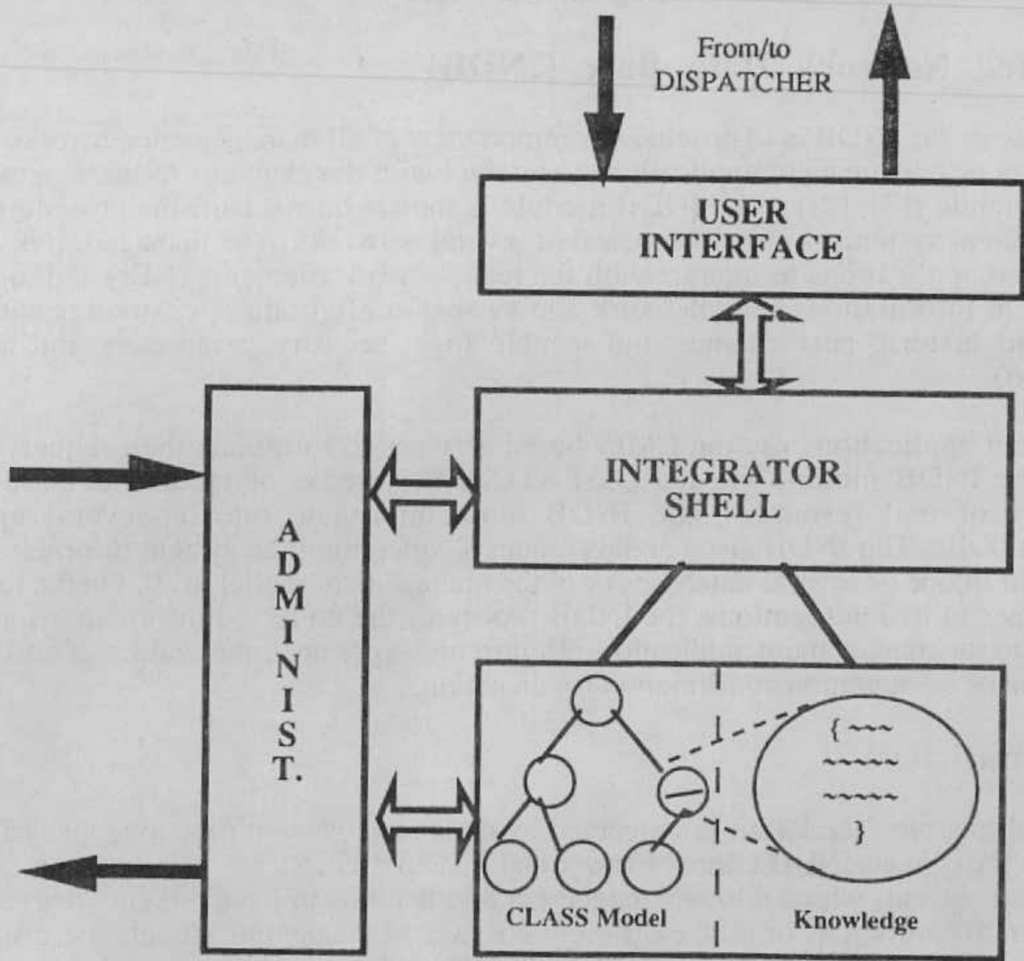


Fig. 4 : Internal Structure of the INTEGRATOR

The "USER" interface

It is to this interface that the INTEGRATOR's clients should import and use in order to manipulate its objects. Its type is that of the hidden interfaces and the operations invoked via this interface are CMIS based.

The INTEGRATOR's Shell

This is the main building block where the INTEGRATOR functionalities are supported i.e. on reception of an external request to undertake the invoked operation on behalf of the Manager. To achieve this some subfunctions are identified:

- Object Management
- Object Persistance Handling
- Syntaxic Interpretation and mapping according to the domain.
- Semantic Interpretation and mapping.
- Notification of abnormal events

The INTEGRATOR Information Base

The remaining two-part block represents the information base of the INTEGRATOR. This should not be confused with the INDB which is the repository to all the system's shared information. In contrast to the INDB, this information base stocks the internal knowledge of the INTEGRATOR i.e. its object descriptions, along with the different instances, their static attributes as well as the code for the procedures that permits mapping the attributes (syntaxically and semantically) to their corresponding real resources.

Integrated Network Data Base (INDB)

The purpose of the INDB is to provide a common view of all management networks in order to permit users or management applications to interact with the elements of these networks via a unique semantic ([7], [8]) . The INDB module is the set of mechanisms providing a unique view of the real system which is composed of several networks to be managed. It is used by all management applications to interact with the real Network Elements (NEs) and to store their management informations (e.g. network and system configuration, customers and services, current and historic performance and trouble logs, security parameters and accounting information).

Management applications use the CMIS-based services to formulate their requests and send them to the INDB module via the DISPATCHER. In case of reading or modifying the parameters of real resources, the INDB must interogate one or several appropriate INTEGRATORs. The INDB also handles requests concerning the system informations which may be held in one or several databases or in the management model itself. On the reception of the responses to its interogations, the INDB processes the collected informations and returns the results to the management applications. Return messages are either values of attributes for a consultation or an operation confirmation for an action.

Functionalities

To ensure these roles, the INDB is concerned by three management functionalities (fig. 5) :

- to provide an INDB Global Conceptual Schema (GCS), i.e. a unique point of view of the managed system, which allow management applications to have the knowledge about the management informations of all the different NEs via Management Models (described in OSI NM/Forum Templates [9], [10]), and their interactions between them via Management Trees (Inheritance, Naming, Containment and Registration Trees [11]). The GCS gives an unique homogeneous view of several heterogeneous environments.

- to provide an INDB Management Information Base (MIB), i.e. a conceptual repository which contains all the management informations of the Network Elements,

- to provide INDB Management Information Services (MIS) based on CMIS, i.e. a user interface on which management applications can manage (creation, deletion, manipulation) NEs.

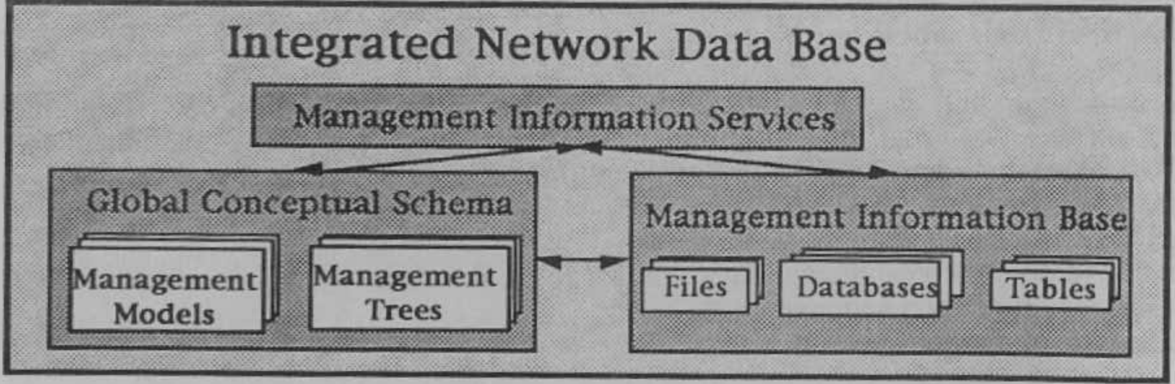

Integrated Network Data Base
Management Information Services
Global Conceptual Schema
Management Models
Management Trees
Management Information Base
Files
Databases
Tables

Fig 5 : INDB Functionalities

Logical Architecture

To ensure its functionalities, the INDB module has to manage three phases of treatment when an INDB request sended by users is received (fig. 6). These phases of treatment are request analysis, request treatment and reply generation phase.

Request Analysis phase

This first phase of analysis of the INDB request consists on its understanding by the INDB module, that means, its control, its interpretation and the fashion to resolve it. A first subphase of interpretation (research of selected instances in applying request scoping and filtering) followed by the subphase of localisation (GCS Manager, INTEGRATORS, MIB Manager) are realised to analyse the request.

Request Treatment phase

The second phase of treatment of the INDB request consists on its execution on the Network Elements via INTEGRATORS. Several kinds of execution are considered which are Storage, Collection, Modification and Activation of managed objects.

Response Generation phase

The last phase of generation of the response to the user consists on the collection of the results generated by the treatment phase and on the design of the INDB response.

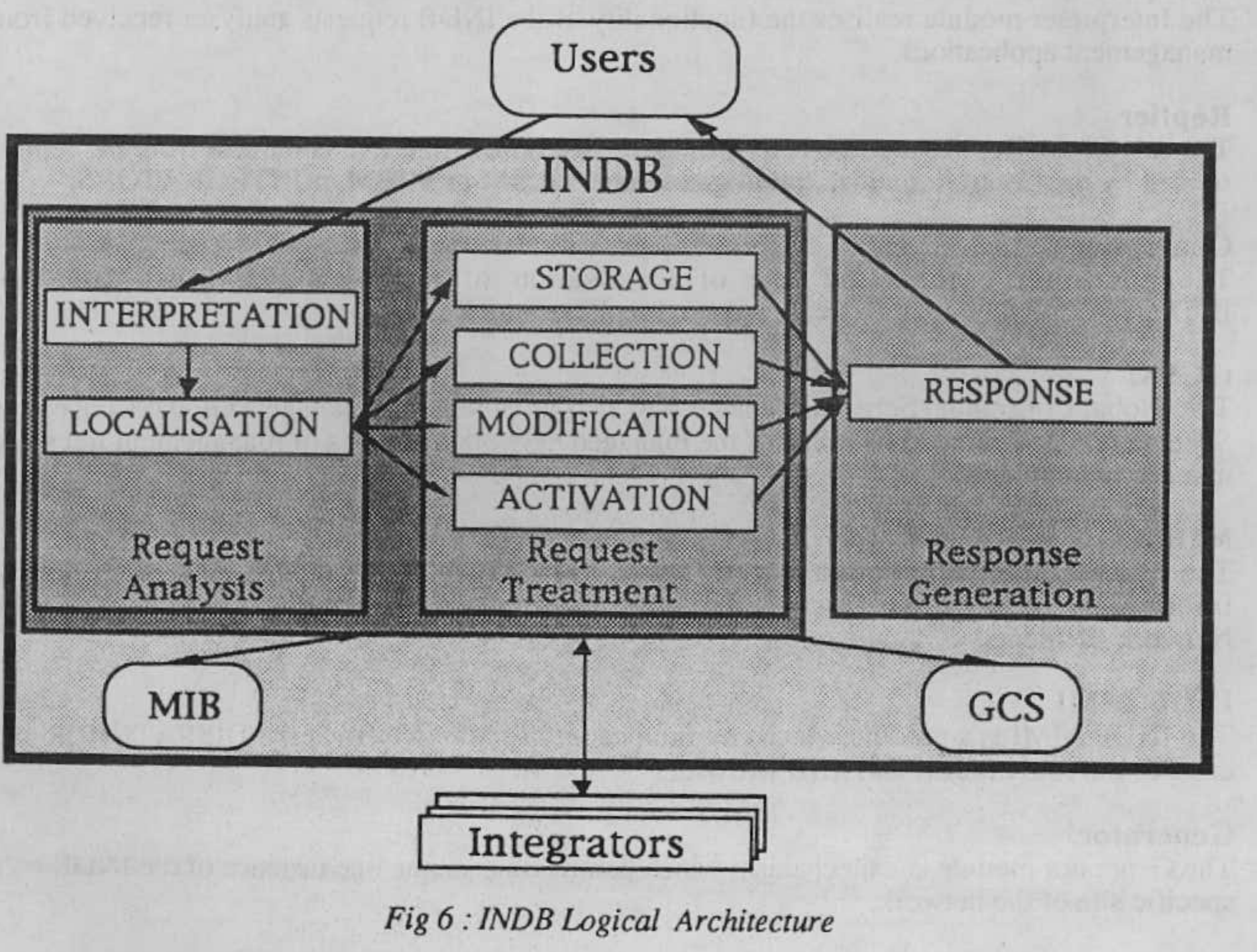


Fig 6 : INDB Logical Architecture

Physical Architecture

This section presents a brief description of the internal structure of the INDB which is composed of the following modules (fig. 7).

The INDB supports the organisation of the Network Elements. Organisation means the description and the relationships of these NEs throughout a Global Conceptual Schema (GCS) plus the manipulation of their management informations throughout either a Management Information Base (MIB) for static informations or existing INTEGRATORS for dynamic informations. Any management application request concerning management information must be directed to the INDB module.

The INDB Physical Architecture is composed of the modules :

Receiver
The Receiver module assumes the role of reception of messages from the DISPATCHER module and realises the unmarshalling of a message structure into an exploitable structure either for the Interpreter (Requests of management applications) module or for the Controller (Replies of INTEGRATORS).

Sender
The sender module receives either the INDB response from the Replier to the requested management application or requests from the Controller to a specific integrator and realises the marshalling of the reply or request structure into a message structure to send to the DISPATCHER.

Interpreter
The Interpreter module realises the functionality of the INDB requests analyser received from management applications.

Replier
The role of the Replier module is to format the response of the INDB request from the replies sended by the Controller which interrogates either GCSM or MIBM or INTEGRATORS.

Controller
The controller ensures the role of localisation of module (GCSM or MIBM or INTEGRATORS) to treat requests and send replies to the Replier.

GCSM
The Global Conceptual Schema Manager (GCSM) module has to manage a Global Conceptual Schema (GCS) i.e. the knowledge of the managed networks in terms of management trees and management models.

MIBM
The Management Information Base Manager (MIBM) module manages the Management Information Base (MIB) which is the repository of the management informations of the Network Elements.

INDB MMI
The INDB MMI is a module which communicates with the INDB via the DISPATCHER and ensures the role of GCS and MIB Browser.

Generator
The Generator module is a mechanism which permits to generate one instance of the INDB on a specific site of the network.

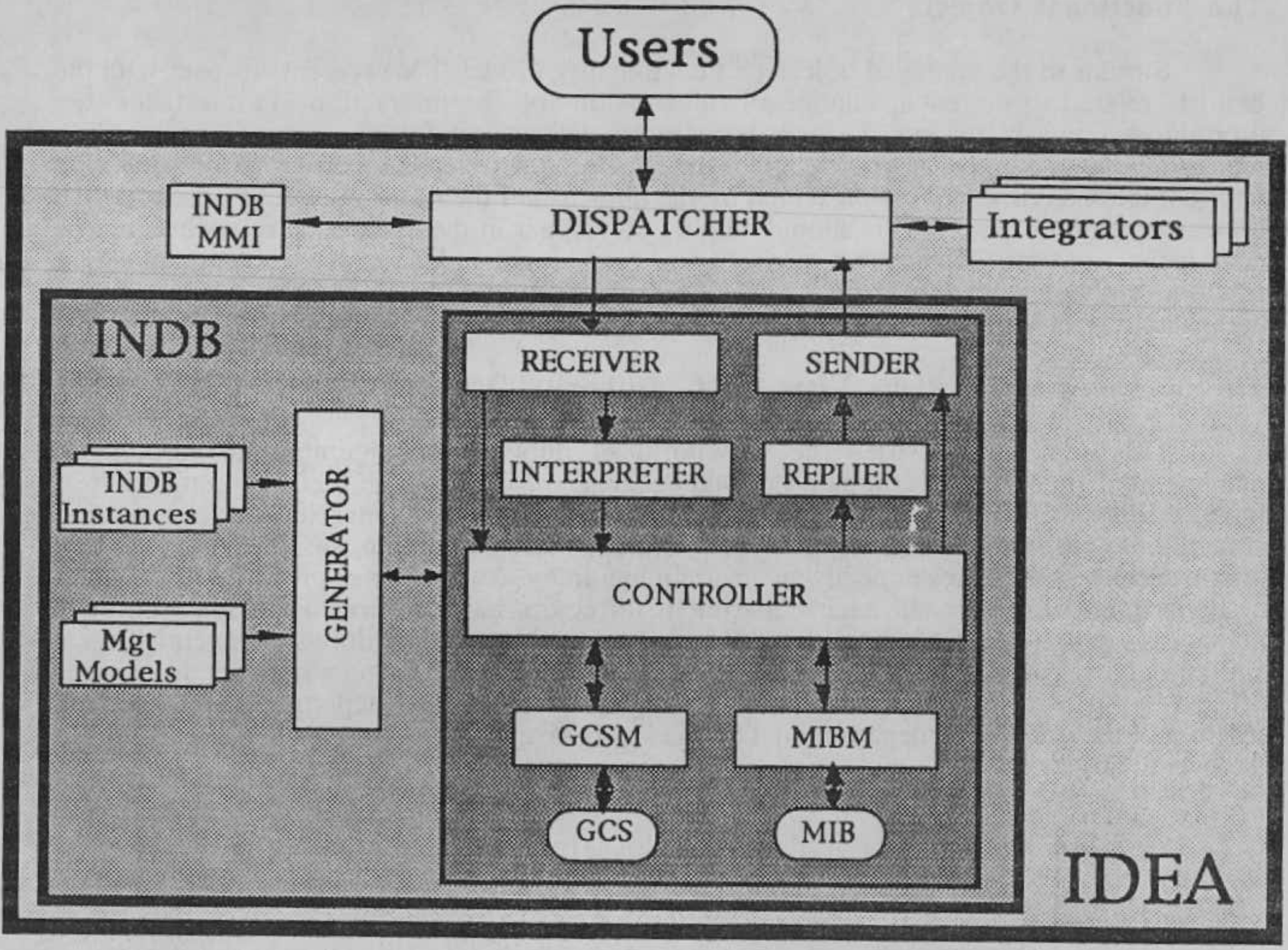


Fig 7 : INDB Physical Architecture

Functional Objects Handler (FOH)

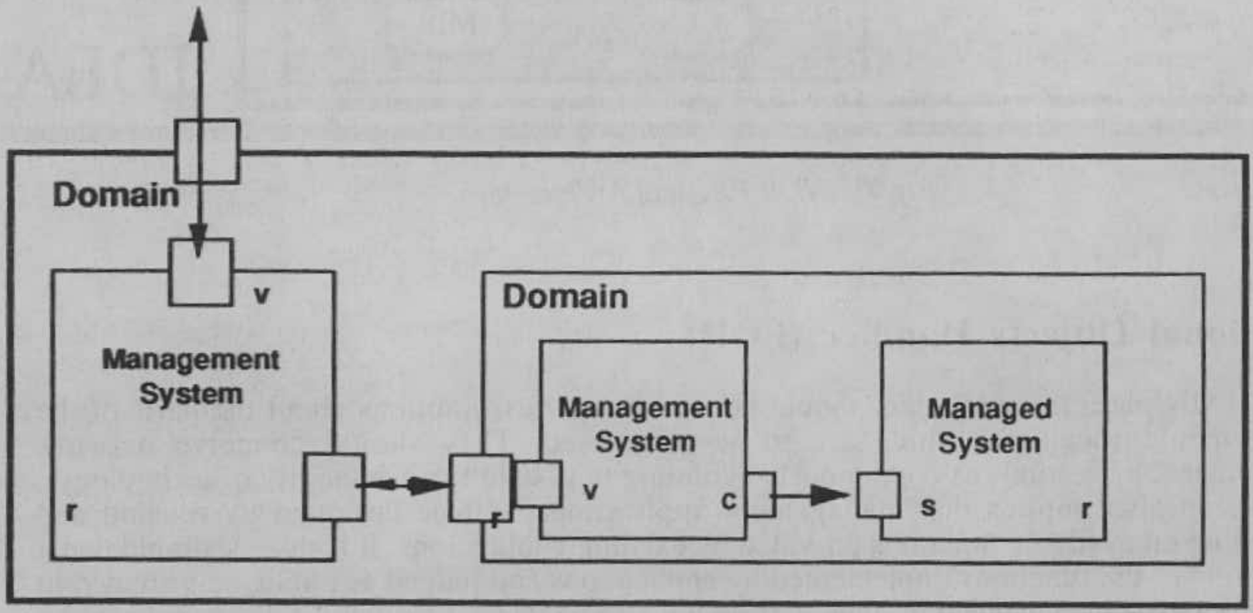
TMN platforms providers should not make early assumptions about the parts of the management procedures which are to be automated. They should conceive network management applications as continuously evoluting (e.g. with the availability of technology). This vision also implies that management applications will be designed by reusing and elaborating on available functions provided by existing applications. It is thus desirable that a description of the functions implemented by applications and judged as public be provided to the performance management designers. It should also be interesting that the designer be given the capability of reusing the services of these applications during run-time. The goal of the FOH is to provide an object oriented approach for the invocation of common services. In order to achieve this, a model is needed for the description of the common management services, their properties and their inter-relationships (in fact, a repository of shared services should be provided which represents an extension to the definition of the MIB). Moreover, supporting mechanisms should be incorporated in the platform so that applications may perform their activities by invoking, on-line, operations on shared or imported service objects. The needed model is subjected to the same modelling criteria as the management information model. This means that it should provide the MAs with enough details, yet with the suitable abstraction, about the existing functionalities. Enough details to allow them to be aware of the surrounding offered services (in analogy to the existing elements to be managed) and at execution time to be able to exploit these functions for the fulfillment of their own roles. In this section, the concept of a functionalities repository, and essentially on its model, is presented. This work is still in an investigation stage, therefore, no concerning specification is yet given.

The Functional Objects

Similar to the managed objects, a Functionality Model (FM) presents its user with the benifits related to the encapsulation of information and the interaction via interfaces (i.e. modularity, reusability, etc..). Such aspects are interesting for the purpose of building managment applications. A knowledge of the existing functions and their properties in such a very complex environment is presented by the objects and their attributes (e.g. access rights, allocated resources, etc.). This should help the developer in the estimation of such things as dead locks, run-time. Moreover, a message-based development, encourages the upgrading of a function without worrying about its effect on other applications (given that the downward compatablity of the interface is kept).

The Management Systems View w.r.t. Authority Domains

This section identifies the viewpoint of network management developers on management functions as compared to management information. In fact, the perception of functionalities differs from the inside to the outside of authority domains. Network organization according to authority domains (fig 8) is analogue to that of an entreprise. The administration organization in both cases depends on a certain hierarchy. An entreprise president has a board of department directors. In each department, directors have several divisions with staff managers (and so on). For administring this entreprise the president draws the overall targets and strategies. Each director manages his own department. For doing so, he needs services offered by his or by other departments (e.g. the *customer handling* department may need the services of the *accounting* department). Directors also have to cooperate together, to accomplish the overall targets.



v : visible port
s : supplier port
c : consumer port
r : recurring object

Fig 8 : A Simplified Domains Organization

A network administration domain is analogue to a department in an entreprise. Inside a domain, management functionalities cooperate to achieve the overall target. They recognise each other as potential providers of services. Some functions may offer themselves as "public" or shared . Meanwhile, these functions look to a superior domain as objects to be managed (to perform second order management).

In summary, we may state that a functionality views other cooperating functionalities differently from its superiors or subordinates ones. To it, subordinate functions maybe regarded either as services (in which case they should be represented as *managing objects*) or as functions to be managed (second degree management) i.e. *managed objects*. The same goes for the superior's view of the given functionality. As for the cooperating functionalities, they should be represented as "higher level" managing objects.

Migration of Functionality

While studying the different functionalities taking part in the management environments, the standardization bodies accepted the fact that not all of these functionalities should be automated. They stated that it is better not to make an early commitment to which functions will be automated and which will not. By doing so they allow for the migration of functionality from manual to automated (or human to computer). This situation must be allowed for in the FM. An object present in the FM may represent a human intervention untill the time it should be replaced by a management application (maybe without leading to changes in the FM).

Another type of functionality migration exists within the automated part itself. As higher technology permits, NEs will be built with more management facilities. They will have a kind of built-in automanagement. This situation is anticipated and allowed for in the TMN architecture. The CCITT MDF block provides for those needed functionalities missing in the NEF and which are anticipated for later incorporation. As NEs grow more and more capable (by adding built-in diagnostics and mentainance), the MD will diminish until it eventually disappears, leaving the OSs in direct contact with the NEs. This phenomenon should be provided for by allowing objects to immigrate from the SM into their corresponding MIB objects (as methods). In a layered model (detailed later) this may lead to the future disappearance of the bottom SM layer.

SM Object Composition Mechanisms & The Layering approach

Object composition mechanisms increase reusability and concurrency. It is the means to build new object types by making use of, and adding to the existing objects' functionalities. There are several mechanisms to compose objects, the CSA [*] (Communications System Architecture) provides three of them, namely Inheritance, Importation and Sharing:

Inheritance

This mechanism allows an object to inherit the operations (or a selected number of them) of its "super class" type. The inherited operations are visible at the interface of the inheriting object type. "The inheritance in CSA could be therefore considered as a means to delegate a request for an operation to objects which provide a response to that request".

Importing

In contrast to the inherited operations which are invoked at the super class on behalf of the inheriting object, operations of imported object types are completely hidden and accessed locally by their importing object. In this sense, imported operations may be considered as part of the importing object's local variables. Static import is undertaken at the instantiation of the importing object while dynamic import is assigned at runtime.

Sharing

Sharing is a means to declare public object types accessible by other object types i.e. a shared object may be regarded as playing the role of a server for its client objects.

Layering in itself is not a novel approach. It has been already imployed in different contexts. The merits of layering might be identified as:

* Flexibility in defining the interrelations between the two participating layers.
* Conceptual clearness by removing redundancy and reducing the number of entities.
* Reusability of entities on one level to support several entities on the adjacent level.
* Economic implementation by utilizing modules for several purposes.

Hence, by envisaging several SM layers to host the different management categories (not necessarily a layer per category) these advantages might be gained. In addition, this should give the possibility of refining the framework for each layer to accomodate its specifities. The juxtaposition of the different layers should give the totality of the managing objects. It should be noted that if the frameworks of two (or more) layers are found identical (or could be approached together), these layers may be eventually merged under one top. Actually, the SM layers differ according to the criteria of classifying the management functions (implementation-oriented, management-degree-oriented, etc.). A good criterion maybe the degree of elaboration of management. Thus, functionalities tending to directly manipulate the managed objects (resources) belong to one layer. Higher level management functions (e.g. configuration, performance, etc.) go into the next layer up. Finally, a top layer might be envisaged to englobe the "intelligent" functionalities (e.g. pilote system, man machine interfaces). The juxtaposition of these layers should give the integral view of the surroundings.

Conclusion

The architecture of the supporting platform for heterogeneous networks management systems has been developed continuously at the MASI laboratory. By now, the Integrator, INDB and Dispatcher blocks can be found in its first prototypes versions so that the main functionalities specified in the architecture are shown. These prototypes have been validated in the real world using environment like ethernet and X25 networks, where, for instance, the integrator block interacts actually with the Snmp and X25 integrators. This work has been undertaken in several europeen projects that are interested in investigating the construction of TMN platforms, like ADVANCE (RACE programme) and PEMMON (ESPRIT programme). The results obtained from these projects have been used as a feed-back on the support platform development. As a further work, we think to improve our specifications in order to have a generic platform and investigate other aspects of the network management information like security.

ANNEX 1:

This annex describes the idl specification of the platform's interface:

```
-- Specification of IDEA Interface in IDL

IDEA_InterfaceTypes : INTERFACE =

BEGIN

        I_BaseObject :          TYPE = RECORD [
                                BaseObjectClass     : STRING,
                                BaseObjectInstance : STRING,
                                        ];
        I_ScopeContext :        TYPE = { BASE_OBJECT,
                                INDIVIDUAL_LEVEL,
                                BASE_TO_NTH_LEVEL,
                                WHOLE_SUBTREE };

        I_Scope :               TYPE = RECORD [
                                BaseObject              : I_BaseObject,
                                Level                   : INTEGER,
                                ScopeContext        : I_ScopeContext,
                                        ];
        I_ObjectSelection : TYPE = RECORD [
                                Scope                   : I_Scope,
                                Filter                  : STRING
                                        ];

        I_Oper :                TYPE = { CREATE, DELETE, GET, SET,
                                ACTION, COLLECT, EVENT-REPORT,
                                CANCEL-GET, CANCEL-COLLECT,
                                CANCEL-EVENT-REPORT};

        I_Session :             TYPE = { CALL, CAST, CCAST,BIND,UBIND };

        I_Block :               TYPE = { INT, FOH, MA, INDB, IOH };

        I_PDU :                 TYPE = RECORD [
                                SenderName : STRING,
                                IDEA_Block : I_Block,
                                PDU_Size : INTEGER,
                                Session : I_Session
                                        ];
        I_MESS :                TYPE = RECORD [
                                OperationName : I_Oper,
                                InvokerId       : INTEGER,
                                ObjectSelection : I_ObjectSelection,
                                Operands            : STRING
                                        ];
        I_FRAME :               TYPE = RECORD [
                                IDEA_PDU : I_PDU,
                                IDEA_MESS : I_MESS
                                        ];

END.
```

References

[1] Working group 2 of project R1003 "GUIDELINE",*"An Architecture for the Telecommunications Management Network"*, Sept. 1991.

[2] J.P. Claudé et al., *Unification of Heterogeneous Management by a Generic Object Oriented Agent"*, TMN Conf, Nov. 1990.

[3] ISO/IS 9595, Information Technology - Open Systems Interconnection, Common Management Information Service Definition, July 1991.

[4] M. TAG, *"Developpement D'Applications Reparties Sur le Systeme Distribué Orienté Objet CSA"*, Doctorat of the University Paris VI, June 1989.

[5] ISO/IS 9596, Information Technology - Open Systems Interconnection, Common Management Information Protocol Specification, July 1991.

[6] Advanced Networked Systems Architecture, *"ANSA Reference Manual"*.

[7] Deliverable 17, *"Description of the INDB"*, PEMMON Project, May 1992.

[8] Deliverable 18, *"Documentation of the INDB"*, PEMMON Project, Nov. 1992.

[9] OSI/Network Management Forum : FORUM 003, *"Object Specification Framework"*, Issue 1.0, September 1989.

[10] OSI/Network Management Forum : FORUM TR 102, *"Modelling Principles for Managed Objects Technical Report"*, Issue 1.0, January 1991.

[11] OSI/Network Management Forum : FORUM 007, *"Managed Object Naming and Addressing"*, Issue 1.0, May 1990.