Experience with the Interoperability between Telecommunications Management Network Systems (TMN)

Mauro OLIVEIRA(*), Nazim AGOULMINE, Neuman DE SOUZA(*), J-P.Claude
barbosa@masi.ibp.fr

Laboratoire MASI (UA 818 CNRS)
Universite Pierre et Marie Curie
45, Avenue des Etats-Unis
78000 - Versailles FRANCE

Abstract

Interoperability in TMN context is the ability of the network management products and services from different suppliers to work together to manage communication and computer networks. The interoperability problem between different TMNs is very complex.

This paper treats the interoperability aspects of the inter-TMNs communication. It presents a description of the first prototype of the inter-TMNs communication that has been developed in the ADVANCE Project of the RACE European Programme.

Key words: TMN architecture, interoperability, OSI/NM Forum, ODP viewpoints, ANSA System, ISODE, ASN.1 notation.

1. INTRODUCTION

Telecommunication Management Network (TMN) is the term commonly used to indicate the computer based system which supports all the operations of network management in a telecommunication environment [1]. TMN functions are concerned with the transport and the processing of the information related to the management of a single Network Element (NE) or the whole telecommunication network, in special the IBCN (Integrated Broadband Communication Networks).

IBCN due to its physical nature will be naturally distributed and will exhibit a genuine parallelism between its components. IBCN Management Applications are likely to be distributed as a direct consequence of the natural distribution of the IBCN [2]. The IBCN management system may well consist of a number of TMNs (possibly under different control), which in turn consist of a number of network Management Centres.

The ADVANCE Project is one of the CEC RACE Programme. The ADVANCE architecture was developed with two main goal in mind. The first was to provide a framework which supported the design and test of network management functions in form of Management Applications. The second was the definition of Telecommunication Management Network Computer Platform (TMNCP). The ADVANCE's long term study concerns the issues associated the options for implementation of TMNCP over a wide area [3].

(*) Authors supported by ETFCe/ CNPq and BNB/CNPq

This paper describes the work undertaken within this long term study. It treats the interoperability aspects of the inter-TMNs communication and describes the approach we have taken within the ADVANCE Project.

In the next section, we describe the architecture of the TMN introduced by the CCITT. Afterwards, section we identify how the interoperability of TMNs can be achieved and we propose an architecture of the functional block that permit the interoperability between TMNs called CME and the mecanisms negociation between the TMNs. In the last section we present a prototype that has been developed.

2. THE INTEROPERABILITY PROBLEM

In the real context of communication there is a set of organizations with there own equipment, application and politic. Each organization has generally a set of networks connected together to provide communication service inside and outside the organization. In each organization, we find different levels of heterogeneity : technology, technique, etc.

In each organization we find one or several management systems that support the management of the organization' networks. But each management systems permit only to manage a sub-set of the entire networks. In order to provide an overall management their is a need to cooperate the differents management centers by permiting them to interoperate

The Interoperability term is defined in [4] as the ability of the network management products and services from different suppliers to work together to manage communication and computer networks. In particular, a product from one supplier must be able to manage or be managed by a product from a different supplier, or act in both of these roles.

The interoperability between different TMNs is complex due to the management policy of each organization, the security aspect and the heterogeneity of the information and the underlying networks' architecture and resources. If the problem of communication between heterogeneous networks were resolved, the problem of cooperation between manufacturers administration systems would still exist.

2.1 TMN Functional Components

A TMN is used to manage and control services, customer and networks, then, the initial situation supposes, also, several customers of each one of the organizations, several interconnected networks through several accesses, and several organizations working together to provide and consume services and to manage and control both services and networks. This means that each one the organization has connected its TMN system to its networks (public, PABX, LAN, MAN, etc.) and uses its TMN system to manage and control both its networks, the services provided through it and the customer which use them.

The figure 1 show the principal components (function blocks and reference points) of the TMN Functional structure [1,5]:

(i) **Functions blocks:**

- The Operation Systems Functions (OSF) enabling the management of the Managed Network Element Functions (MNEF);

- The Work Station Functions (**WSF**) enabling the user to interact the OSFs;
- The Mediation Function (**MF**) used for transfer of information between OSFs and MNEFs.

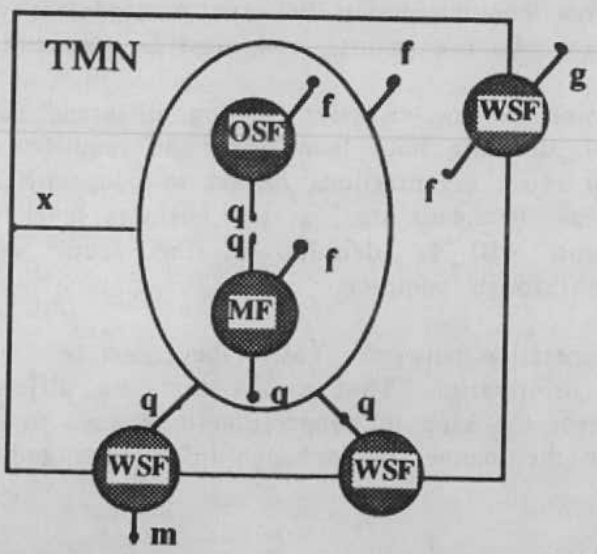

Fig 1 - The TMN Functional architecture

(ii) References points:

- The **q** reference points connect the functions blocks for Network Element Function (NEF) to MF, MF to MF, MF to OSF and OSF to OSF either directly or via the Data Communication Function (DCF);
- The **f** reference points connect OSF to WSF;
- The **g** reference are points between the WSF and the user;
- The **x** reference points connect a TMN to other management type networks including other TMNs.

The basic idea behind a TMN is to provide an organized network structure to achieve the interconnection of the various types of **OS**. The **OSF processes** information related to telecommunication management to support and control the realization of various telecommunication management functions. There are at least three functional types of OSFs (i.e. basic, network and services). Basic OSFs and Network OSFs share infrastructure aspect of a telecommunication network. Service OSFs are concerned with service aspects of one or more telecommunication networks.

There is a vertical hierarchical communication on the one hand within a TMN, from a lower layer **OS** to the nest higher layer **OS**, and there is horizontal peer to peer communication involving management center to management center within a given **OS** layer on the other hand. The vertical communication is associated with the **q reference points** and represents internal communication, while the peer to peer communication is external with the **x reference points.**

2.2 Interoperability Requirements

To achieve a global network management activity, there is a need to permit the cooperation between different network management systems. This is realized by a TMN to TMN communication which is necessary to effect a global integral view at the international level. This communication between management centers must take place over the X interface (x reference points), and must be standardized.

The inter-relations characterization among different cooperating TMN system begin at highest level, defining both boundaries and requirements of each one of the TMN systems: two or more organizations decide to cooperate among them to improve their respective business. It means that, at the business level, the requirements of the inter-TMNs cooperation will be decided in the scope of an inter-organization cooperation and agree through contracts.

Before any cooperation between TMNs they must be some agreement about the manner to exchange information. That means that the different organization should agree at the politic level the kind of cooperation they want to do. After that they must be an agreement about the manner to exchange information and the manner to represent it.

When the TMNs interoperate, it is necessary for each TMN to have an understanding of the management functions supported or required by the others. The OSI/NM architecture addresses the problem of interoperable network management and presents a framework for describing the various aspects of the problem and the solutions [6]

3. A SOLUTION FOR THE INTEROPERABILITY

After highlighted some issues about the interoperability, we present in this section the architecture of a component to be included in a management system in order to be interoperable.

The approach presented is based on the OSI/NM FORUM [7] work. Two important concepts were introduced in the Forum :

- **Interoperable Interface**: the interoperable interface is a set of protocols, procedures, message formats and semantics used to communicate management information between management systems.

- **Conformant Management Entity**: the CME is an open management system that supports the interoperable interface. Thus, two CMEs communicate across the interoperable interface.

The work undertaken by the OSI/NM FORUM gives some information about the manner to permit to a management system to be interoperable, but it don't give any details about the internal architecture of the management systems. This is the point we are trying to resolve in this paper by proposing an architectural approach for the interoperability.

3.1 The CME component

The proposal here is to extend the existing architecture of the ADVANCE platform [15] by adding a new functional block called CME (Cooperation Management Entity).

The CME will be responsible for the support of the cooperation between the different TMNs. Each TMN is realized on the ADVANCE platform. This definition of the CME (functional block) is different from the NM/FORUM definition. In the NM/FORUM, the CME is the management system itself [7].

Two aspects of the interoperability are identified in the CME proposed in this work:

- **Information Aspect:**

  The information aspect is concerned by all the aspect of information exchange. The information to be exchanged between the different TMNs must be modelised. The TMNs will cooperate with each other by exchanging a set of information for a variety of purposes while at the same time they must maintain a high degree of autonomy.

- **Communication Aspect:**

  The communication aspect between the different TMNs : When the TMNs are cooperating with each other they do so by means of associations (OSI meaning) established between them. The protocol used and the message to be exchanged must be a shared knowledge between all the partners before any cooperation can be achieved. An organizational model must also be defined, determining the role of each partner in the cooperation.

Therefore, the CME is composed of the two components (or layers): Information Entity Block (IEB) and Communication Entity Block (CEB), shown in the figure 2.

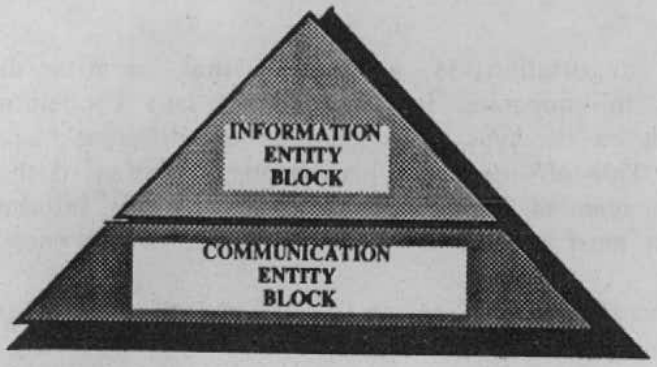

Fig 2- CME Architecture

3.2 The Communication Entity Block

The Communication Entity Block (CEB) of the CME provides such a support for the communication. This block permits to hide the complexity of naming and addressing the target ADVANCE platforms. This is done by the X.500 directory [8] .

The CEB component is an Engineering Object that provide inter-domains communication capability to the CME, to offer a location transparent and access transparent to the communication between Management Centers, in the same or in different TMNs.

The architecture of the management centers (ADVANCE Management System) is based on the ADVANCE Platform. The differents TMNs communicate to permit the exchange of information necessary for the cooperation. We have agree the standard protocol and service CMIP [9] and CMIS [10] as support for the management information exchange between TMNs.

3.3 Information Entity Block

The Information Entity Block (IEB) is responsable for the effective realization of the cooperation. This block uses the service provided by the CEB to communicate with a different IEB. This block provides a service of the Context negotiation described below.

To participate a in cooperation, the different ADVANCE Management Systems must be capable of performing the manager role, the agent role or both as defined in the OSI system management. A Management System performing the agent role makes managed objects visible to other Management Systems by receiving operations and issuing notifications. The definition of these responsabilities permit to make clearer the behaviour of each management system when cooperating.

3.4 Context Negotiation

An One important point in the cooperation between the Management Systems is to define which information can be shared or exchanged them. Each Management System has its set of management information about the users, the services and the network it supports. Each one of these system constitutes the basic element of the cooperation and has individual information that wish to share and exchange to satisfy a certain type of a global management .

The context negotiation is a process that permits different ADVANCE Management Systems to cooperate. It clearly before any cooperation may occur there must be an agreement on the type of cooperation the different Management Systems are willing to have [11]. This off-line phase permits the definition of the type of protocol to use, the type of management class which exists, etc. This information constitutes the shared knowledge that must be known by every partner of the cooperation.

The other process which is an on-line process of the context negotiation occurs at the beginning of an association . This context differs depending on the requirements of management applications or policy, etc.

Two schemas must be defined while a Management System want to participate in a cooperation. The first, one called Export schema, defines the information that the Management System is exporting to the other Management Systems and the import schema that the Management System (depending on management applications interests) is willing to import from the different partners of the cooperation. The export schema and the import schema define the cooperation context.

After this phase the two management systems can cooperate by exchanging management informations or providing services to each other depending of the role there were assigned.

4. PROTOTYPE

The previous section outlined requirements for cooperation among different TMNs and described an architecture of a component to add to the ADVANCE platform [15] to meet some of those requirements. In order to validate the concept and software structures, a first prototype of the ADVANCE platform including the CME component was developped to meet these requirements [20].

4.1 Scenario

The scenario (figure. 3) adopted for the first prototype is the cooperation between two ADVANCE Platforms, one performing the role manager and the other one the agent manager. The goal of this prototype is to realize the cooperation between the two ADVANCE Platform's components by enabling to a management application supported by the manager platform to access to management information represented by managed object instances the agent platform. This cooperation will be realized through the CMEs functional blocks of the ADVANCE ADVANCE Platform.

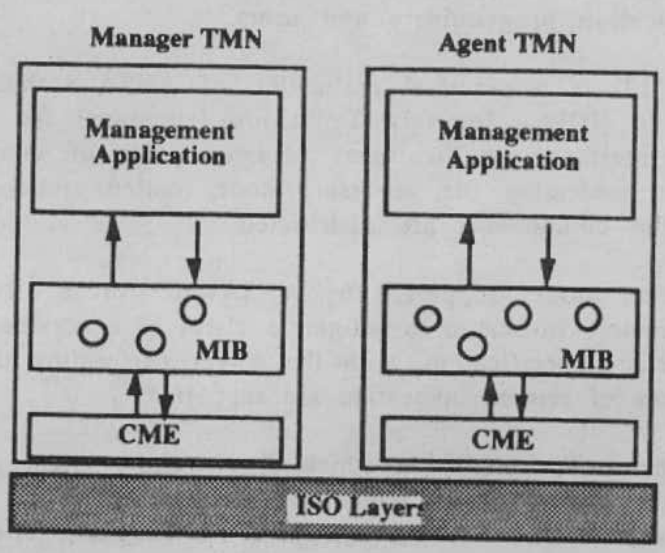

Fig 3 - Communication between TMNs

The scenario is composed by three phases (figure 4) :

- **The first phase:**

Each ADVANCE Platform will support a certain type of instances of managed object class. In the first phase is the cooperation context negotiation where the manager platform sends its import schema through the CME to the agent platform.

- **The second phase**

The second phase consist to share management information between the agent management system and the manager one. The management application supported by the manager system sends requests to the agent system through the CME.

- **The third phase**

The third phase corresponds to the dynamic cooperation context negotiation by changing the export schema of the agent platform and/or the import schema of the manager platform.

4.2. Support technology

The implementation, is based on two environments which are presented below :

- **ANSA distributed System:** Each management system is realized by an ADVANCE Platform. The distributed mechanisms are provided by ANSA System.

- **ISO Environment Development:** This environment provides rge necessary services that permit the peer to peer communications between management systems of different TMNs.

4.2.1 ANSA distributed System

a- Overview

The scope of the Advanced Networked Systems Architecture (ANSA) is to provide an architecture for distributed system. It is an architecture for building distributed systems that can operate as a unified whole such that the fact of distribution is transparent to application programmers and users.

ANSAware [12] is a practical realization of ANSA architecture. It provides a specification language (IDL - Interface Definition Language) for defining the sets of legal interactions (interface specifications) between pairs of components and a tool (stub compiler) for generating the necessary code (called stubs) which facilitate the interactions when the components are distributed.

The interaction model supported by ANSAware forces all interactions between components into a remote invocation paradigm; a client of a service invokes an operation defined in the interface specification, with the server performing the operation at some later time. Two forms of remote invocation are supported:

- Interrogation, (synchronous mode) in which the invoking client activity waits for the server to perform the operation and return any results;
- Announcement, (asynchronous mode) in which the invoking client activity does not wait for the server to perform the operation.

ANSAware supports concurrency through the provision of a threads[1]

b- Application programmer viewpoint

- **Operation signatures in IDL**

All interactions between objects in ANSAware are based upon interface specifications written in IDL for particular services; they are the only information upon which the interactions are based.

---

[1] A Thread is an independent execution path through a sequence of operations within a capsule (the unit of autonomous operation within ANSAware); from the point of view of the programmer, it represents the unit of serial activity. A thread performs one logical activity within a capsule, but threads can share data structures and can synchronise with each other at significant points.

A specification written in IDL has the following general structure:

```
-------------------------------------
TypeName : INTERFACE =
IS COMPATIBLE WITH TypeName1;
NEEDS TypeName2;

BEGIN
-- interface-specific constructed
data types
-- operation signatures
END.
-------------------------------------
```

TypeName, TypeName1 and TypeName2 are the interface type names that are used by the trading system[2].

IS COMPATIBLE WITH and NEEDS are optional statements used to avoid the rewriting process of types and operations already defined in others interface specifications.

Each operation in an interface has a name, an argument list (possibly empty) and a result list (possibly empty). Arguments and results are specified by position. Arguments must be given formal parameter names; results can be named, if desired. Arguments and results are passed by value. Consider the following examples:

```
BasicType : TYPE = [ INT, STR ];

ValueType : TYPE = CHOICE BasicType OF [
INT => INTEGER,
STR => STRING
];

Get : OPERATION [ instance: STRING; attribute: STRING ]
RETURNS [ INTEGER; STRING; ValueType ];

AddInstance : INTERROGATION      OPERATION [
        instance : STRING;
        domain : STRING;
        class : STRING;
        superior : STRING
        ]
RETURNS [ INTEGER; STRING];

Synch : ANNOUNCEMENT OPERATION [ ]
RETURNS [ ];
```

An invocation of Get generates an interrogation (by default), passing the values of STRING type and returning an integer, a string and a ValueType. An invocation of AddInstance proceeds in the same way, with passage of parameters and the return of results.

---

[2] A trading system allows clients to find servers dynamically via a system-wide directory structure for recording and determining the availability of services. The architectural service which provides trading is implemented in ANSAware as an object called the Trader.

- **Operation invocation using PREPC language**

The actual program source is written using PREPC[3] statements. All PREPC statements begin with an exclamation mark ("!") in column 1 of the source program.

The generic syntax for an operation invocation is:

```
!     [ results ] <- ref$op (arguments) exceptions
```

where results is a comma-separated list (possibly empty) of results to be returned, ref is an InterfaceId (a well-know reference or a reference received as an invocation argument or result), op is the name of an operation to invoke in that interface, arguments is a comma-separated list (possibly empty) of arguments, and exceptions is a statement (possibly empty) of actions to take under specified exception conditions.

c- Aspects at the engineering support viewpoint

The network communications model used by ANSA and implemented in ANSAware is based on remote procedure calls (RPC). A clear separation is maintained between the programming language aspects of RPC (which feature in the computational model), the service primitives to the RPC protocol and the design of the protocol itself. This is to permit alternative language representations of RPC and to enable the protocol to be operated over widely differing kinds of network Communications in the ANSAware engineering model is divided into three layers:

a) at the bottom are a number of Message Passing Services (MPS) that manage connection and disconnection, and the transmission an receipt of messages between nodes;

b) above them are Execution Protocols that map computational model invocations onto message exchange through the message services. The REX (Remote Execution Protocol) is a single execution protocols included in ANSAware, which is a protocol for single endpoint to single endpoint communication;

c) the third layer is made up by the Session objects which is responsible of the coordination between protocols and threads; a session represents local state about interactions with a remote interface; both the client and the server maintain session information during an interaction; the function of a protocol, in addition to transporting data is to maintain session state between the client and server session objects.

4.2.2 ISO Development Environment (ISODE)

a- Overview

ISODE is a non-proprietary implementation of some protocol defined by ISO/IEC. The purpose of making this software openly available is to accelerate the process of the development of the applications in the OSI protocol suite [13].

---

[3] The PREPC language provides a means for embedding invocations of interface operations in C source files. A compiler called prepc translates PREPC statements into invocations of the stub routines generated by stubc (another compiler provided for automatically generating stub code for marshalling and unmarshalling the data types specified by the IDL interface and for dispatching the appropriate service operation on receipt of a operation invocation).

This software can support different network services below the transport service access point (TSAP). One of these network services is TCP. This permits the development of the higher level protocol in an internet environment. However, the software also operates over pure OSI lower levels of software.

- **Remote Operation**

Remote Operation is a popular technique for building distributed applications. The Remote Operation Services Elements (ROSE) is responsible for request and reply interaction. OSI provides a powerful notation (RO-notation), for specifying the external interaction of these systems. Then, to facilitate the development of the application using ROSE, the ASN.1 description of ROSE provide macros (RO-notation).

An objet model for programming follows the use of abstract data types rather than concrete data structure. The use of **operations**, rather than direct manipulation, provides an important level of indirection: data structure may be accessed without regard to their local implementation.

An **operation** is a simple request or reply interaction. It is invoked by:

- an operation number
- an arbitrary complex argument
- an invocation identifier

Once an association is established, the initiator requests the responder to perform remote operation. An association-descriptor is used to reference the association. This is usually the first parameter given to any of the remaining routines.

When a request to perform a remote operation has been received by the responder to an association, the responder either returns a result or an error.

- **Naming & addressing**

A collection of remote operations form only a partial definition of a service. Other parts of this definition include naming and addressing information:

- **abstract syntax:** this describe the data structure being exchanged by the services;
- **application context name:** this describe the protocol being used by the service;
- **application-entity information:** this uniquely names an entity in the network;
- **presentation address:** this locates an entity in the network; and,
- **local program:** this identifies the program on the local system which implements the services.

The ISODE Entities Database

The database *isoentities* contains a simple mapping between application-entity information and presentation address. This database is used by the stub-directory service. The database itself is an ordinary ASCII text file containing information regarding the known application-entities on the network. This file replaces the "real" directory services.

The ISODE Objects database

The database *isobjects* contain a simple mapping between object descriptors and object identifiers. The database itself is an ordinary ASCII text file containing information regarding the known objects on the host.

The ISODE Services Database

The database isoservices contains a simple mapping between textual descriptions of services, services selectors, and local programs. The database itself is an ordinary ASCII text file containing information regarding the known services on the host. The steps involved to defining new services in ISODE are simple:

- The **application context** and **abstract syntax** for the services is registered in the *isobjects* file this allow to use the textual designator for these values rather the **object identifier** form;
- An entry is created in the **X500 Directory** containing information indicating where the service resides in the network, and, optionally, what UNIX program will be invoked whenever there is an incoming connection for the service.

b- The ISODE Application Cook-book

The ISODE has an interesting package called "The Application Cookbook" [14] which automates much tasks (code writing) involved in the implementation of a network application in the OSI environment which was used in the prototype. Then, it provides the utensils (programming tools) and various recipes (boilerplates routines) which can be used to cook-up OSI applications

• **The rosy Compiler**

The rosy (Remote Operation Stub-generator YACC-based) is a compiler for specification of the remote operations. The rosy program reads a description of a remote operation module and produce the corresponding C stubs and definition for use with the run-time environment. Operations are defined by a name starts with a lower case character followed by the keyword OPERATION:

- ARGUMENTS: defines the ASN.1 type which the operation expects as its arguments.
- RESULTS: defines the ASN.1 type which the operation returns on success
- ERRORS: defines the errors which the operation return on failure.

The rosy program will produce several files after reading its remote operation module:

- Abstract Syntax Module
- C Language Stubs (three separate stub files are produced by rosy)

• **The pepsy Compiler**

The pepsy is a new program that has been developed to replace both pepy (Presentation Element Parser YACC-based) and posy (Posy Optional Structure generator YACC based).

The pepsy program reads a description of an abstract syntax module and produces the corresponding C structures along with several tables that define the mapping between the ASN.1 objects in the module and the C structures. The pepsy program produces two files after reading its abstract syntax module:

- C languages Structures
- Mapping tables

4.3 Implementation

4.3.1 ODP Computational and Engineering Aspects

A prototype of the ADVANCE Platform has been already developed [15]. It is based on the ANSAware distributed environment. The goal here is to extend the existing architecture of the ADVANCE platform by adding a new functional block called CME (Cooperation Management Entity).

The communication between the CMEs will be realized using the ISODE facilities. The IEB block of the CME uses the Remote Operation for request or reply interactions and X500 directory to map the logical address of the ADVANCE Platform components with their physical one.

- **Computation Aspect**

Application Programmer's Interface:

This a primitive set used by architecture's components communicate to the Platform. It is the only mechanism that enables components of ADVANCE Platform to interact with each other. The functions available for the MA developers are: Deliver, Call, Cast, SrongCast, WeakCast, Reply, UserReject, Request, Collect and IncomingPDU.

This interface has been implemented in the ADVANCE Platform in two environments: Prolog, SPOKE [16] and 'C' [17]. It does not support transactions and message is assumed to have one, and only one target.

The CME Protocol Data Unit Structures.:

The exchanged messages between CMEs are embedded into structures called CME Protocol Data Unit (CME_PDU) [18]. Each CME_PDU contains the CME byte stream (CME message itself) and any necessary control information that must also be transferred with that byte stream. For the new scenario we have included a new field in the CME_PDU that identify the target TMN The CME_PDU has the following structure.

```
typedef struct        {
        int     PDU_Size;
        int     PDU_Type;
        int     ParsedOrNot;
        char    *TargetTMN;
        int     Context;
        char    *TargetName;
        char   *Source.Name;
        int     CMEBitstreamSize;
        CMEBitstream I_CMEBitstream;   } I_CME_PDU;
```

The Target TMN will denote a simple name for an end-system. It corresponds to a Distinguished Name (DN) for the use of the Directory.

- **Engineering Aspect**

The CME Delivery Mechanism (CME DM):

This component has the mechanisms to interpret the message and to deliver it to the target. its purpose is to transfer CME messages and their associates response between (the Visible Interfaces) the system's components (e.g. CIM, Management Application).

An essential part of the CME_DM is identifying the correct target for any messages, given a target name. The CME_DM is more discussed in [19]. We consider here the cooperation two TMNs A and B.

CEB A Acting in a Manager Role

This component in the TMN A, receives the CME message from the platform and send it to the target block in the TMN B (which is the target TMN) through the different ASEs (ACSE, ROSE, CMIS) provided by ISODE.

CEB B : Acting in an Agent Role

This component in the TMN B, receives the message from the CEB A (in the TMN A)

The CME of a particular TMN export its service to the CME_DM :

```
!       {er}<-traderRef$Export("CMEInterface",
"/CME","Classes{'Customer"Service' 'Network'} Relations {'is_provided_to'}", n).
```

The MAs can peruse this functional block CME for exchanging information with another functional block of a different TMN :

```
!       {ir} <-traderRef$Import("CMEInterface",
"/CME","'Customer' in Classes")
```

CMEInterface is the only interface in ADVANCE Platform. It supports the following operation:

CMEInterpreter: OPERATION [CMEMessage : String] RETURN[String]

- **Synchronous call and asynchronous cast**

A simple call and cast interactions are illustrated below (figure 5).

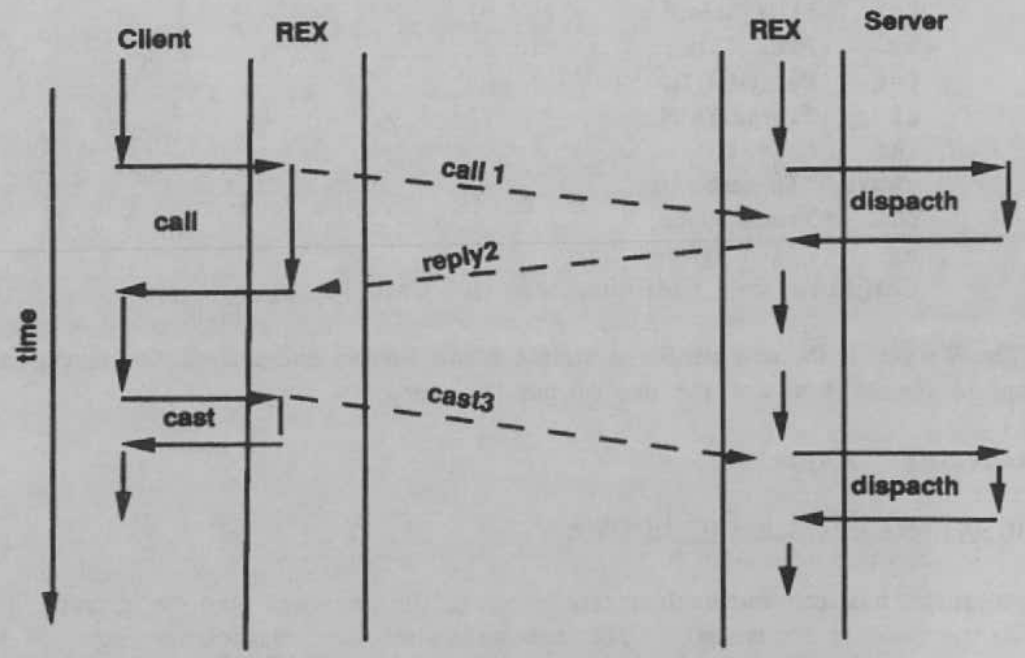

Fig 5 - Synchronous and Asynchronous Communications

4.3.2 Structure of the cme Programs

- **Defining the *cme* Service**

We begin by defining the naming and addressing information for the service:

abstract syntax: defined in the *isobjects* file as:

" isode cme pci " 1.17.1.4.1

application context name: defined in the *isobjects* file as:

" isode cme " 1.17.1.4.2

application-entity information & presentation address: defined in the *isoentities* file as:

" default cme" 1.17.4.0.13 #522

local program: defined in the *isoservices* file as:

"tsap/cmestore" #522 ros.cme

- **Remote Operations Module**

A *remote operation module* defines the operations (and associated errors) along the abstract syntax of the data description exchanged by the service. It is placed in cme.ry program [20]:

```
cmeisode DEFINITIONS ::=

BEGIN

-- operations
call OPERATION
        ARGUMENT    Message
        RESULT      Reply
        ERRORS      { reject, congested}
        ::=     0

cast OPERATION
        ARGUMENT    Message
        RESULT      Empty
        ERRORS      { congested }
        ::=     1

-- errors
reject ERROR ::= 0

- congestion at responder
congested ERROR::=1

-- types
Message ::= IA5String
Reply ::= IA5String
Empty ::=NULL
END
```

• INTERACTIVE INITIATOR

The interactive initiator of the service is placed in cme_i.c file. The main parts of this file is shown bellow:

```
•     CMEI.C PROGRAM
/*-----------------------------*/
/*  cme_i.c                    */
/*  cme  service  --initiator  */
/*-----------------------------*/

/* ARGUMENTS */
int  do_call(),  do_cast();

/* RESULTS */
int  call_result,  cast_result();

/* ERRORS */
int  cme_error  ();

/* DEFINITION OF THE  cme SERVICE */
char *myservice = " cme";
char *mycontext = " isode  cme";
char *mypcy = " isode  cme pci ";

main (argc, argv, envp)
{
/* GET THE INCOMING PDU FROM CME_DM (IPC QUEUE) */
InitialiseIPC(...);
DM_GetIncoming(...,PDU);

/* SET OF THE INCOMING PDU TYPE (REMOTE OPERATION) */
argv[2] = PDU->type;

/* CALLING THE ISODE GENERIC INITIATOR */
ryinitiator  (argc,argv,myservice,  mycontext,  mypcy,...);
}

/* ARGUSED */
do_call(...);
do_cast(...);

/* CALL RESULT */
call_result(sd,...,result,...)
{
char  *cp;

/* CONVERT THE qbuf (BUFFER-QUEUE FORM) INTO ONE STRING */
cp  =  qb2str(result);

/* SENDING THE INCOMING REPLY TO CME_DM (IPC QUEUE)  */
DM_Reply(cp,...);
}

/* CAST RESULT */
cast_result(...);

/* ERRORS */
cme_error  (...);
```

• RESPONDER

The responder of the service is placed in cme_r.c file. The main parts of this file is shown bellow:

CMER.C PROGRAM

```
/*-----------------------------*/
/*   cme_r.c                   */
/*   cme  service  --  responder  */
/*-----------------------------*/

/* OPERATIONS */
int  op_call(),  op_cast();

/* ASSIGNING THE  cme SERVICE */
char *myservice = " cme";
char *mycontext = " isode   cme";

main(argc,  argv,  envp)
{
/* CALLING THE ISODE GENERIC RESPONDER */
ryresponder(argc,  argv,  host,myservice,  mycontext,          dispatches,...);
}

/* CALL OPERATION */
op_call(sd,...,in,...)
{
char  *cd;
char  *res;

 /* CONVERT THE STRING INTO ONE qbuf (BUFFER-QUEUE FORM) */
cp  =  qb2str(in);

/* DO SOMETHING WITH cp HERE. FOR EX. RESULT = "reply" */

/* CONVERT THE qbuf (BUFFER-QUEUE FORM) INTO ONE STRING */
res  =  str2qb("reply");

/* IT RETURN A RESULT TO AN INVOCATION */
RyDsResult(sd,  ...,res,...);
}

/* CAST OPERATION */
op_cast(sd,...,in,...);
{
char    *cd;
cp  =  qb2str(in);

RyDsResult(sd,  ...,NULL,...);
}
```

The ryinitiator.c (generic interactive initiator) and ryresponder.c (generic idempotent responder) routines are shown in the Appendices.

5. CONCLUSION

This prototype constitues the first phase of the experimentation of the interoperability between TMNs within the ADVANCE Project. It has permitted to handle some problems about the communication aspect and to identify requirements at the informational level

The scenario used was the realization of a cooperation between a manager management system and an agent one. Both are supported by the ADVANCE platform. Each platform support a CME functional block that realize the X interface between the conceptual TMNs that include the management systems.

The next step of this work is to handle the informational aspects we have identified such as the problem of the translation of CME Message to CMIS Message and CMIS Message to CME Message. This aspect can be generalized as the problem of protocol and service translation and a specially in the case of interoperability as a translation of a proprietary service and protocol to normalized CMIS and CMIP service and protocol.

6. REFERENCES

[1] CCITT M30 Recommendations "Principles for a Telecommunications Management Network"

[2] 09/BCM/RD3/DS/C/002/A1. Initial User Requirements. Broadcom Eireann Research Limited. Dublin. December 9,1988.

[3] ADBC0735.MSW. Long Term Study Statements. Broadcom Eireann Research Limited. Dublin. May 2, 1991.

[4] Information Processing Systems - Open Systems Interconnection, Systems Management: Overview, 2nd DP10060

[5] An Implementation Architecture for thr TMN - CEC RACE Programme. Project ADVANCE R1003. Docment 03/CAS/SAR/DS/B/001/b1. Dublin. Dec 88.

[6] OSI-NM/FORUM - Forum 003, Forum Architecture , Issue 1 June 1989"

[7] OSI-NM/FORUM, Forum 009 "Shared Management Knowledge", Issue 1,1990

[8] "X500: The Directory", Overview of Concepts, Models and Services Recommendation X.500, CCITT Blue Book, Volume VIII - Fascicle VIII.8. 1988.

[9] Information Processing System, Open Systems Interconnection, Management Information Protocol Specification, CMIP - IS9596

[10] Information Processing System , Open Systems Interconnection, Management Information Service Definition - CMIS - IS9595

[11] Agoumine N., Oliveira M. A Solution for the Cooperation of TMN. Globecom/IEEE. Orlando, 1992.

[12] ANSAware 3.0. Implementation Manual, Doc:RM.097.00, Cambridge, U.K.,1991.

[13] Rose M.T. "The OPen Book - A Pratical Perpective on OSI". Prentice Hall. New Jersey (USA). 1990.

[14] Rose M.T. "The ISO Development Environment: User's Manual (version 7)" -Volume 4: The Applications Cookbook. Palo Alto (USA). 1991.

[15] Oliveira M., De Souza N., Penna M., Celestino J. Uma Plataforma de Computação para Administração de IBCNs. X SBRC. UFPE. Recife (Br), 1992.

[16] D.Harkness. CP SIG Contribution. Roke Manor Research Ltd. Sep91

[17] Oliveira M. ADVANCE Project - ADDN033. The Visible Interface for 'C' Application Programmer (The User Guide & Installations Notes). october91.

[18] Harkeness D. The ADVANCE Platform Computing Platform: open issues & Roke Manor contributions ADPL157. February 1991.

[19] M. Penna & al. ADVANCE Project - ADDN016 doc. The locating Problem in the CME Deliver Mechanism. March 91.

[20] Oliveira M, Agoulmine. ADVANCE Project, ADDN45 doc. A Prototype for Inter-TMN Cooperation.

APPENDICE I

```
/*-----------------------------------------*/
/*RYINITIATOR.C                            */
/*GENERIC INTERACTIVE INITIATOR*/
/*-----------------------------------------  */
ryinitiator (argc,argv,myservice, mycontext, mypcy,...)

{
/* CONTRUCTING A PRESENTATION ADDRESS*/
aei = str2aei(...,myservice,...);
pa = aei2oid(aei);

/* MAPPING BETWEEN OBJ DESCRIPTORS AND OBJ IDENTIFIERS */
ctx = ode2oid(mycontext);
pci = ode2oid(mypci);

/* ASSIGNING THE REMOTE OPERATION (INCOMING PDU TYPE) */
ds->ds_name = argv[2];

/* A-ASSOCIATE.REQUEST */
(ctx,...,aei,...,pa,...,acc);

/* ASSIGNING THE ASSOCIATION-DESCRIPTOR */
sd = acc -> acc_sd;

/* SELECTING AN UNDERLING SERVICE */
RoSetService(sd,...);

/* INVOKING OPERATIONS *:
invoke(sd,...,ds,...)
{
/* IT PROVIDES AN ASYNCHRONOUS INTERFACE */
RyStub(sd,...,ds->ds_operation,.
                ds->ds_result,...);
}
```

APPENDICE II

```
/*--------------------------------------*/
/* RYRESPONDER.C                        */
/* GENERIC RESPONDER                    */
/*--------------------------------------*/

ryresponder(argc, argv, host,myservice, mycontext, dispaches,...)

{
/* RETURNS THE APPLICATION-ENTITY INFORMATION STRUCTURE */
aei = str2aei(host, myservice,...);

/* ASSIGNING THE DISPATCH STRUCTURE USED BY THE INITIATOR*/
ds = dispatches;

/* IT REGISTERS A HANDLER FOR ANOPERATION */
RyDispatch(...,ds->ds_operation, fnx,...);

/* FNX: THE ADDRESS OF A DISPATCH ROUTINE TO BE INVOKED */

/* GENERIC SERVER DISPATCH */
isoserver(argc, argv, aei, ros_init, ros_work,...);
}
ros_init (...)

{
int cd, result;

/* A-ASSOCIATE.INDICATION */
AcInit(...,acs,...);

/* ASSIGNING THE ASSOCIATION-DESCRIPTOR */
sd = acs -> acs_sd;

/* A-ASSOCIATE.RESPONSE */
result = AcAssocResponse(sd,...);

/* SELECTING AN UNDERLING SERVICE */
RoSetService(sd,...);
}

ros_work(fd)
int fd;
{
int result;

/* USED TO WAIT FOR SOME EVENT TO OCCUR */
RyWait(fd,...);
}
```