CONCEPTION ET RÉALISATION D'UNE APPLICATION DE GESTION DE RÉSEAU À BASE DE COMPOSANTS RÉPARTIS

par

Mohamed Arezki

mémoire présenté au Département de mathématiques
et d'informatique en vue de l'obtention du grade de maître ès sciences (M.Sc.)

FACULTÉ DES SCIENCES
UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, juillet 2000

Le 17 juillet 2000 , le jury suivant a accepté ce mémoire dans sa version finale.
date

Président-rapporteur: M. Guy Custeau
Département de mathématiques et d'informatique

Membre: M. Gabriel Girard
Département de mathématiques et d'informatique

Membre: M. Michel Barbeau
Université de Carleton

Sommaire

Les applications actuelles de gestion des réseaux reposent essentiellement sur l'approche centralisée. Cependant, cette approche a atteint ses limites. Elle ne répond plus aux nouvelles exigences, telles que la tolérance aux fautes, la flexibilité et l'expansibilité des applications de gestion. Pour répondre à ces nouveaux besoins, plusieurs approches décentralisées ont été développées ; elles consistent à répartir des tâches de gestion sur plusieurs entités du réseau.

Ce mémoire propose une nouvelle approche décentralisée dans laquelle nous abordons le problème d'intégration dynamique de nouvelles fonctions à l'application de gestion, en utilisant les composants répartis et l'environnement distribué *Common Object Request Broker Architecture* (CORBA). Actuellement, il n'existe pas d'application de gestion capable de détecter et d'intégrer, au cours de son exécution, de nouvelles fonctions de gestion. Nous avons donc développé une architecture décentralisée afin d'illustrer le développement de ce type d'application.

Cette architecture est constituée de composants répartis qui utilisent la technologie CORBA pour communiquer entre eux. Ces composants accèdent aux informations de gestion des ressources du réseau à travers une passerelle CORBA/SNMP. L'architecture définit des *agents* CORBA pour représenter les ressources à gérer sur le réseau et définit des composants CORBA pour représenter les fonctions de gestion de réseau. Ces entités (composants CORBA) peuvent être branchées sur le réseau au cours de l'exécution de

l'application de gestion qui les détecte et les utilise automatiquement. Ce mécanisme est appelé *plug and play*.

Nous avons développé notre architecture sur une plate-forme Windows NT en utilisant le langage Java 1.2, l'API SNMP de AdventNet, le service de nom de Java 1.2 et OrbixWeb d'IONA.

Remerciements

Je remercie en premier lieu mon directeur de recherche, le professeur Michel Barbeau, pour ses précieux conseils et ses orientations durant toute ma maîtrise. Je le remercie également pour son aide financière qui m'a permis de me consacrer entièrement à mes études.

Un grand merci à Mme Taleb pour ses encouragements et à tout le personnel de *l'Université Libre de Tunis.*

Je tiens à remercier ma famille pour son appui et sa patience durant ces deux années de formation.

Je remercie enfin tous ceux qui m'ont aidé de près ou de loin dans l'élaboration de ce travail.

Table des matières

Liste des abréviations

AgentX	: Agent eXtensibility protocol
API	: Application Programming Interface
ASN.1	: Abstract Syntax Notation One
CIM	: Common Information Model
CMIP	: Common Management Information Protocol
CMIS	: Common Management Information Services
CMISE	: Common Management Information Service Element
CMOT	: Common Management Information Protocol over TCP
COM	: Component Object Model
CORBA	: Common Object Request Broker Architecture
DCE	: Distributed Computing Environment
DCOM	: Distributed Component Object Model
DCP	: Distributed Computing Protocol
DII	: Dynamic Invocation Interface
DME	: Distributed Management Environment

DP : Delegate Program

DSI : Dynamic Skeleton Interface

GDMO : Guidelines for the Description of Managed Objects

HMMP : Hyper Media Management Protocol

HMOM : Hyper Media Object Manager

HNM : Hierarchical Network Management

HTML : Hypertext Markup Language

HTTP : Hypertext Transfer Protocol

ICMP : Internet Control Message Protocol

IDL : Interface Definition Language

IETF : Internet Engineering Task Force

IIOP : Internet Inter-ORB Protocol

IP : Internet Protocol

IR : Interface Repository

ISO : International Organization for Standardization

JDMK : Java Dynamic Management Kit

JIDM : Joint Inter-Domain Management

JMX : Java Management eXtension

M2M : Manager to Manager

Mbd : Management by delegation

MIB : Management Information Base

MO : Managed Object

MOC : Managed Object Class

MRB : Management Request Broker

NMP : Network Management Procedure

OMG : Object Management Group

ORB : Object Request Broker

OSF : Open Software Foundation

OSI : Open Systems Interconnection

RFC : Request For Comment

RMI : Remote Method Invocation

RMON : Remote Network Monitoring

RPC : Remote Procedure Call

SNMP : Simple Network Management Protocol

TCP : Transmission Control Protocol

TINA : Telecommunication Information Networking Architecture

UDP : User Datagram Protocol

WBEM : Web Based Enterprise Management

XoJIDM : X/Open's Joint Inter-Domain Management task force

Liste des tableaux

Liste des figures

Introduction

La gestion est une activité essentielle de planification, d'organisation et d'administration des différentes ressources matérielles et logicielles d'un réseau. Elle est soutenue par des systèmes de gestion normalisés.

Les systèmes de gestion actuels reposent principalement sur l'approche centralisée. Cette approche consiste à acheminer toutes les informations de gestion de chaque ressource du réseau vers un point central. Le traitement et l'interprétation de ces informations au niveau central sont assurés par un programme appelé *gestionnaire*. Chacune des ressources à gérer contient un programme appelé *agent*, qui donne accès aux informations associées à la ressource. Le *gestionnaire* utilise le protocole *Simple Network Management protocol* (SNMP) ou *Common Management Information Protocol* (CMIP) pour collecter de l'information de gestion à partir de ces *agents*.

La gestion centralisée est confrontée à la réalité des réseaux actuels. Ces réseaux sont constitués de plates-formes hétérogènes et atteignent des tailles de plus en plus importantes. La gestion de ce type de réseaux à partir d'un point central devient vulnérable aux pannes qui peuvent survenir sur le réseau. L'approche centralisée ne peut pas assurer la résistance aux pannes en cas de problème sur le *gestionnaire*. Elle ne permet pas non plus la flexibilité et l'expansibilité des applications de gestion de réseau.

Pour surmonter ces problèmes, plusieurs approches ont été proposées. Principalement, l'approche distribuée, qui consiste à répartir les tâches de gestion sur plusieurs entités

du réseau. Cette approche a été favorisée par l'apparition des nouvelles technologies de la répartition, telles que CORBA, DCOM et RMI. Celles-ci améliorent la structure et la modularité des applications de gestion. Elles assurent leur portabilité et cachent la réalité hétérogène des systèmes de gestion.

Problématique

Notre problématique consiste à définir une nouvelle architecture modulaire, extensible et portable afin de faciliter le développement des applications de gestion dans un environnement distribué. Cette architecture exploitera la puissance de calcul des différentes machines du réseau ; elle assurera l'autonomie, l'évolution et l'ouverture des applications de gestion.

Résultats

Nous avons développé une application de gestion de réseau à base d'une architecture de composants répartis. Cette application utilise la technologie CORBA, pour la communication entre les composants répartis, et le protocole SNMP, pour la gestion des ressources du réseau. Elle est constituée d'un *gestionnaire* de réseaux et d'un ensemble de composants CORBA. Ces composants représentent les fonctions de gestion, les *agents* CORBA, l'agent *mandataire* et la passerelle CORBA/SNMP. Cette application est capable, au cours de son exécution, de détecter et d'utiliser les nouvelles fonctions de gestion qui viennent d'être lancées sur le réseau. Elle est également capable de détecter automatiquement la présence des *agents* CORBA sur le réseau. Ce mécanisme d'intégration dynamique des fonctions et des *agents* à l'application est appelé *plug and play.*

Pour l'implantation de cette application, nous avons utilisé le langage Java 1.2, l'API SNMP de AdventNet, le service de nom de Java 1.2 et OrbixWeb d'IONA.

Organisation du mémoire

Dans le premier chapitre, nous présentons l'architecture des applications et des systèmes de gestion des réseaux, de même qu'une comparaison entre les standards SNMP et OSI.

Dans le deuxième chapitre, nous décrivons les concepts des composants et des systèmes distribués, les approches décentralisées existantes et l'utilisation de CORBA dans la gestion des réseaux.

Dans le troisième chapitre, nous présentons la conception et la réalisation de notre application, la particularité de notre solution et les technologies disponibles pour le développement d'applications de gestion des réseaux.

Nous terminons par une conclusion, une proposition de travaux futurs et une présentation des perspectives de notre solution.

Chapitre 1

Concepts de la gestion des réseaux

La gestion des réseaux consiste à collecter des informations qui traduisent le comportement des différentes ressources du réseau. Ces informations sont analysées et stockées dans un format de données spécifique. Suivant le résultat de l'analyse, des commandes précises sont transmises aux ressources concernées. Le traitement de ces informations est assuré par des fonctions de gestion des réseaux [11].

1.1 Fonctions de gestion des réseaux

Les applications de gestion des réseaux consistent à planifier, à contrôler et à anticiper sur l'évolution probable d'un réseau, d'un protocole ou sur la possibilité d'une anomalie. Elles assurent également la correction de ces anomalies. Toutes ces opérations ont été normalisées et regroupées en cinq domaines fonctionnels de la gestion des réseaux par l'*International Organization for Standardization* (ISO) [44] :

– Gestion de la configuration

La gestion de la configuration consiste à surveiller, à repérer et à contrôler les ressources matérielles et logicielles disponibles sur le réseau.

– Gestion de la performance

Le but de la gestion de la performance est de surveiller le fonctionnement du réseau en permanence. Dès qu'un seuil est dépassé, une alerte est déclenchée et transmise au système de gestion du réseau. Elle assure également l'évaluation de la performance des composants matériels et logiciels du réseau.

– Gestion des pannes

L'objectif de la gestion des pannes est de détecter, d'isoler et de corriger les anomalies qui surviennent sur le réseau.

– Gestion de la sécurité

La gestion de la sécurité consiste à contrôler l'accès aux ressources et à assurer la confidentialité et l'intégrité de l'information transmise sur le réseau. Elle assure également l'authentification de l'émetteur.

– Gestion de l'information comptable

Le but de la gestion de l'information comptable est de mesurer les paramètres d'utilisation du réseau afin de gérer convenablement l'exploitation des ressources disponibles. Ces mesures peuvent être utilisées pour évaluer l'utilisation des ressources ou pour facturer l'exploitation de ces ressources.

1.2 Architectures des systèmes de gestion

Il existe trois approches de base pour les architectures des systèmes de gestion des réseaux [26].

1.2.1 Approche centralisée

Cette approche consiste à faire remonter toutes les informations de gestion de chaque ressource du réseau vers un point central qui les analyse et décide de l'opération à entreprendre. Chaque ressource est représentée par un programme appelé *agent*. Ce dernier communique les informations sur l'état de la ressource à la station de gestion (figure 1). Cette approche est utilisée dans l'architecture du protocole SNMPv1.

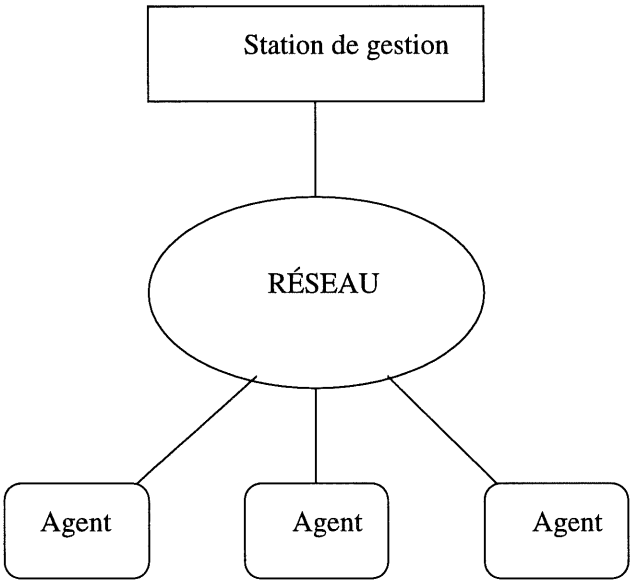

FIG. 1 – *Approche centralisée*

Dans l'approche centralisée, il existe une autre variante qui consiste à subdiviser le *gestionnaire* en deux parties : une plate-forme de gestion et des applications de gestion (figure 2). La plate-forme assure le regroupement et le traitement élémentaire des informations de gestion. Les applications de gestion utilisent les services offerts par la plate-forme pour réaliser les fonctions avancées de la gestion du réseau. La plate-forme rend transparente l'utilisation des protocoles de communication pour les applications de gestion.

Dans cette approche, les applications de gestion peuvent accéder aux informations de

gestion des ressources qui supportent des protocoles différents tels que SNMP ou CMIP.

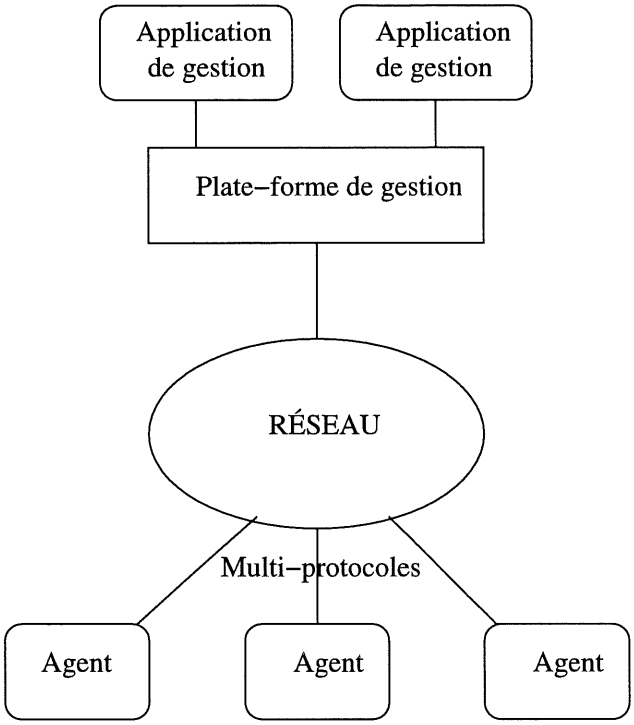

FIG. 2 – *Approche centralisée, basée sur une plate-forme de gestion*

1.2.2 Approche hiérarchique

Cette approche utilise le concept *gestionnaire des gestionnaires* (figure 3). Dans ce concept, chaque *gestionnaire* est responsable de la gestion de son domaine, lequel est constitué d'un ensemble *d'agents*. Les *gestionnaires* de domaine ne communiquent pas directement entre eux ; ils communiquent uniquement avec le *gestionnaire* central. Cette approche est utilisée dans l'architecture des protocoles CMIP et SNMPv2.

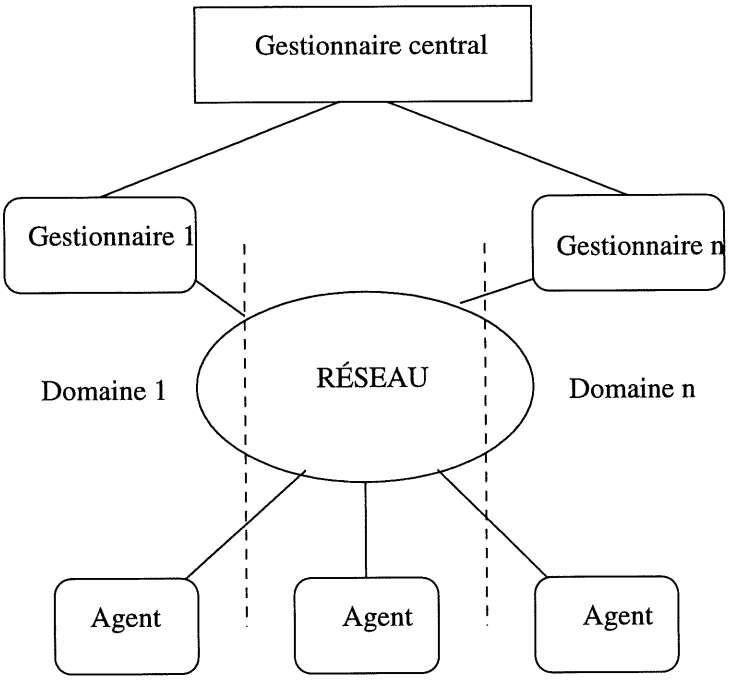

FIG. 3 – *Approche hiérarchique*

1.2.3 Approche distribuée

Cette approche est constituée de plusieurs *gestionnaires* de domaine indépendants qui communiquent entre eux pour s'échanger de l'information sur l'état du réseau. Chacun des *gestionnaires* est responsable de son propre domaine (figure 4). Cette approche permet d'augmenter la fiabilité et la performance des systèmes de gestion des réseaux.

Plusieurs travaux de recherche ont proposé de nouvelles approches pour les architectures distribuées des systèmes de gestion des réseaux ; certaines de ces approches sont abordées dans le chapitre suivant.

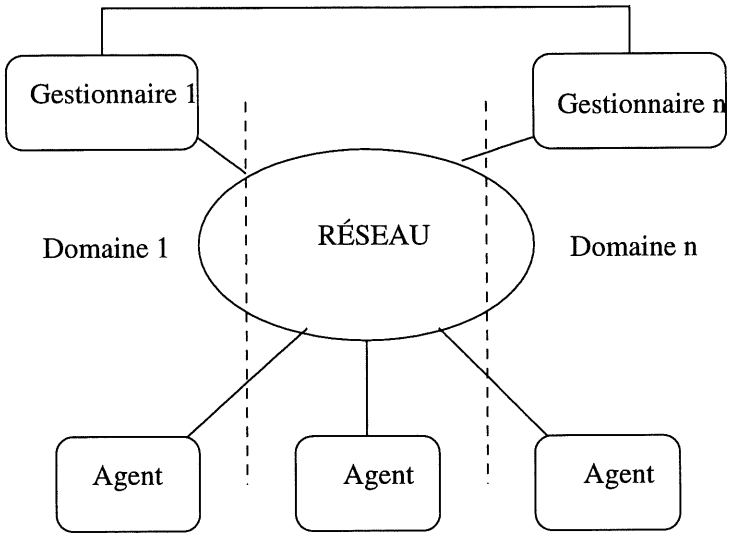

FIG. 4 – *Approche distribuée*

1.3 Structure des systèmes de gestion des réseaux

Un système de gestion est composé d'un ensemble de logiciels et de matériels qui assurent le bon fonctionnement du réseau. La gestion est réalisée par l'échange d'informations entre le *gestionnaire* et les différentes ressources matérielles et logicielles du réseau. Il existe deux principaux standards de systèmes de gestion des réseaux : *Simple Network Management Protocol* (SNMP) de *l'Internet Engineering Task Force* (IETF), et *Open Systems Interconnection* (OSI) de l'*International Organization for Standardization* (ISO).

Pour mieux décrire l'architecture de ces systèmes de gestion, nous décomposons chaque architecture en trois modèles distincts : le modèle informationnel, le modèle de communication et le modèle organisationnel [11]. Ces modèles englobent les parties essentielles d'un système de gestion des réseaux.

1.3.1 Modèle informationnel

Ce modèle fournit une vue du réseau par les données et structure l'information de gestion. Ces informations définissent les besoins de gestion des ressources matérielles et logicielles existantes sur le réseau. Elles sont stockées dans une base de données appelée *Management Information Base* (MIB).

Au niveau de la normalisation, l'ISO définit ces informations de gestion comme des objets stockés dans une MIB. Ces objets sont définis selon le sens du concept orienté objet. Par contre, l'IETF représente ces informations de gestion à travers des variables stockées dans une base de données virtuelle. Le modèle informationnel constitue la base sur laquelle reposent les deux prochains modèles.

1.3.2 Modèle de communication

Ce modèle décrit comment l'information de contrôle est acheminée entre les entités de gestion. Il fournit les moyens de recueillir des informations élémentaires ou statistiques auprès des *agents* représentant les ressources. Pour cela, l'ISO a défini le protocole CMIP et l'IETF a défini le protocole SNMP.

1.3.3 Modèle organisationnel

Ce modèle définit les entités de gestion qui échangent des informations de contrôle en utilisant le modèle de communication. Il repose essentiellement sur le concept de la relation *gestionnaire/agent*. Le *gestionnaire* et *l'agent* sont des processus qui échangent des informations de gestion à travers un protocole de communication. Chaque *agent* gère sa propre MIB sur laquelle le *gestionnaire* peut agir. Ce concept de *gestionnaire/agent* est repris par tous les organismes de normalisation.

1.4 Le protocole Simple Network Management Protocol (SNMP)

Le protocole SNMP est utilisé particulièrement dans les réseaux locaux et dans le monde Internet. Il définit essentiellement les règles de communication entre les entités de gestion d'un réseau (figure 5).

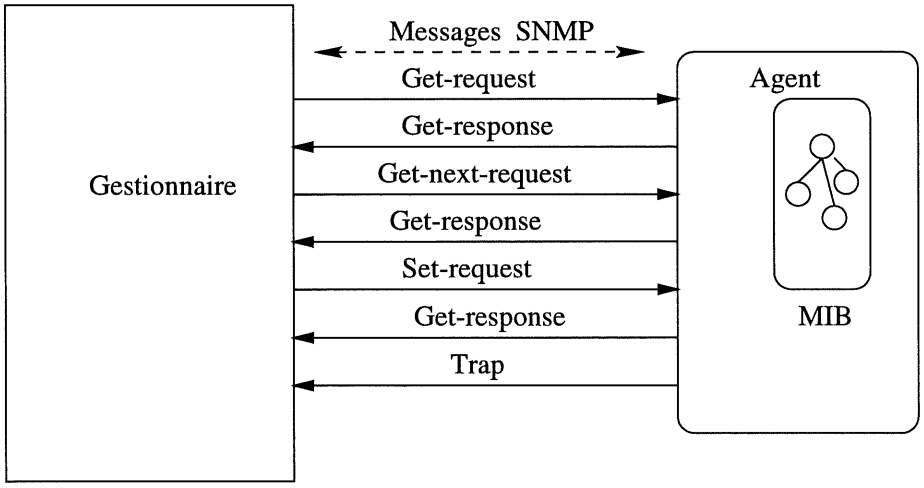

FIG. 5 – *Opérations SNMP*

1.4.1 Modèle informationnel

L'IETF représente ses informations de gestion à travers des variables et des tables stockées dans une base de données hiérarchique appelée MIB. Sa structure est définie dans RFC 1066 pour MIB I et dans RFC 1213 pour MIB II. Les éléments de la MIB sont appelés objets. Le terme objet utilisé par l'IETF est différent de celui utilisé dans le concept orienté objet.

La MIB II est une extension de la MIB I. Elle ajoute de nouveaux objets pour approfondir l'analyse des fonctionnalités d'un équipement. Parmi ces nouveaux objets, nous

trouvons les objets du groupe *SNMP* qui gère les performances du protocole de gestion, et le groupe système qui est doté de nouveaux objets, par exemple, *SysContact* qui spécifie le nom de la personne physique à contacter en cas de défaut irrémédiable et *SysLocation* qui renseigne sur la localisation de l'équipement.

Les objets gérés sont accessibles grâce à la MIB. La structure de ces objets est définie dans le langage de spécification *Abstract Syntax Notation One* (ASN.1) [36]. La définition d'un objet est limitée à un type simple ou à une table d'objets de type simple.Chaque objet possède un identificateur d'objet qui lui est propre. Les identificateurs ont une structure hiérarchique et arborescente (figure 6)

L'IETF utilise l'arbre d'enregistrement définit dans le norme OSI pour nommer ses objets [11]. Cet arbre est composé d'une racine à laquelle sont liés des noeuds marqués (figure 6). Chaque noeud de l'arbre est identifié d'une manière unique par un identificateur et un nom, selon le format suivant :

- Identificateur d'objet : 1.3.6.1.4.1.74.2.21
- Nom correspondant : *iso.org.dod.internet.private.enterprise.att-2.att-mgt.waveLan*

L'identificateur et le nom correspondent à la concaténation des nombres ou des noms qui relient la racine au noeud considéré. Chaque noeud peut avoir lui-même des fils qui sont aussi marqués (figure 6). Pour accéder à une valeur d'instance d'un objet non tabulaire, on ajoute un « .0 » à l'identificateur de l'objet de la variable. Par exemple, la variable *SysContact* est référencée par *SysContact.0*. Par contre, pour un objet tabulaire, on utilise le champ *Index* pour récupérer la valeur de chaque case de la table. Par exemple, pour accéder aux valeurs d'une table de deux lignes et deux colonnes appelée *tab*, on représente chaque ligne de données de la table de la manière suivante : *tab.index.val1.val2.* Si l'*index* est égal à 1, alors la valeur est égale à *val1*, et si l'*index* est égal à 2, alors la valeur est égale à *val2.*

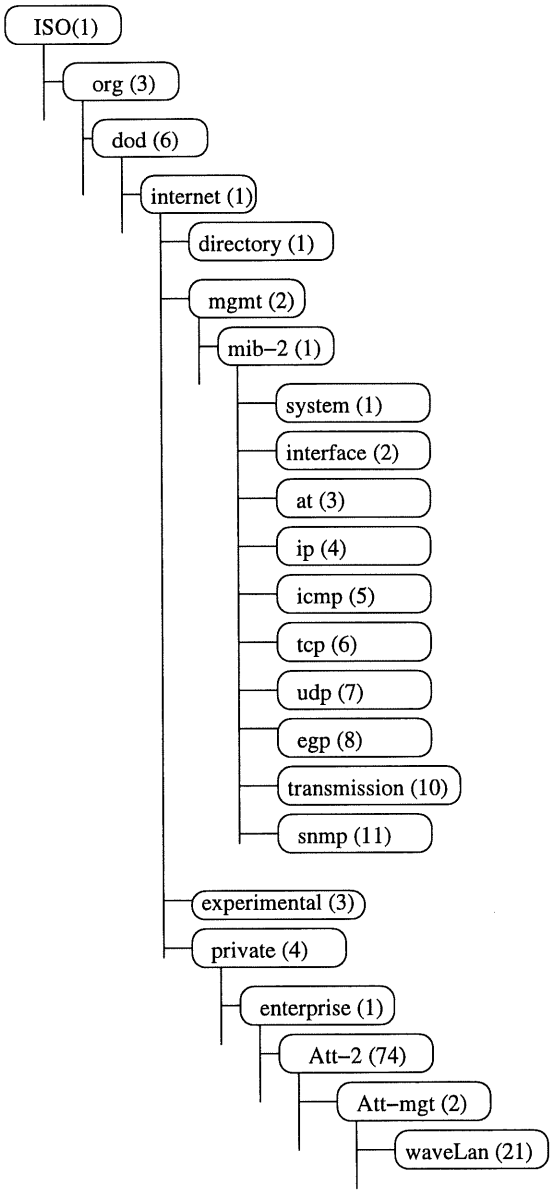

FIG. 6 – *Structure de la MIB SNMP*

Au niveau du sous-arbre *iso.org.dod.internet.private.enterprise* (1.3.6.1.4.1), des objets spécifiques peuvent être définis par des opérateurs privés (CISCO, ATT, NORTEL, etc.) pour gérer les spécificités de leurs équipements. Par exemple, le sous-arbre *waveLan* d'ATT représenté dans la figure 6 définit des objets spécifiques à la gestion des interfaces de *waveLan*. Celui-ci est une implantation de l'architecture des réseaux sans fil ; il permet de relier des ordinateurs entre eux sans avoir de connexion physique.

1.4.2 Modèle de communication

SNMP est le protocole de communication utilisé entre le *gestionnaire* et l'*agent*. Un *agent* est un programme opérant à l'intérieur d'un équipement à gérer (routeur, unité centrale, etc.). Par contre, le *gestionnaire* est un programme résidant dans une station de gestion. Le protocole SNMP fonctionne en mode non connecté en utilisant le protocole de transport UDP ; il détaille le format des paquets échangés.

La première version de SNMP définit cinq types de messages échangés entre le *gestionnaire* et l'*agent* (figure 5) :

- Get-request : obtenir la valeur d'une ou plusieurs variables.
- Get-next-request : obtenir la valeur d'une variable suivant une ou plusieurs autres variables.
- Set-request : définir la valeur d'une ou plusieurs variables.
- Get-response : renvoyer la valeur d'une ou plusieurs variables.
- Trap : signaler au *gestionnaire* un événement survenant chez un *agent*.

1.4.3 Modèle organisationnel

Ce modèle repose sur le concept *gestionnaire/agent.* Le *gestionnaire* collecte l'information de gestion auprès des *agents* à intervalles fixes et assure le traitement et l'interprétation de cette information. Ce mode de fonctionnement est appelé *Sollicitation* [1].

1.5 Évolution du SNMP

Afin de surmonter les limites de la première version de SNMP, l'IETF a mis en place SNMPv2, SNMPv3, AgentX et RMON pour répondre aux nouvelles exigences telles que la sécurité, l'autonomie et l'expansibilité des systèmes de gestion.

1.5.1 SNMPv2

Le protocole SNMPv2 assure les fonctionnalités de SNMPv1 [41]. De plus, il introduit deux nouveaux types de paquets: le *get-bulk-request* qui permet au *gestionnaire* de récupérer de grands blocs de données de manière efficace, et le *inform-request* qui permet au *gestionnaire* de transmettre de l'information à un autre *gestionnaire.* Deux nouvelles MIB sont définies: la MIB SNMPv2 et la MIBv2-M2M (gestionnaire vers gestionnaire). SNMPv2 fournit des améliorations pour la sécurité par rapport à SNMPv1. Avec SNMPv1, le mot de passe de la communauté SNMP échangé entre le *gestionnaire* et l'*agent* est un mot de passe non crypté. Par contre, SNMPv2 assure l'authentification et la confidentialité des messages.

Une des raisons pour lesquelles le protocole SNMPv2 n'a pas été adopté par l'industrie informatique est le désaccord entre les partenaires de l'IETF sur la définition des fonctions de sécurité.

1.5.2 SNMPv3

SNMPv3 a surmonté les problèmes rencontrés par SNMPv2. Il offre de nouvelles capacités d'ouverture et d'interopérabilité de gestion [6]. Les spécifications de SNMPv3 sont basées sur une architecture modulaire ; une entité SNMP(*gestionnaire* ou *agent)* est composée d'un moteur SNMP auquel on associe une ou plusieurs applications.

SNMPv3 fournit les services de sécurité tels que l'authentification, le cryptage des messages, le contrôle d'accès et la confidentialité. Il fournit également une nouvelle architecture modulaire et expansible. Il offre de nouvelles fonctions d'administration telles que la configuration à distance via les opérations SNMP.

1.5.3 AgentX SNMP

L'environnement Agent eXtensibility SNMP est une architecture distribuée de l'*agent* SNMP [30]. Cet *agent* est composé d'un *master agent* et d'un ou plusieurs *subagents* qui communiquent via le protocole AgentX. Le *master agent* supporte les protocoles SNMP et AgentX. Il maintient une table d'informations sur chacun des *subagents* et les régions de la MIB gérées par ces derniers. Le *subagent* est responsable de l'accès à l'information de gestion de la ressource. Il gère une région spécifique de la MIB.

Le protocole AgentX permet d'étendre dynamiquement la gestion des objets dans le noeud où le *master agent* réside.

Une application de gestion accède au *master agent* d'une manière transparente. Le *master agent* est considéré comme un *agent* SNMP classique.

1.5.4 RMON

Remote Network Monitoring (RMON) est un standard de gestion des réseaux [43]. RMON est en fait une sonde qui intègre un *agent* intelligent capable de capter de l'information circulant sur le réseau, et capable de l'analyser pour déterminer les actions à

entreprendre. La sonde assure également une analyse détaillée du trafic des données sur le réseau et ne communique au *gestionnaire* que les informations pertinentes.

Ces informations sont stockées dans une MIB RMON (figure 7). Elles sont structurées en plusieurs groupes : neuf groupes sont utilisés pour la gestion du réseau Ethernet (Statistics, History, Alarm, Host, HostTopN, Matrix, Filter, Packet capture, Event), et un groupe est utilisé pour la gestion des réseaux Token Ring.

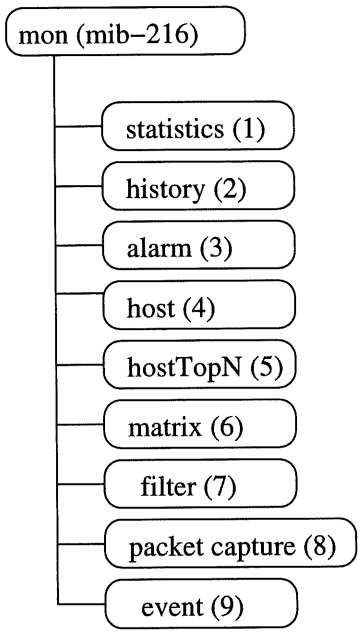

FIG. 7 – *MIB RMON*

L'utilisation de RMON dans la gestion SNMP est illustrée dans la figure 8.

La MIB RMON se concentre sur la gestion des réseaux, alors que les MIB I et II ont été développées dans le but de gérer un équipement de réseau particulier. La MIB RMON a évolué pour devenir RMON2 [22]. Ce nouveau standard consiste à établir des statistiques sur le trafic engendré par chaque type de protocole présent sur le réseau et sur le trafic généré par le dialogue entre des postes de travail et des serveurs désignés.

L'administrateur peut gérer la MIB RMON à partir d'une station de travail. Cela lui

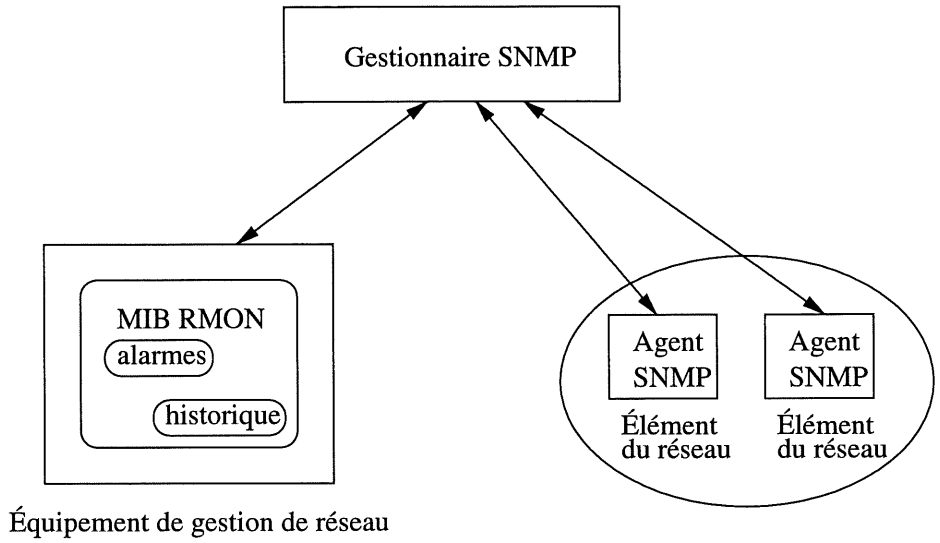

FIG. 8 – *MIB RMON et la gestion SNMP*

permet d'analyser les performances du réseau, de prévoir les problèmes et de déterminer les solutions adéquates.

1.6 Open Systems Interconnection (OSI)

La norme OSI traduit une approche système de la gestion des réseaux. Elle définit les éléments nécessaires à la spécification de la gestion des ressources de communication d'un système. Dans chaque système, on trouve les composants architecturaux chargés de la gestion des relations de communication entre les systèmes. Chaque système est structuré en sept couches selon le modèle de référence de l'OSI.

La gestion système consiste à définir le dialogue entre les couches adjacentes d'un même système, qui se dénomme service, et le dialogue entre les couches de même niveau de systèmes différents, qui se dénomme protocole. L'application de l'administration est localisée au niveau de la couche application. OSI a défini *Common Management Information Service* (CMIS) comme service et *Common Management Information Protocol*

(CMIP) comme protocole (figure 9).

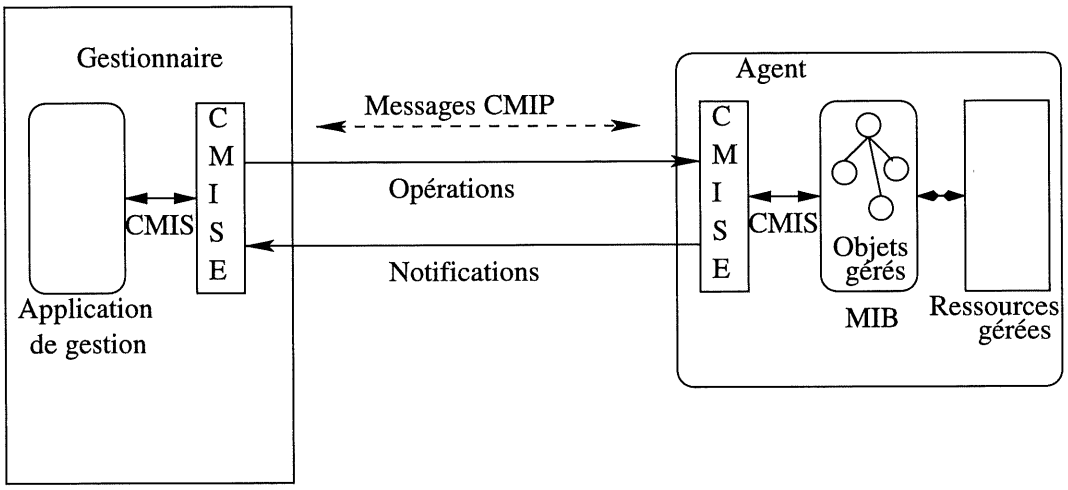

FIG. 9 – *Architecture du système de gestion OSI*

1.6.1 Modèle informationnel

Dans la norme OSI, les ressources à gérer sur un réseau sont modélisées sous forme d'objets de gestion appelés *Managed Objects* (MO). L'ensemble des informations gérées par un *agent* à travers les MO est appelé MIB.

Les entités gérées sont préalablement décrites dans un modèle d'information défini dans les langages *Guidelines for the Definition of Managed Objects* (GDMO) et *Abstract Syntax Notation 1* (ASN.1). Les types des objets dans le modèle d'information sont décrits dans une syntaxe abstraite qu'est ASN.1. Les classes décrivant les entités à gérer sont les *Managed Object Classes* (MOC) qui sont spécifiées en GDMO. Ce langage permet de décrire les MOC, leurs attributs, les opérations sur ces attributs, les actions et les notifications que ces entités peuvent générer.

1.6.2 Modèle de communication

Afin d'effectuer des opérations de gestion sur un MO, le *gestionnaire* envoie des requêtes CMIP à l'*agent* OSI. CMIP est le protocole utilisé pour échanger des données entre le *gestionnaire* et les *agents* OSI (figure 9). L'application de gestion utilise l'interface CMIS pour accéder via CMIP aux informations de gestion des *agents* OSI.

L'interface CMIS fournit les fonctions de la gestion système qui permettent le transfert des opérations et des notifications de gestion en utilisant CMIP pour la communication réseau. Par exemple, elle fournit les moyens d'envoyer une requête de lecture des valeurs des attributs d'un objet.

Chaque entité du réseau possède une interface appelée *Common Management Information Service Element* (CMISE) qui fournit les opérations de type création, modification et suppression d'objets. Ces opérations permettent la mise à jour et la lecture de valeurs d'attributs d'objets, l'exécution d'une action et l'établissement des rapports d'événements pour le service de notification.

L'entité CMISE distingue six services d'opération et un service de notification :

- M_Create : demande la création d'un objet dans la MIB de l'*agent*.
- M_Delete : supprime des objets dans la MIB de l'*agent*.
- M_Action : demande l'exécution d'une action sur un ou plusieurs objets.
- M_Set : modifie les valeurs d'attributs d'objets.
- M_Get : consulte les valeurs associées à des attributs d'objets.
- M_Cancel_Get : demande de ne pas recevoir les résultats d'un M_Get précédent.
- M_Event_Report : transmet un rapport d'événement relatif à un objet.

1.6.3 Modèle organisationnel

Ce modèle repose sur le concept *gestionnaire/agent.* Le *gestionnaire* spécifie des objets filtres dans les ressources gérées. Un objet filtre définit un intervalle de valeurs acceptables ou non pour la ressource. Lorsque la valeur seuil de ce filtre est dépassée, l'*agent* génère un événement à son *gestionnaire* qui va assurer le traitement et l'interprétation de cette notification. Ce mode de fonctionnement est appelé *Rapport* [1].

1.7 Objectifs communs des protocoles SNMP et CMIP/CMIS

Ces deux protocoles partagent plusieurs caractéristiques, comme la transmission des informations de gestion des réseaux, l'utilisation d'une base de données (MIB), l'utilisation des mêmes primitives de base (set, get), la signalisation d'événements par un message (trap) et l'utilisation du modèle *gestionnaire/agent.*

1.8 Comparaison des protocoles SNMP et CMIP/CMIS

Le tableau 1 ci-dessous donne un aperçu général des avantages et des inconvénients des protocoles SNMP et CMIP/CMIS.

TAB. 1 – *Comparaison des protocoles SNMP et CMIP/CMIS*

Protocoles	**Avantages**	**Désavantages**
SNMP -Utilise le mode non connecté. -Déclenchement de requêtes par *Sollicitation.*	-Implantation simple -Utilisation très large	-Absence de sécurité (réglée par SNMPv3) -Modèle d'information basé sur des objets décrits sous forme de variable simple -Existe uniquement sur les réseaux qui supportent IP.
CMIP/CMIS -Utilise le mode connecté. -Déclenchement de requêtes par *Rapport.*	-Sécurité développée -Modèle d'information basé sur l'approche orientée objet	-Non supporté par tous les équipements

Des différences majeures distinguent ces deux protocoles.

Le protocole CMIP permet d'effectuer certaines actions sur les objets en utilisant les primitives M_Action, M_Create et M_Delete. Par contre, SNMP n'offre pas la possibilité d'agir sur un objet en dehors de la lecture et de l'écriture des données dans les attributs de la MIB.

Dans CMIP, les requêtes et les réponses peuvent inclure des objets complexes, contrairement au SNMP où elles incluent seulement les valeurs d'attributs élémentaires.

CMIP utilise le mode connecté. Par contre, SNMP utilise le mode datagramme. Ce mode permet à SNMP de diminuer le temps de traitement de la transmission et de la réception des requêtes, mais il ne lui permet pas d'assurer à l'émetteur l'arrivée du message au destinataire.

La technique de *Rapport* utilisée par OSI permet de diminuer le volume de trafic sur le réseau. Cependant, si le nombre des messages de notification est très élevé, cette technique risque de paralyser le *gestionnaire.* Par contre, la technique de *Sollicitation*

utilisée par SNMP renforce la sécurité du système, mais elle génère beaucoup de trafic sur le réseau et limite le nombre de ressources que peut gérer un seul *gestionnaire*.

Ben Artzy et al. [1] ont étudié quantitativement les flux générés par une *Sollicitation* et un *Rapport* afin d'estimer le nombre de ressources d'un réseau local ou étendu gérables par un seul *gestionnaire* SNMP ou OSI. Ils ont utilisé une implantation de CMIS/CMIP sur TCP/IP (CMOT). Le résultat de l'étude a démontré qu'un *gestionnaire* SNMP peut gérer 4 500 ressources sur un réseau local et 750 ressources sur un réseau étendu, avec une période de *Sollicitation* d'un objet de 15 minutes. Un *gestionnaire* OSI peut gérer 18 000 ressources sur un réseau local ou étendu, avec une période de *Rapport* d'un objet de 15 minutes.

Cette étude met en évidence les désavantages de la gestion centralisée. Ce type de gestion impose une limite au nombre de ressources à gérer sur le réseau.

1.9 Discussion

Dans ce chapitre, nous avons abordé les aspects architecturaux des applications et des systèmes de gestion.

Sur le plan de la gestion, l'architecture de l'ISO a ouvert la voie à la structuration du système de gestion et à l'utilisation de la puissance de l'objet. Par contre, la simplicité et la rapidité de mise en œuvre de SNMP lui ont permis d'être adopté par l'industrie informatique, et d'être utilisé pour la gestion de la majorité des réseaux actuels. Cela, en dépit de ses faiblesses au niveau de la sécurité et au niveau de la gestion dynamique des objets de la MIB.

La plupart des réseaux existants sont hétérogènes et leur gestion exige la coexistence de différents systèmes de gestion. La coexistence de ces systèmes pose le problème de l'intégration entre les protocoles de gestion des réseaux. Cette intégration est devenue un

souci permanent des organismes de normalisation. Ces derniers n'ont pas réussi à imposer une approche unique pour cette intégration. Cette situation a poussé les chercheurs, ingénieurs et industriels du domaine à proposer des solutions spécifiques à leurs besoins.

Les travaux d'intégration de CMIP et de SNMP reposent principalement sur l'approche de passerelle CMIP-SNMP [37], sur l'approche d'intégration générique [27] ou sur le mécanisme d'intégration de plusieurs interfaces de gestion (SNMP et CMIS) sur un même *agent* [38]. Ces approches n'ont pas eu le succès escompté.

L'apparition des nouvelles technologies de la répartition a donné lieu à de nouvelles orientations pour l'intégration des systèmes de gestion. Ces nouvelles orientations consistent à utiliser une plate-forme distribuée pour assurer l'interopérabilité des protocoles existants CMIP et SNMP. Des travaux ont été réalisés dans le cadre de *Joint Inter Domain Management* (JIDM) sur l'intégration des systèmes de gestion SNMP et OSI en utilisant l'environnement distribué CORBA [40].

Chapitre 2

Les approches décentralisées de la gestion des réseaux

Les approches décentralisées consistent à répartir les tâches de gestion entre les différentes entités qui interviennent dans la gestion d'un réseau. Ces entités coopèrent entre elles pour accomplir ces tâches. Chaque entité est autonome et peut assurer un traitement particulier.

Le passage vers l'approche décentralisée permet d'assurer l'autonomie, l'évolution et l'ouverture des systèmes de gestion des réseaux. Pour cela, plusieurs architectures ont été proposées. Parmi les plus importantes, nous présentons les architectures *Distributed Management Environment* (DME), *Hierarchical Network Management* (HNM) et *Management by delegation* (Mbd).

2.1 Approches décentralisées

2.1.1 Distributed Management Environment (DME)

L'architecture DME est proposée par l'*Open Software Foundation* (OSF) [33]. Elle définit une norme pour le développement d'applications de gestion qui concernent aussi bien la gestion des logiciels, des systèmes et des services, que la gestion des unités de traitement réparties.

L'architecture DME offre un certain nombre de composants de base intégrés et réutilisables. Ces composants constituent un environnement de développement d'applications de gestion. DME est composée de deux parties essentielles : le *Management Request Broker* (MRB) et les objets serveurs (figure 10a). Le MRB est la pièce centrale de l'architecture DME. Il implante une API de base qui accède aux services fournis par DME. Le MRB fournit une interface de programmation normalisée pour utiliser les protocoles SNMP et CMIP à travers une API. Il fournit également des mécanismes d'appel des méthodes associées à un objet DME. Ce dernier est utilisé pour la modélisation des ressources à gérer et pour la modélisation des applications de gestion.

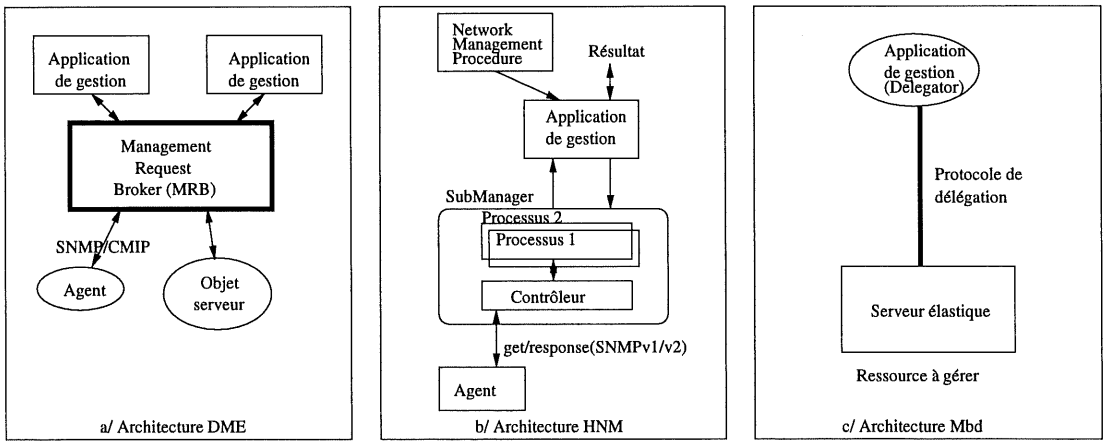

FIG. 10 – *Architectures décentralisées*

L'objet serveur de DME offre des opérations qui permettent l'accès aux données enregistrées dans cet objet, de même que la création et la suppression des instances de cet objet. À chaque fois que le MRB reçoit un message, il le transmet à l'objet serveur

qui est associé à l'objet spécifié. La localisation des objets DME sur le réseau se fait par l'intermédiaire du service *Répertoire* de *Distributed Computing Environment* (DCE) [10].

2.1.2 Hierarchical Network Management (HNM)

L'architecture HNM est proposée par l'Université de Vienne [39]. Elle utilise le concept de *SubManager* semblable au protocole de *gestionnaire de gestionnaires* décrit dans le SNMPv2 [6]. Un *SubManager* est associé à un groupe *d'agents*. Il rassemble l'information élémentaire de ces *agents*, exécute quelques traitements et produit des valeurs plus significatives qui peuvent être employées par un *gestionnaire* supérieur (figure 10b). Cette méthode réduit de manière significative la quantité de trafic de gestion, car c'est uniquement l'information traitée qui est envoyée au *gestionnaire* principal.

À chaque fois qu'on veut intégrer des fonctionnalités supplémentaires au système, des procédures *Network Management Procedure* (NMP) peuvent être chargées d'une manière dynamique sur le *SubManager*. Ces procédures sont enregistrées dans deux tables (*subMgrEntry* et *subMgrOps*) de *SubManager*. Chaque procédure est lancée périodiquement et un processus est créé. Le processus exécute la procédure et enregistre le résultat dans la table *subMgrValue* qui est lue par le *gestionnaire*.

2.1.3 Management by delegation (Mbd)

L'architecture Mbd est proposée par l'Université de Californie [47]. Elle définit un modèle plus flexible qui utilise le concept du serveur élastique dont la fonctionnalité peut être étendue au moment de l'exécution. Cela se fait en déléguant de nouvelles procédures fonctionnelles au serveur.

Dans l'approche Mbd, au lieu d'apporter les données de la ressource gérée vers l'application de gestion, les applications sont déléguées aux ressources gérées et exécutées dans le même environnement que les données. Le *gestionnaire* (*delegator entity*) emploie

un protocole de délégation pour télécharger et contrôler un programme délégué (DP) (figure 10c). Ce dernier peut être délégué à une ressource gérée au démarrage, de telle sorte que la ressource assume les responsabilités d'autonomie. L'architecture Mbd n'exige pas un modèle sémantique uniforme des données de la ressource gérée, ce qui résout le problème d'hétérogénéité entre les différentes ressources du réseau et élimine les limites du développement d'applications de gestion.

L'architecture Mbd peut interopérer avec les protocoles existants SNMP et CMIP. Elle étend les capacités de ces protocoles en ajoutant des fonctionnalités supplémentaires aux ressources gérées ; par exemple, une ressource qui supporte le protocole SNMP peut évoluer pour supporter des accès de CMIP.

L'utilisation du Mbd permet à des ressources du réseau d'acquérir des capacités de gestion d'autonomie locale. Cela leur permet d'assurer une gestion entièrement autonome en cas de perte de communication avec le *gestionnaire*.

2.1.4 Résumé des architectures DME, HNM et Mbd

Les architectures DME, HNM et Mbd proposent de nouvelles approches des systèmes de gestion des réseaux. Elles assurent une indépendance vis-à-vis des protocoles de gestion des réseaux, assurent l'expansibilité des fonctionnalités du système de gestion et délèguent des tâches de gestion aux ressources gérées. La complexité de ces architectures ne leur a pas permis d'être adoptées par l'industrie informatique, ce qui a limité leur développement.

Ces architectures n'offrent pas la possibilité aux applications de gestion de détecter de nouvelles fonctions sur le réseau, de les intégrer et de les utiliser dynamiquement. Afin de répondre à cet objectif, nous proposons une nouvelle architecture pour les applications de gestion. Cette architecture est constituée d'un ensemble de composants distribués qui communiquent entre eux à travers l'Intergiciel CORBA.

2.2 Concept des composants distribués

Un composant est un module logiciel autonome. Il peut être construit et programmé indépendamment des autres composants. Il est caractérisé par ses attributs et ses méthodes. Il réduit l'accès à ses services et à sa structure interne en proposant une ou plusieurs interfaces publiques. Il décrit un traitement spécifique d'un système [9].

Un composant logiciel encapsule et masque de l'information, ce qui nous permet d'établir une analogie avec les composants matériels. De plus, la notion d'héritage permet de développer une spécialisation de ces composants. Ces derniers fournissent une infrastructure utilisable et réutilisable pour le développement rapide de familles d'applications.

Dans le domaine de la gestion des réseaux, les composants sont utilisés pour la modélisation des ressources à gérer et des fonctions de base de la gestion.

Une application distribuée est constituée d'un ensemble de composants logiciels répartis sur plusieurs ordinateurs du réseau. Un même composant peut être utilisé par plusieurs applications et inséré au fur et à mesure de l'évolution des besoins dans le système sans remettre en cause l'architecture initiale. L'infrastructure logicielle qui supporte ces composants est appelée Intergiciel.

2.3 Intergiciel

L'Intergiciel assure le lien entre les composants qui résident sur les différentes plates-formes. Il constitue un environnement distribué pour ces composants. Il permet aux composants de coopérer entre eux avec une transparence totale de leur localisation. Il masque l'hétérogénéité des plates-formes malgré les différences de protocoles réseaux, de systèmes d'exploitation et de langages de programmation.

Dans l'Intergiciel, la répartition des ressources est transparente aux utilisateurs ; par

conséquent, ces derniers ne distinguent pas les composants locaux des composants distants.

Une application répartie est constituée d'un ensemble de composants qui communiquent entre eux par l'échange de requêtes. Cette communication est assurée par l'Intergiciel, qui offre des mécanismes de communication et de transport pour la localisation et l'activation des composants distribués. L'Intergiciel est le vecteur d'interaction des composants distribués.

Plusieurs technologies sont en compétition. La plus propriétaire est le modèle COM/DCOM de Microsoft qui permet de faire interagir des composants ActiveX [18]. La plus ouverte se nomme CORBA/IIOP, de l'*Object Management Group* (OMG), qui offre une infrastructure de communication entre les objets distribués hétérogènes [19]. Enfin, le *Remote Method Invocation* (RMI) de JavaSoft offre un autre modèle de communication entre les objets distribués [14]. JavaSoft a défini également les composants Java Beans [5]. Ces composants sont équivalents aux composants ActiveX de Microsoft. Ils définissent des composants logiciels réutilisables écrits en Java ; ils permettent de construire une application par assemblage de composants logiciels à l'aide d'outils visuels interactifs comme *Visual Cafe* de Symantec et *Java Workshop* de SunSoft.

Le modèle RMI contient une série de classes et d'interfaces permettant au développeur d'appeler des objets qui existent déjà dans une application qui s'exécute sur une autre machine virtuelle Java. Ces objets sont décrits par des interfaces qui définissent leurs attributs et leurs méthodes. Java RMI est un mécanisme qui utilise uniquement le langage Java.

Le modèle *Distributed Component Object Model* (DCOM) est une extension du modèle de composant COM qui permet de développer des applications par assemblage de composants répartis [9]. Il offre la transparence vis-à-vis des langages de programmation (Java, C, C++, etc.) et sépare l'interface de l'implantation des objets. Il fournit une API pour découvrir, charger et appeler dynamiquement les interfaces des objets. *Component Object*

Model (COM) est une architecture de base qui fournit des services de plus haut niveau. Ces services sont utilisés par les composants OLE et ActiveX. Le modèle COM dispose d'un mécanisme pour la communication entre objets. Ces objets peuvent être situés dans des applications ou des bibliothèques dynamiques différentes, mais fonctionnent sur la même machine. Par contre, DCOM utilise le standard DCE (*Distributed Computing Environment*) pour mettre en relation deux objets COM sur des machines distantes [10].

Common Object Request Broker Architecture (CORBA) est une spécification normalisée du développement logiciel orienté objet en environnement distribué [24]. Elle définit un bus logiciel d'objets distribués. Dans la suite de ce chapitre, nous détaillons davantage l'architecture CORBA que nous utilisons dans notre solution.

Les architectures proposées par Microsoft et Sun représentent en réalité des solutions propriétaires. Par contre, la solution CORBA de l'OMG est considérée comme la norme la plus stable par les développeurs et concepteurs d'applications distribuées. D'ailleurs, des solutions d'interopérabilité entre les systèmes d'objets DCOM, RMI et CORBA ont été proposées pour rendre la communication transparente entre les objets de ces systèmes [5]. L'OMG et Microsoft ont défini des spécifications en vue d'établir des passerelles DCOM/CORBA et CORBA/DCOM. Par contre, Sun a intégré le support de IIOP dans RMI afin de lui permettre de communiquer avec les objets CORBA.

2.4 Common Object Request Broker Architecture (CORBA)

Les spécifications de CORBA sont définies par l'OMG (*Object Management Group*) qui est une organisation internationale regroupant plus de 700 acteurs de l'industrie de l'informatique [19]. CORBA assure la transparence des communications entre objets distribués. Elle supporte des langages et des plates-formes hétérogènes.

2.4.1 Architecture globale de CORBA

Cette architecture vise à prendre en compte toutes les fonctionnalités nécessaires à la construction d'applications dans un environnement ouvert et distribué. Elle structure ces fonctionnalités en cinq grandes catégories (figure 11) :

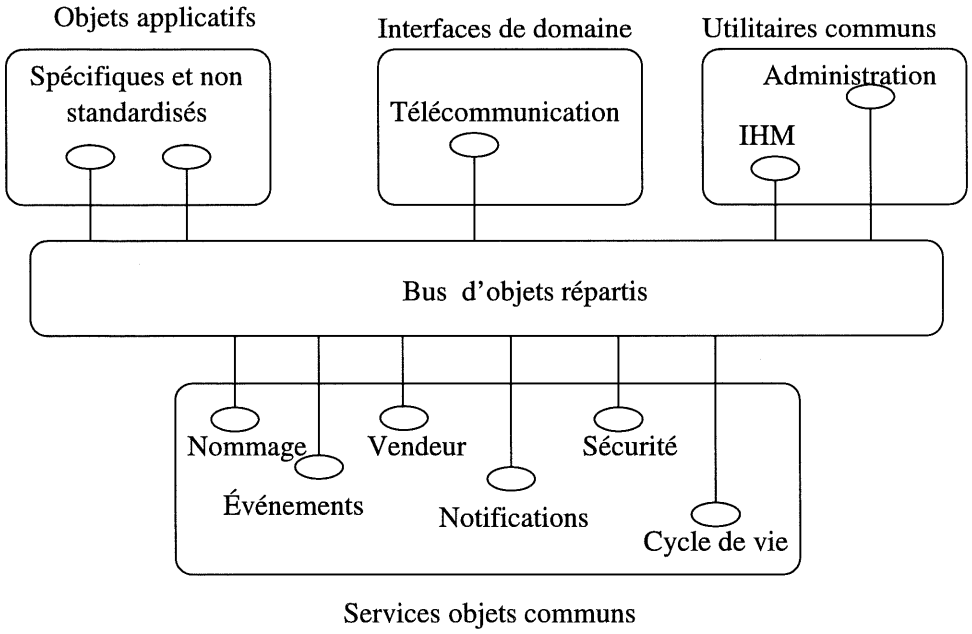

FIG. 11 – *Architecture globale de CORBA*

Le bus d'objets répartis (*Object Request Broker*)

C'est le bus d'interconnexion entre objets. Il encapsule tous les mécanismes nécessaires pour communiquer avec des objets serveurs, les activer et les enregistrer. Il assure la liaison avec les langages de programmation et la transparence des appels, ainsi que la communication entre les différents bus ORB.

Les services objets communs (*Common Object Services*)

Ces services fournissent des abstractions aux fonctions système de base. L'utilisation de ces services permet d'écrire des applications indépendantes des mécanismes sous-jacents du système. Cette indépendance permet la portabilité des applications utilisant ces services. Généralement ces services sont développés par les concepteurs d'ORB. Parmi les services définis par l'OMG (figure 11), nous décrivons les rôles des services Nommage, Vendeur, Événements et Notifications.

Le service Nommage (*Naming Service*) permet la recherche des objets en fonction de noms symboliques leur ayant été associés. C'est l'équivalent des pages blanches de l'annuaire téléphonique.

Le service Vendeur (*Trader Service)* permet la recherche des objets en fonction de leurs caractéristiques. C'est l'équivalent des pages jaunes de l'annuaire téléphonique.

Le service Nommage est très couramment utilisé dans les applications CORBA et son utilisation est beaucoup plus simple que celle du service Vendeur. Cependant, ce dernier doit être impérativement mis en œuvre lorsque les applications CORBA manipulent des critères de recherche complexes.

Le service Événements (*Event Service*) permet aux objets de produire des événements asynchrones à destination d'objets consommateurs via des canaux d'événements. Les canaux fournissent deux modes de fonctionnement. Dans le mode *push*, le producteur a l'initiative de la production et le consommateur est informé des événements. Dans le mode *pull*, le consommateur demande explicitement un événement, et le producteur est alors sollicité.

Le service Notifications (*Notification Service*) est une extension du service précédent. Les consommateurs sont informés uniquement des événements qui les intéressent. Pour cela, ils posent des filtres sur le canal, réduisant ainsi le trafic réseau engendré par la propagation des événements.

Les utilitaires communs (*CORBA Facilities*)

Ces utilitaires sont définis comme une collection de services et d'objets spécifiques à un domaine particulier. Ils représentent un ensemble de classes réutilisables. Ils peuvent être appelés par les autres objets CORBA via leurs interfaces. Ces utilitaires communs couvrent des services de plus haut niveau, comme l'interface utilisateur et l'administration des systèmes et des réseaux.

Les interfaces de domaine (*Domain Interfaces*)

Ces interfaces définissent des objets de métiers spécifiques à des secteurs d'activités, comme les télécommunications. Leur objectif est d'assurer l'interopérabilité sémantique des systèmes d'information d'entreprises d'un même métier.

Les objets applicatifs (*Application Objects*)

Ces objets ne sont pas normalisés. Ils définissent des objets spécifiques à une application répartie. Toutefois, dès que le rôle de ces objets apparaît dans plus d'une application, ils peuvent être intégrés dans les catégories utilitaires communs ou interfaces de domaine, et être normalisés par l'OMG.

2.4.2 Appel statique et dynamique

CORBA offre deux mécanismes d'appel des opérations sur les objets distants : l'appel statique et l'appel dynamique.

Appel statique

Un objet client associé à la souche IDL d'un objet serveur peut appeler des opérations de l'objet serveur d'une manière transparente. La souche et l'ossature sont générées par

le compilateur IDL à partir de l'interface IDL de l'objet serveur. Les programmes d'implantation du client et du serveur sont compilés avec le résultat de la compilation des interfaces IDL de l'objet serveur. Dans ce cas, les appels sont contrôlés à la compilation.

Nous illustrons à la figure 12 l'appel d'une opération sur un objet serveur distant. Une application client CORBA appelle une opération d'un objet distant en spécifiant la référence de l'objet, le nom de l'opération et les arguments nécessaires. À la réception de cette requête, l'objet serveur exécute cette opération et renvoie le résultat au client CORBA. Ce résultat peut être un argument de sortie de l'opération ou une exception causée par une erreur d'exécution de l'opération.

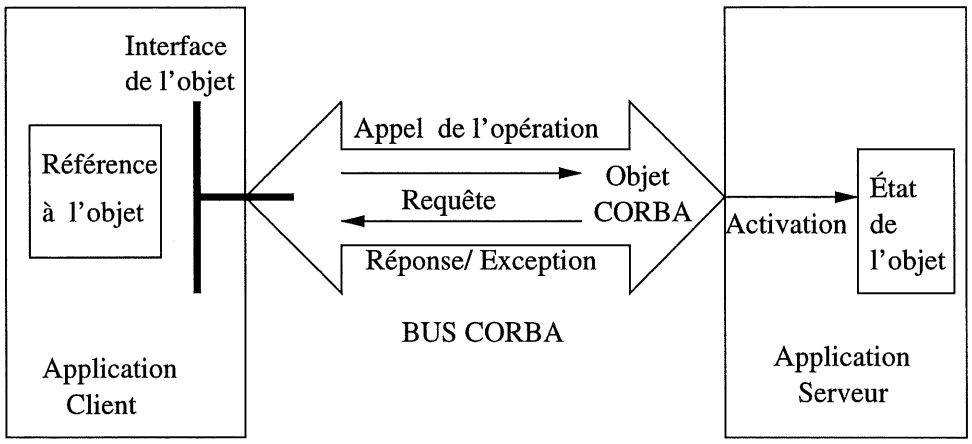

FIG. 12 – *Appel d'une opération d'un objet distant via CORBA*

Appel dynamique

Au cours de l'exécution d'une opération qu'il souhaite appeler, un objet client peut découvrir la définition de celle-ci, lui envoyer un appel paramétré et recevoir une réponse. Dans ce cas, les appels sont contrôlés à l'exécution.

CORBA offre un ensemble de mécanismes pour exploiter et implanter dynamiquement des objets répartis : le référentiel des interfaces (IR), l'interface d'appels dynamiques (DII) et l'interface d'ossatures dynamiques (DSI). Ces mécanismes dynamiques permettent

de construire des applications qui s'adaptent automatiquement aux changements des spécifications IDL.

Afin d'appeler dynamiquement une opération sur un objet distant, le programme client doit respecter les étapes suivantes :

- Trouver la référence de l'objet en utilisant le service de Nommage.
- Obtenir l'interface de l'objet en utilisant l'interface d'appels dynamiques (DII).
- Obtenir la description de l'opération à appeler en utilisant le référentiel des interfaces (IR).
- Construire la liste des arguments à transmettre.
- Créer la requête.
- Traiter les résultats retournés.

2.4.3 Langage de description d'objets

Dans l'architecture CORBA, les services sont décrits dans un langage de spécification appelé *Interface Definition Language* (IDL). IDL est indépendant des langages de programmation que nous utilisons pour implanter les comportements des objets.

Le langage IDL permet de définir les interfaces de chaque objet. Ces interfaces sont ensuite projetées vers les langages de programmation évolués (C++, Java, etc.) en utilisant un compilateur (Idltojava, Idltoc, etc.) spécifique à chaque langage. Ce compilateur génère automatiquement les fichiers qui contiennent la souche et l'ossature dans le langage de programmation utilisé pour l'implantation des objets (figure 13).

Nous illustrons à la figure 13 le concept de projection d'une spécification IDL vers les langages de programmation évolués. À partir d'un fichier IDL qui décrit les interfaces et les attributs d'un objet, nous générons les fichiers qui contiennent la souche et l'ossature en utilisant des compilateurs spécifiques à chaque langage. Ces fichiers sont utilisés par le client et le serveur CORBA pour générer des applications exécutables. Ces

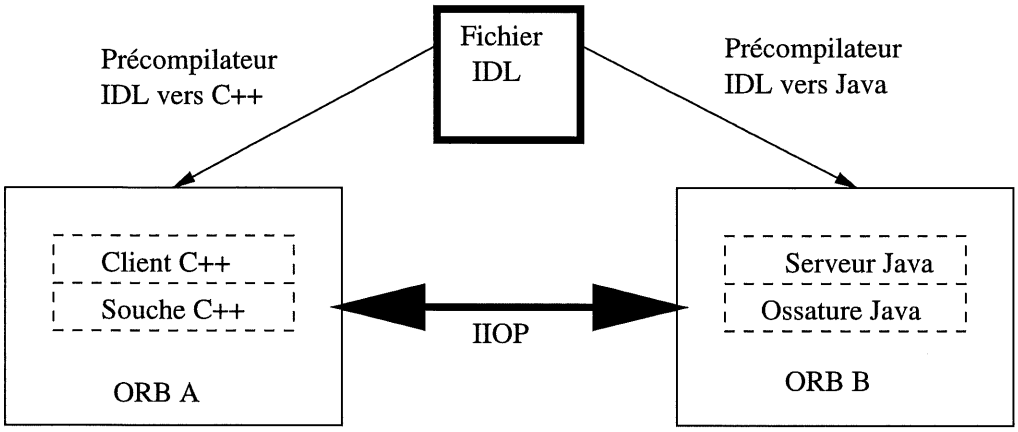

FIG. 13 – *Communication entre des applications hétérogènes*

applications sont écrites dans des langages différents et utilisent des ORB différents. Elles communiquent entre elles grâce au protocole *Internet Inter-ORB Protocol* (IIOP).

2.5 CORBA et la gestion des réseaux

Dans l'architecture CORBA, la modélisation des différentes entités d'un système de gestion des réseaux peut être mise en œuvre sous forme d'objets CORBA [34]. Ces entités représentent les ressources à gérer sur le réseau et les composants de l'application de gestion. Ces ressources sont représentées par des objets gérés (figure 14). Un objet géré possède une identité unique et un ensemble d'attributs et d'opérations (actions et notifications). Chaque objet géré peut être développé indépendamment des autres. Ceci permet une plus grande flexibilité de configuration, l'unité de développement n'étant plus forcément un *agent* entier. Par exemple, un objet géré peut représenter un groupe particulier de la MIB. La colocalisation des objets gérés peut donc être modélisée en fonction du niveau d'interaction requis entre les différents objets.

L'application de gestion est représentée par un ou plusieurs objets. Elle est capable d'émettre et de recevoir des requêtes en appelant les opérations des objets gérés via

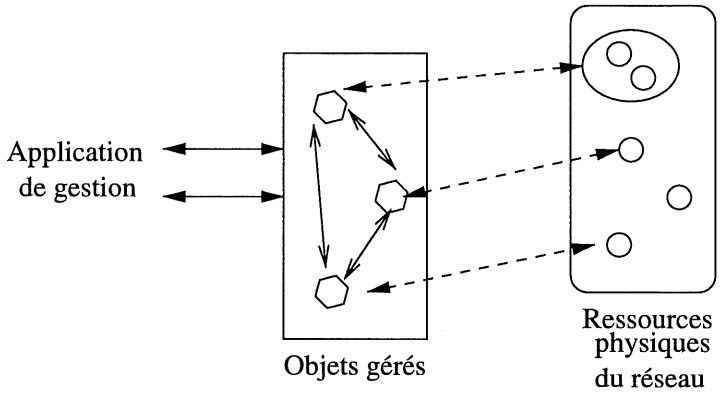

FIG. 14 – *Objets gérés*

CORBA (figure 15). Ce mécanisme d'appel rend transparente l'utilisation des protocoles de gestion des réseaux.

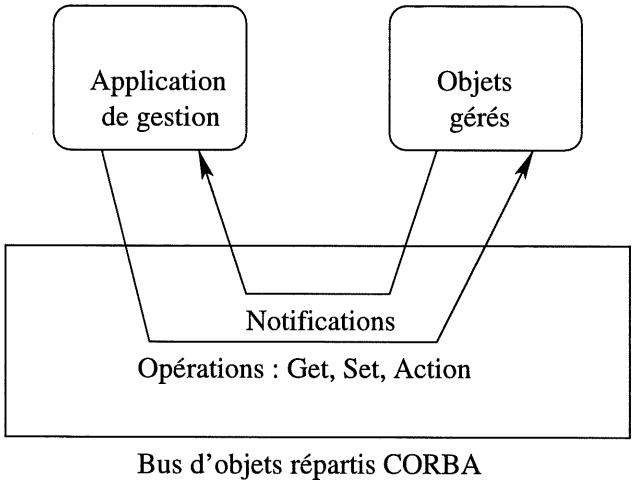

FIG. 15 – *Interaction entre le gestionnaire et les objets gérés*

La localisation des objets gérés se fait par l'intermédiaire du service de Nom de CORBA. À travers ce service, nous enregistrons seulement la racine de l'arbre qui constitue la MIB et chaque objet gère les références de ses subordonnées.

Les objets gérés peuvent émettre des notifications à l'application de gestion en utilisant le service Événement de CORBA.

2.5.1 X/Open's Joint Inter-Domain Management task force (XoJIDM)

Le *Network Management Forum* (NMF) et *X/Open* ont créé le groupe *Joint Inter-Domain Management* (XoJIDM) [20] pour étudier et fournir les outils qui permettent le dialogue entre les systèmes de gestion basés sur CMIP, SNMP et CORBA. XoJIDM a défini des algorithmes de traduction des spécifications de modèles d'information de GDMO/ASN.1 vers CORBA-IDL et de SNMP MIB vers CORBA-IDL. Il a défini également des algorithmes de traduction des interactions entre CMIP, SNMP et CORBA. La traduction des interactions consiste à traduire les requêtes et les réponses de CMIP vers CORBA, de SNMP vers CORBA, de CORBA vers CMIP et de CORBA vers SNMP.

L'interopérabilité des domaines CORBA et CMIP ou CORBA et SNMP peut se faire de deux façons différentes. La première consiste à utiliser une passerelle, qui constitue un point d'interconnexion entre les deux domaines, et à laisser les composants existants inchangés. La deuxième consiste à intégrer le support de l'un des domaines dans l'autre. La première solution est plus simple à implanter à court terme, mais elle a un impact négatif au niveau de la performance des interactions.

Deux approches se distinguent pour le développement de la passerelle : l'approche statique et l'approche dynamique.

Approche Statique

Cette approche consiste à traduire le modèle d'information (MIB) d'un *agent* (OSI ou SNMP) en une spécification CORBA-IDL [46]. Les modèles d'information sont décrits en langages de définition ASN.1 pour SNMP et GDMO/ASN.1 pour OSI.

Le principe de la traduction des objets CMIP de GDMO/ASN.1 vers IDL repose

essentiellement sur les points suivants [4] :

- Chaque attribut de l'objet géré peut être spécifié par un *attribute* dans la spécification de l'interface.
- Chaque opération qui peut être appelée sur l'objet géré doit être déclarée comme opération dans son interface IDL.
- Les opérations de création et de suppression d'un objet géré sont fournies par le service CORBA *Cycle de vie.*
- Les notifications sont spécifiées dans des interfaces séparées.

Le principe de la traduction de la MIB SNMP de ASN.1 vers IDL est défini par les points suivants [28] :

- Traduction de chaque *table entry* en interface IDL. Les éléments de cette table représentent les attributs de cette interface.
- Traduction de chaque variable d'un groupe en interface IDL. Chaque variable est représentée par un attribut dans cette interface.
- Traduction des notifications en opérations dans des interfaces séparées.

Dans l'approche statique [35], l'application de gestion (client CORBA) doit être compilée avec le résultat de la compilation de l'interface IDL de la MIB concernée. Cette interface est générée par le traducteur des spécifications GDMO/ASN.1 vers CORBA-IDL. L'implantation de ces interfaces IDL assure la traduction des requêtes et des notifications de CORBA en requêtes CMIP ou SNMP [8]. Dans cette approche, l'interface IDL de la MIB est connue par le client CORBA au moment de la compilation. Tout changement de cette interface implique une recompilation de l'application de gestion avec la nouvelle interface générée par le traducteur GDMO/ASN.1 - IDL.

Approche Dynamique

Afin de surmonter les limites de l'approche statique, plusieurs travaux de recherche ont proposé diverses solutions pour assurer une intégration dynamique entre CORBA, CMIP et SNMP. Parmi ces solutions, nous citons *Generic Object Model* (GOM) [2] et *CORBA-Liaison* [7], qui proposent des solutions dynamiques d'interprétation et de traduction des requêtes au moment de l'exécution. Cela se fait sans avoir besoin de compiler la spécification GDMO/ASN.1 de la MIB avec l'application de gestion (client CORBA), contrairement à l'approche statique. Ces solutions utilisent les mécanismes DII et DSI de CORBA pour construire des passerelles qui interprètent et traduisent les requêtes d'une manière dynamique.

Dans cette approche, le client ne possède pas une connaissance préalable sur les interfaces des objets disponibles dans le système. Il est indépendant de la modification des interfaces. En cas de changement de ces interfaces, le client continue à fonctionner, contrairement à l'approche statique où le client doit être recompilé à chaque changement de l'interface.

Les solutions GOM et *CORBA-Liaison* s'appliquent pour les architectures à base d'un *gestionnaire* de réseau représenté par un client CORBA utilisant des *agents* SNMP ou OSI [3].

2.5.2 Les cas d'utilisation de CORBA dans la gestion de réseau

Sur le plan pratique, nous distinguons quatre cas d'utilisation de CORBA dans la gestion des réseaux :

- Gestion des *agents* OSI en utilisant une application de gestion CORBA.
- Gestion des *agents* SNMP en utilisant une application de gestion CORBA.
- Gestion des *agents* CORBA en utilisant une application de gestion OSI.
- Gestion des *agents* CORBA en utilisant une application de gestion SNMP.

Les deux premiers cas consistent à utiliser des *agents* OSI ou SNMP à partir d'une application de gestion représentée par un client CORBA. Dans ces cas, nous utilisons les algorithmes de traduction des spécifications GDMO/ASN.1 vers CORBA-IDL de XoJIDM [20]. Ces traductions permettent de réaliser des passerelles entre CORBA et les *agents* OSI ou SNMP (figure 16).

Le troisième cas consiste à utiliser une application de gestion OSI pour gérer des *agents* CORBA. Dans ce cas, XoJIDM a défini un *agent mandataire* qui reçoit les requêtes CMIP et les transmet aux objets concernés en utilisant l'appel dynamique de CORBA [45]. Ces objets sont enregistrés dans une MIB. *L'agent mandataire* utilise les traductions des spécifications de OSI en CORBA-IDL pour assurer sa mission.

Enfin, le quatrième cas consiste à utiliser une application de gestion SNMP pour gérer des *agents* CORBA. Mazumdar [31] a défini des *agents* basés sur CORBA se comportant comme des *agents* SNMP. Ils assurent la conversion des requêtes SNMP en requêtes CORBA, et la conversion des événements CORBA en alarmes SNMP. Cela, en utilisant la traduction de XoJIDM des spécifications de SNMP en CORBA-IDL.

Dans notre architecture, nous utilisons le deuxième cas qui consiste à gérer des *agents* SNMP à partir d'une application de gestion CORBA en utilisant une passerelle CORBA-SNMP.

2.5.3 CORBA et les architectures des applications de gestion des réseaux

Actuellement, plusieurs groupes de recherche à l'échelle internationale développent de nouvelles architectures pour les services de télécommunication et la gestion des réseaux. Ces architectures utilisent CORBA comme plate-forme pour le développement et l'interopérabilité des logiciels de contrôle et de gestion des réseaux. L'OMG [29] a développé le concept de *facilities* pour le domaine de l'administration des systèmes et des réseaux.

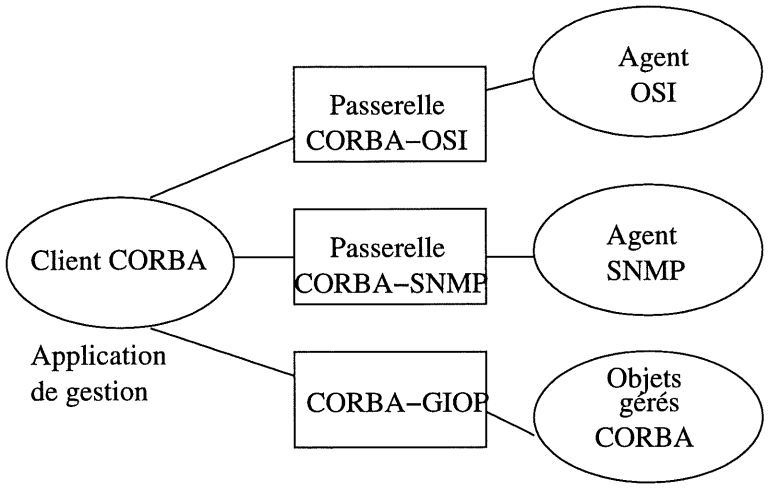

FIG. 16 – *CORBA et la gestion des réseaux*

Le consortium TINA-C a développé l'architecture *Telecommunication Information Networking Architecture* (TINA) [25].

Facilities de l'OMG

L'OMG a défini des interfaces et des services pour administrer, tester, configurer et réparer des objets répartis. Ces objets peuvent représenter tous les types de ressources telles que les équipements matériels, les applications et les objets individuels. Pour qu'un objet soit administrable, il doit implanter les interfaces IDL des fonctions de gestion suivantes :

- L'instrumentation
- La collecte de données
- La sécurité
- Le suivi d'instance
- L'ordonnancement
- La qualité du service
- La gestion des événements

Ces fonctions utilisent des services CORBA tels que Nom, Événement, Sécurité pour accomplir leurs différents traitements.

TINA

TINA [25] est le résultat de la convergence des applications informatiques et du monde de la télécommunication. C'est une architecture de service qui définit les concepts, les principes et les règles pour construire, développer et exploiter des services de télécommunication.

Elle définit un service de télécommunication comme un ensemble d'objets interagissant dans un environnement de traitement réparti. Les objets communiquent entre eux sans savoir où ils se trouvent, car l'interface entre les applications et le réseau est assurée par la plate-forme de traitement distribué CORBA.

Les principes fondamentaux de l'architecture TINA sont les suivants :

- Le modèle des composants de service universel identifie et sépare les aspects relatifs à l'accès au service, à sa gestion, à la logique, et à l'utilisation des ressources ou autres services.
- Le concept de session représente un ensemble d'objets, avec leurs relations et leurs interactions, qui permettent de réaliser le service pour l'utilisateur.

Plusieurs travaux de recherche proposent des scénarios pour le passage des architectures existantes de gestion des réseaux vers TINA [35].

2.6 Discussion

L'approche centralisée consiste à gérer toutes les ressources d'un réseau à partir d'un point central. Par contre, l'approche décentralisée répartit les tâches de gestion sur plusieurs éléments d'un réseau.

L'utilisation de CORBA dans l'approche décentralisée a permis de faire évoluer les systèmes de gestion propriétaires vers des systèmes de gestion intégrant SNMP, CMIP et CORBA. Cette dernière ajoute plus de flexibilité et d'ouverture aux applications de gestion.

CORBA apporte des solutions aux limites des protocoles SNMP et CMIP, que nous avons soulevées dans le Chapitre 1. Elle permet à une station de gestion de gérer un nombre plus important de ressources sur le réseau. La communication entre les différentes entités du réseau se fait par appel de méthodes des objets qui représentent ces entités. Chaque objet est défini par une interface IDL qui supporte l'héritage multiple. Enfin, l'interaction entre les ressources gérées et les applications de gestion se fait par notification d'événements en utilisant le service Événement de CORBA [13].

Les solutions de l'interopérabilité de CORBA, SNMP et CMIP définies par XoJIDM permettent une transparence d'accès entre les *gestionnaires* et les *agents* de ces différentes architectures. Cela permet au programmeur de développer des applications de gestion et des *agents* OSI ou SNMP sans avoir une connaissance approfondie de GDMO, ASN.1, SNMP et CMIP.

L'utilisation du concept de composants a permis d'améliorer la productivité des développeurs et d'assurer la réutilisation de code existant. Ce concept permet une programmation par assemblage de composants plutôt que par le développement de toutes les entités d'une application, ce qui réduit le coût et le temps de développement.

CORBA offre une infrastructure adéquate qui supporte les applications à base de composants distribués.

Malgré tous les avantages de CORBA dans la gestion des réseaux, certaines faiblesses sont à signaler. CORBA utilise le protocole de communication orienté connexion ; par contre, SNMP utilise le protocole de communication orienté datagramme. L'échange de messages via CORBA entre les différentes entités impliquées dans la gestion risque de générer un trafic d'information de contrôle et de gestion très important, au détriment

des informations utiles sur le réseau. La représentation des informations de gestion sous forme d'objets CORBA peut générer un nombre très élevé d'objets sur le réseau [28]. Ces objets exigent plus de ressources mémoires, ce qui peut engendrer une paralysie du réseau.

Après avoir présenté l'apport de CORBA et des composants logiciels dans le développement d'applications de gestion, nous décrivons dans le Chapitre 3 la conception et la réalisation de l'architecture de notre application de gestion des réseaux.

Chapitre 3

Réalisation d'une application de gestion des réseaux

Dans ce chapitre, nous présentons notre application qui est basée sur l'approche distribuée de la gestion des réseaux. Elle est constituée d'un ensemble de composants répartis. Ces derniers communiquent entre eux via l'Intergiciel CORBA. Nous utilisons le protocole SNMP pour gérer les ressources du réseau.

Cette application est constituée d'un *gestionnaire* de réseau qui communique avec les *agents* SNMP à travers une passerelle CORBA-SNMP. Cette passerelle assume le rôle de traducteur des requêtes CORBA vers SNMP et de SNMP vers CORBA. Le *gestionnaire* représente une interface graphique qui offre à l'administrateur un choix parmi les fonctions de gestion disponibles sur le réseau. Ces fonctions sont implantées par des composants CORBA (figure 17). Elles assurent les traitements de base de la gestion des réseaux, tels que la gestion de la configuration, la gestion de la performance et la gestion des fautes .

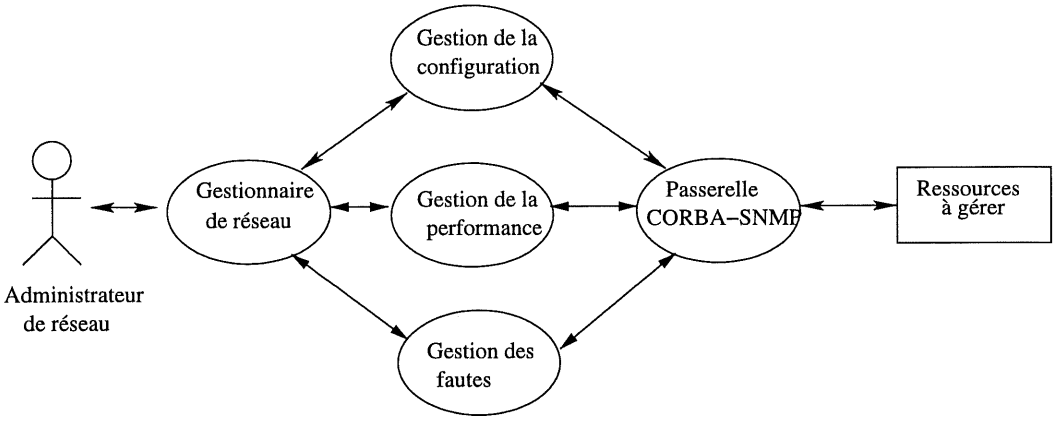

FIG. 17 – *Structure générale de l'architecture*

3.1 Conception

3.1.1 Description générale de l'application

Nous présentons à la figure 18 le schéma synoptique de notre application. Celle-ci est constituée d'un *gestionnaire*, d'un *mandataire*, d'une passerelle CORBA-SNMP, des fonctions de gestion et d'*agents* CORBA.

Le *gestionnaire* a une structure très simple, car il n'intègre pas les fonctions de gestion. Ces fonctions sont représentées par des composants CORBA indépendants.

Le *gestionnaire* offre des accès directs aux ressources du réseau via les opérations de base SNMP. Ces opérations sont définies dans le composant *mandataire*. Ce *gestionnaire* améliore ses fonctionnalités en utilisant les fonctions de gestion disponibles sur le réseau. Ces fonctions sont connectées sur le réseau sous forme de composants CORBA. Elles sont détectées automatiquement et utilisées par le *gestionnaire*. Cela permet à l'administrateur d'utiliser au besoin les fonctions nécessaires à un traitement particulier.

Le composant *mandataire* assure le lien entre les composants de l'application et la passerelle CORBA-SNMP. Il offre les opérations de base SNMP telles que *CorbaGet* et *CorbaSet,* qui permettent l'accès aux informations de gestion des *agents* SNMP. Il valide

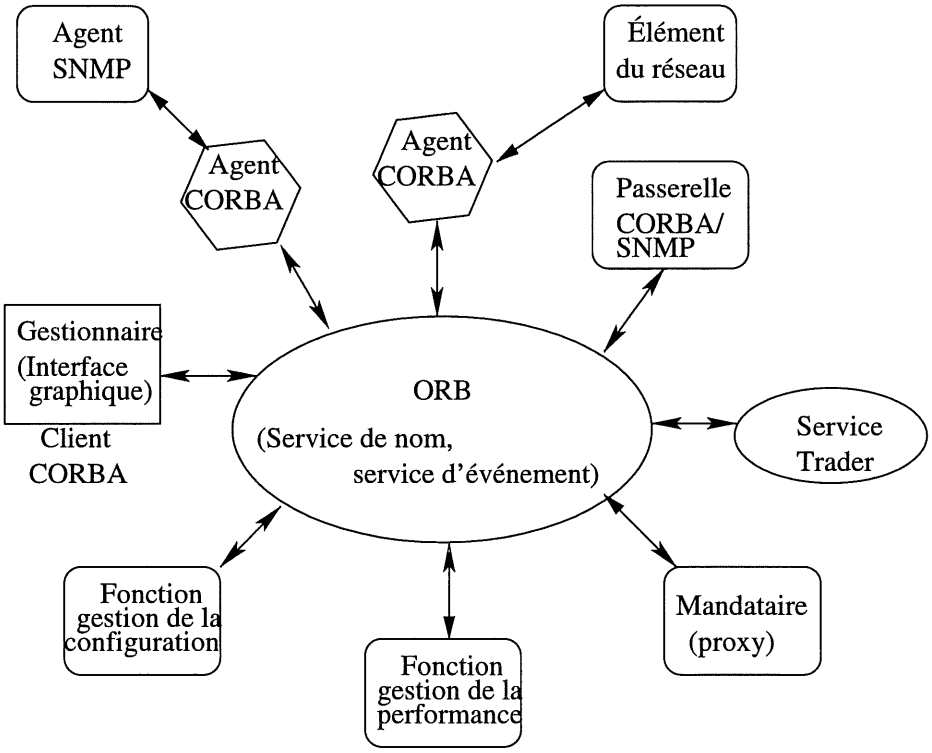

FIG. 18 – *Schéma synoptique de notre architecture*

les requêtes CORBA avant de les transmettre à la passerelle SNMP, gère les exceptions et uniformise l'accès aux différentes opérations SNMP. Ce composant assure une indépendance entre les composants de l'application de gestion et la passerelle CORBA-SNMP.

La passerelle CORBA-SNMP est développée par la compagnie AdventNet [16]. Elle propose une librairie complète d'API SNMP sous forme de classes Java. Elle offre une interface CORBA-IDL à ces API. L'interprétation de la MIB SNMP et la traduction des requêtes CORBA-SNMP se font au moment de l'appel de cette API.

Notre application s'appuie sur la norme ISO [44] pour définir les interfaces IDL des fonctions de gestion. Celles-ci sont élémentaires et offrent les services de base de la gestion des réseaux. Elles peuvent provenir de fournisseurs différents, à condition que les interfaces IDL soient respectées.

Tous les composants de notre application utilisent les services Nom et Vendeur de CORBA. Nous avons développé le service Vendeur afin de permettre l'enregistrement et la recherche des fonctions et des *agents* CORBA selon leurs types. Le *gestionnaire* utilise ce service pour s'informer sur les fonctions et les *agents* disponibles sur le réseau.

Afin d'illustrer notre solution, nous avons développé les opérations de base de la fonction de gestion de la configuration. Ces opérations sont l'inventaire des ressources disponibles, l'accès aux informations de gestion et l'analyse de la configuration du réseau.

Dans notre application, nous avons défini des *agents* CORBA pour représenter les ressources à gérer du réseau. Ces *agents* fournissent des opérations de gestion pour assurer certains traitements au niveau de la machine sur laquelle ils s'exécutent. Parmi ces opérations, nous avons développé une opération qui donne les caractéristiques de la ressource, telles que le nom de la machine, l'information sur la présence ou non d'*agent* SNMP et le type de MIB utilisé.

Nous avons utilisé la méthodologie orientée objet *Unified Modeling Language* (UML)

[23] pour concevoir notre application répartie.

Dans la suite de ce chapitre, nous présentons les cas d'utilisation, les diagrammes de classes et les diagrammes de séquences de cette application.

3.1.2 Cas d'utilisation

Le principal utilisateur de notre application est l'administrateur (figure 19). Il gère les ressources disponibles sur le réseau à travers le *gestionnaire* de l'application. L'administrateur accède aux informations de gestion des ressources du réseau. Il détecte et utilise les services des fonctions de gestion et des *agents* CORBA disponibles sur le réseau.

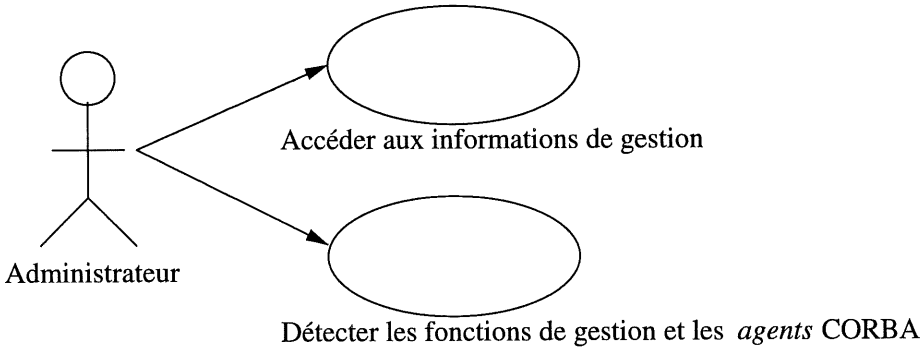

FIG. 19 – *Tâches effectuées par l'administrateur*

Après la détection de la fonction de gestion de la configuration, celle-ci active les menus correspondant à ses opérations (figure 20). L'administrateur les exploite pour assurer l'inventaire des ressources matérielles connectées au réseau, l'analyse des changements de la configuration, la récupération des propriétés d'un *agent* SNMP et la vérification de la présence d'une ressource ou d'un *agent* SNMP sur le réseau.

3.1.3 Diagrammes de classes

Nous présentons d'abord le diagramme de classes du *gestionnaire* (figure 21) qui représente un élément essentiel de notre application. C'est le principal composant, qui

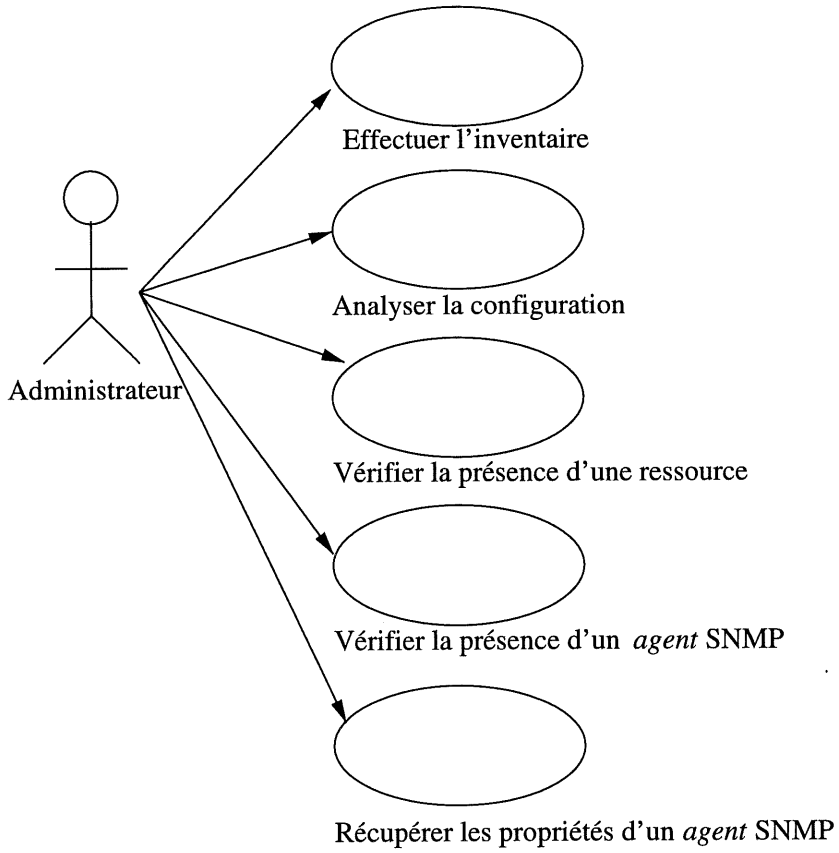

FIG. 20 – *Tâches effectuées par l'administrateur après la détection de la fonction de gestion de la configuration*

communique et coordonne les différentes tâches de l'administrateur. Il utilise les fonctionnalités offertes par les autres composants de l'application. Il comprend la classe principale *UserInterface* qui présente l'interface graphique de l'application.

Au lancement du *gestionnaire*, la classe *UserInterface* fait appel à la classe *InitConfiguration* qui charge les informations de la configuration à partir du fichier *prefSnmp.cfg*. Celui-ci contient les paramètres de la configuration de l'ORB et de SNMP. Parmi ces paramètres, il y a le nom du serveur de l'ORB, le numéro du port de l'ORB, le mot de passe de la communauté SNMP et le nom du fichier de la spécification IDL de la MIB SNMP utilisée.

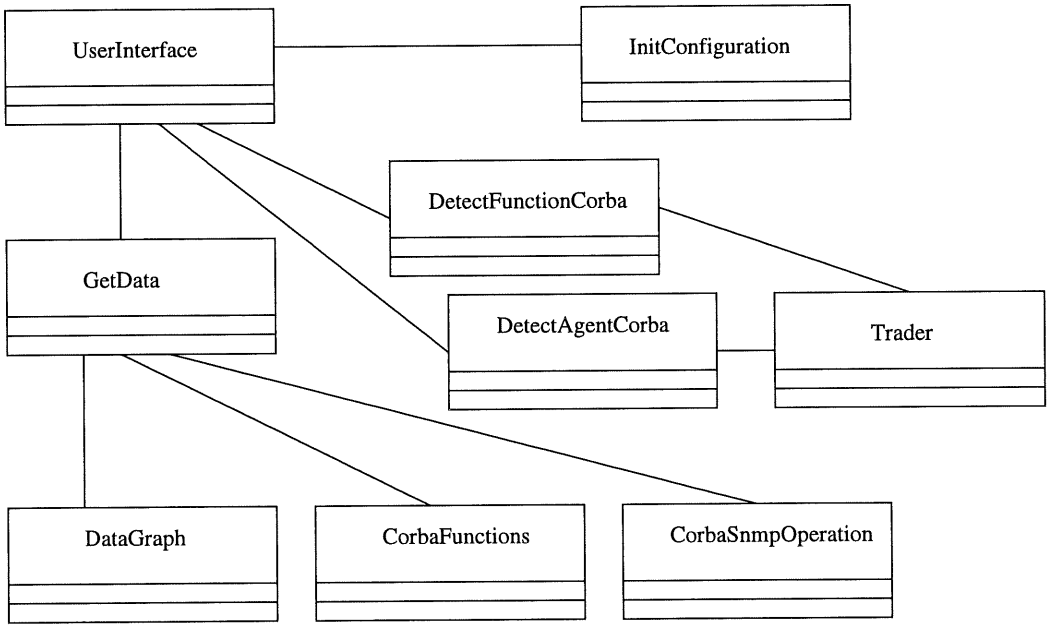

FIG. 21 – *Diagramme de classes du gestionnaire*

À partir de la classe *UserInterface*, deux processus sont lancés pour détecter les fonctions de gestion et les *agents* CORBA disponibles sur le réseau. Ces processus utilisent les classes *DetectFunctionCorba* et *DetectAgentCorba*. Ces mêmes classes utilisent la classe *Trader* pour obtenir la liste des objets CORBA disponibles sur le réseau. La classe *Trader* représente le service annuaire de type pages jaunes de CORBA.

La classe *UserInterface* fait appel à la classe *GetData* pour récupérer les informations de gestion à partir des *agents* SNMP et pour appeler les opérations des fonctions de gestion disponibles sur le réseau. Cela est accompli respectivement par la classe *CorbaSnmpOperation* et la classe *CorbaFunctions*. La première classe offre les opérations de base SNMP en appelant les opérations du composant *ProxyServerCorba* (figure 22). La deuxième classe accède aux opérations des fonctions qui sont détectées sur le réseau.

La classe *DataGraph* offre les opérations de représentation graphique des informations de gestion, qui facilite l'interprétation de ces informations par l'administrateur.

Dans le diagramme de classes à la figure 22, nous présentons les relations entre les différents composants qui interviennent dans notre application. La classe *AgentCorbaServer* modélise *l'agent* CORBA. Elle assure son enregistrement auprès de l'instance de la classe *Trader*. La classe *ManagementConfigServer* modélise la fonction de gestion de la configuration. Elle assure l'enregistrement de celle-ci auprès de l'instance de la classe *Trader*. Une fois enregistrée, la fonction de gestion utilise les services de la classe *ProxyServerCorba*. Cette dernière utilise les opérations de la classe *PasserelleCorbaSnmp* pour accéder aux informations de gestion.

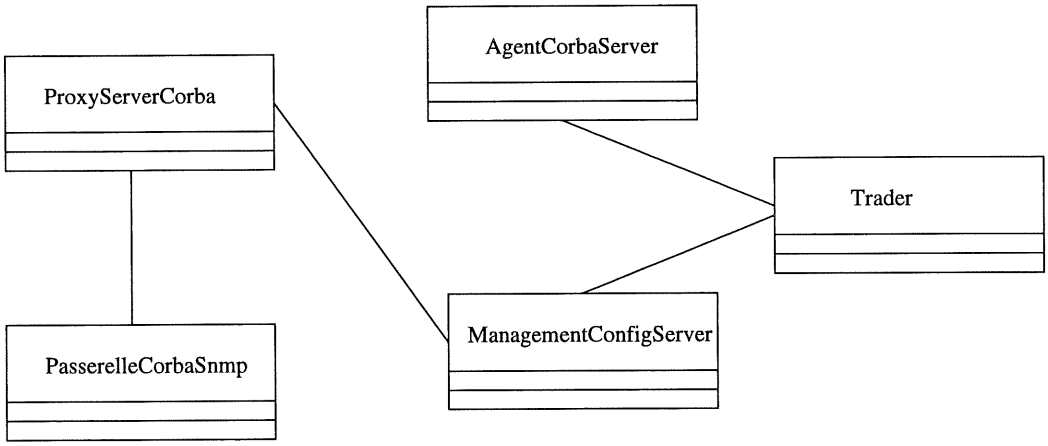

FIG. 22 – *Diagramme de classes des objets serveurs de notre application*

Les classes *AgentCorbaServer*, *ProxyServerCorba*, *ManagementConfigServer* et *Trader*

représentent des composants répartis. Nous avons défini une interface IDL pour chaque composant. Elle décrit les opérations et les attributs accessibles aux objets extérieurs.

La classe *ManagementConfigServer* offre les opérations suivantes :

- Ping : vérifier si un élément est disponible sur le réseau.
- DiscoverNode : découvrir les éléments disponibles sur le réseau à partir d'une plage d'adresses IP.
- IsSnmp : vérifier si un élément sur le réseau est équipé d'un *agent* SNMP ou non.
- SearchNewOldNode : assurer l'analyse de la configuration actuelle et la comparer avec la précédente.
- InfoAgentSnmp : donner les informations de base de l'*agent* SNMP (ex., nom, description et localisation physique de la ressource).
- DePlugFunction : débrancher la fonction du réseau.

La classe *ProxyServerCorba* offre les opérations suivantes :

- CorbaGetTable : lire les attributs tabulaires de la MIB SNMP via CORBA.
- CorbaSetVar : modifier les valeurs des attributs de la MIB SNMP via CORBA.
- CorbaGetVar : lire les valeurs des attributs simples de la MIB SNMP via CORBA.
- CorbaGetNext : lire la valeur du prochain attribut simple de la MIB SNMP via CORBA.

La classe *Trader* offre les opérations suivantes :

- Add : enregistrer le nom d'un objet CORBA en lui associant un type. Ce type peut être une fonction de gestion (FCT) ou un *agent* CORBA (AGT).
- Remove : supprimer le nom d'un objet CORBA et son type.
- Get : donner la liste des objets CORBA suivant le type spécifié.

La classe *AgentCorbaServer* offre les opérations suivantes :

- InfoAgentCorba : donner les informations de base telles que l'adresse IP et le nom de la machine sur laquelle l'*agent* est en cours d'exécution.
- DePlugAgent : débrancher l'*agent* du réseau.

3.1.4 Diagrammes de séquences

Alors que les diagrammes de classes UML présentés ci-dessus représentent le modèle statique de l'application, son modèle dynamique est représenté par des diagrammes de séquences. Ces derniers illustrent les interactions des différents objets composant notre application. Nous présentons les diagrammes de séquences des principales tâches effectuées par l'administrateur. Ces tâches sont l'accès aux informations de gestion, l'établissement d'un inventaire, l'analyse des configurations et la détection dynamique des fonctions de gestion et des *agents* CORBA.

Concernant l'accès aux informations de gestion (figure 23), l'administrateur choisit une des opérations SNMP. Cette dernière est prise en charge par l'instance de la classe *UserInterface* qui la transmet à l'instance de la classe *GetData,* qui elle la transmet à l'instance de la classe *CorbaSnmpOperation.* L'instance de *CorbaSnmpOperation* appelle l'opération SNMP sur l'objet distant *ProxyServerCorba*, puis celui-ci valide et transmet cette requête à l'instance de la classe *PasserelleCorbaSnmp.* Le résultat est structuré au niveau du *ProxyServerCorba* et transmis ensuite à l'administrateur.

La détection automatique consiste à découvrir, au cours de l'exécution du *gestionnaire,* les fonctions de gestion et les *agents* CORBA disponibles sur le réseau. Les interfaces IDL de ces fonctions et *agents* sont connues à la compilation de l'application. Le *gestionnaire* intègre et utilise automatiquement les opérations disponibles dans ces fonctions de gestion et ces *agents* CORBA.

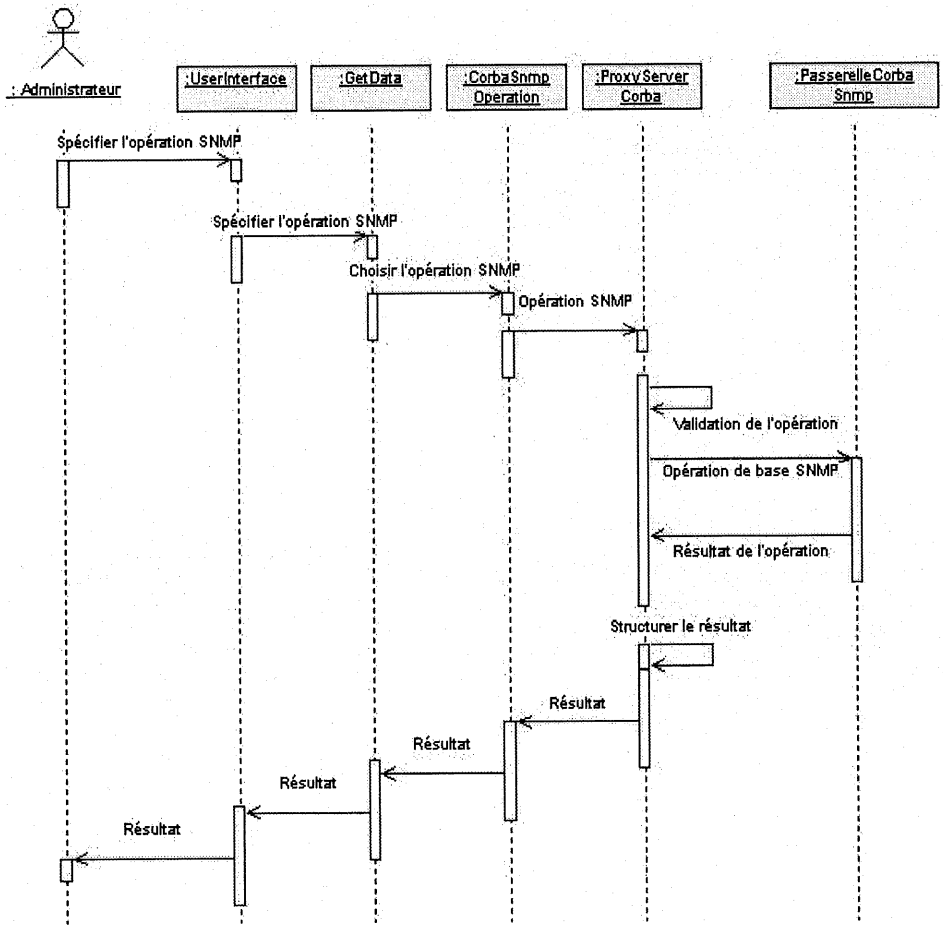

FIG. 23 – *Accès aux informations de gestion*

La détection dynamique des fonctions de gestion et des *agents* CORBA (figure 24) se fait par l'intermédiaire de l'instance de la classe *UserInterface*. Cette dernière transmet une requête de détection des fonctions de gestion à l'instance de la classe *DetectFunctionCorba* et une requête de détection des *agents* CORBA à l'instance de la classe *DetectAgentCorba*. Ces deux instances transmettent périodiquement des requêtes de recherche de fonctions et *d'agents* CORBA à l'instance de la classe *Trader*. Le résultat de cette recherche est analysé par l'instance correspondante *DetectFunctionCorba* ou *DetectAgentCorba*. Cette analyse consiste à vérifier la présence réelle sur le réseau des fonctions ou *agents* enregistrés sur le *Trader* en appelant une de leurs opérations. Le résultat de cette vérification donne la liste réelle des fonctions ou *agents* disponibles sur le réseau. Cette liste est transmise au *Trader* pour mettre à jour la liste correspondante. Enfin, le résultat obtenu par les instances des classes *DetectAgentCorba* ou *DetectFunctionCorba* est communiqué à la classe *UserInterface*.

L'enregistrement auprès du Trader se fait au moment de l'exécution des fonctions de gestion ou des *agents* CORBA. Par contre, l'annulation de l'enregistrement se fait automatiquement par les instances *DetectAgentCorba* et *DetectFunctionCorba* qui détectent la déconnexion de ces fonctions et *agents* et mettent à jour la liste correspondante au niveau du Trader.

La demande d'inventaire des ressources matérielles disponibles sur le réseau est déclenchée par l'administrateur (figure 25). Elle est prise en charge par les instances des classes *UserInterface* et *GetData*. Cette dernière utilise l'opération *Découvrir les ressources disponibles* du composant distant *ManagementConfigServer*. Celui-ci transmet des requêtes *Ping* pour toutes les adresses IP spécifiées pour déterminer les ressources disponibles sur le réseau. Le résultat est analysé par l'instance *ManagementConfigServer* et communiqué ensuite à l'instance *GetData* qui structure et transmet l'information à l'administrateur.

La demande d'analyse de la configuration du réseau (figure 26) est déclenchée par l'administrateur. Elle est prise en charge par les instances des classes *UserInterface* et

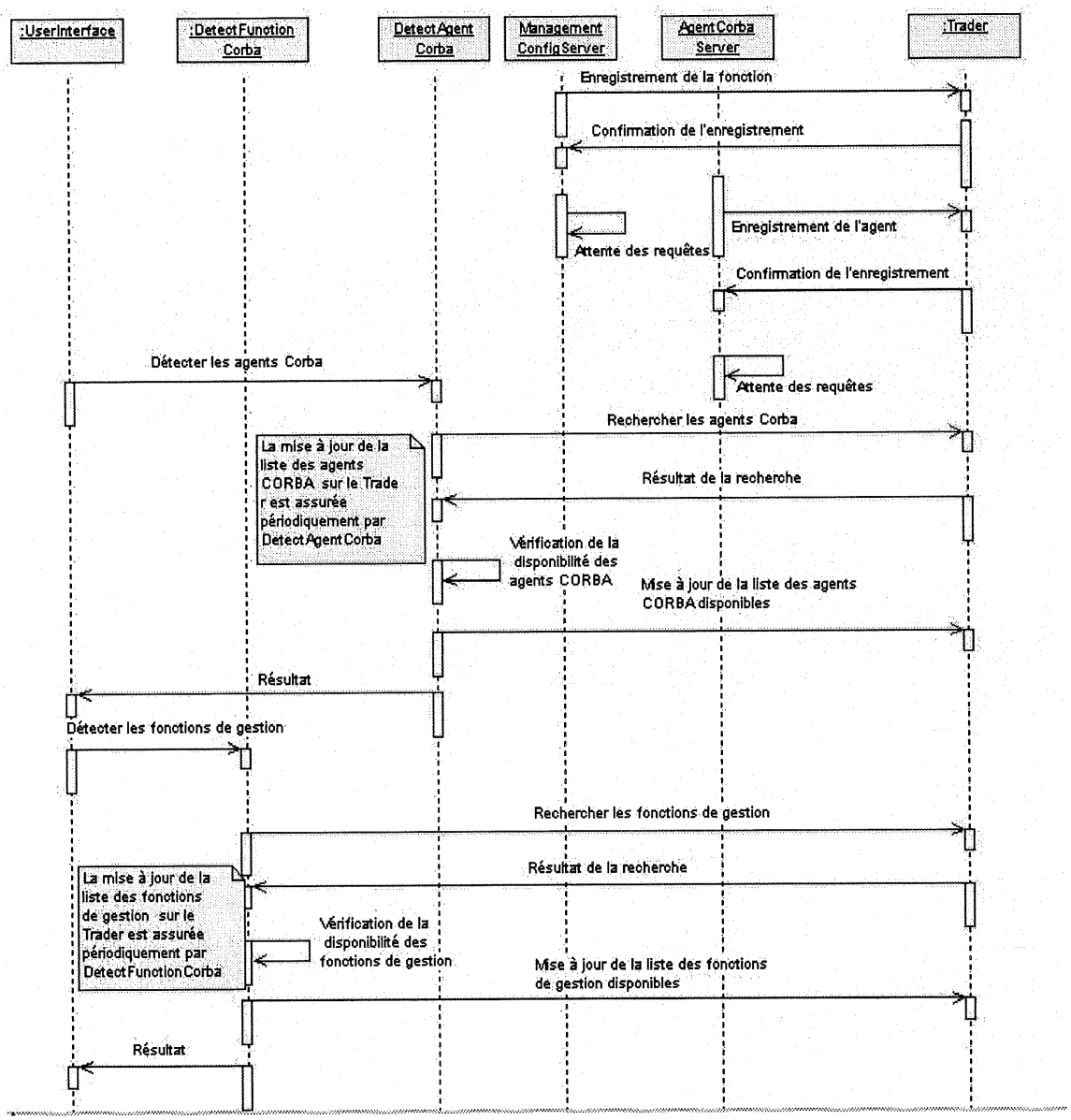

FIG. 24 – *Détection dynamique des fonctions de gestion et des agents CORBA*

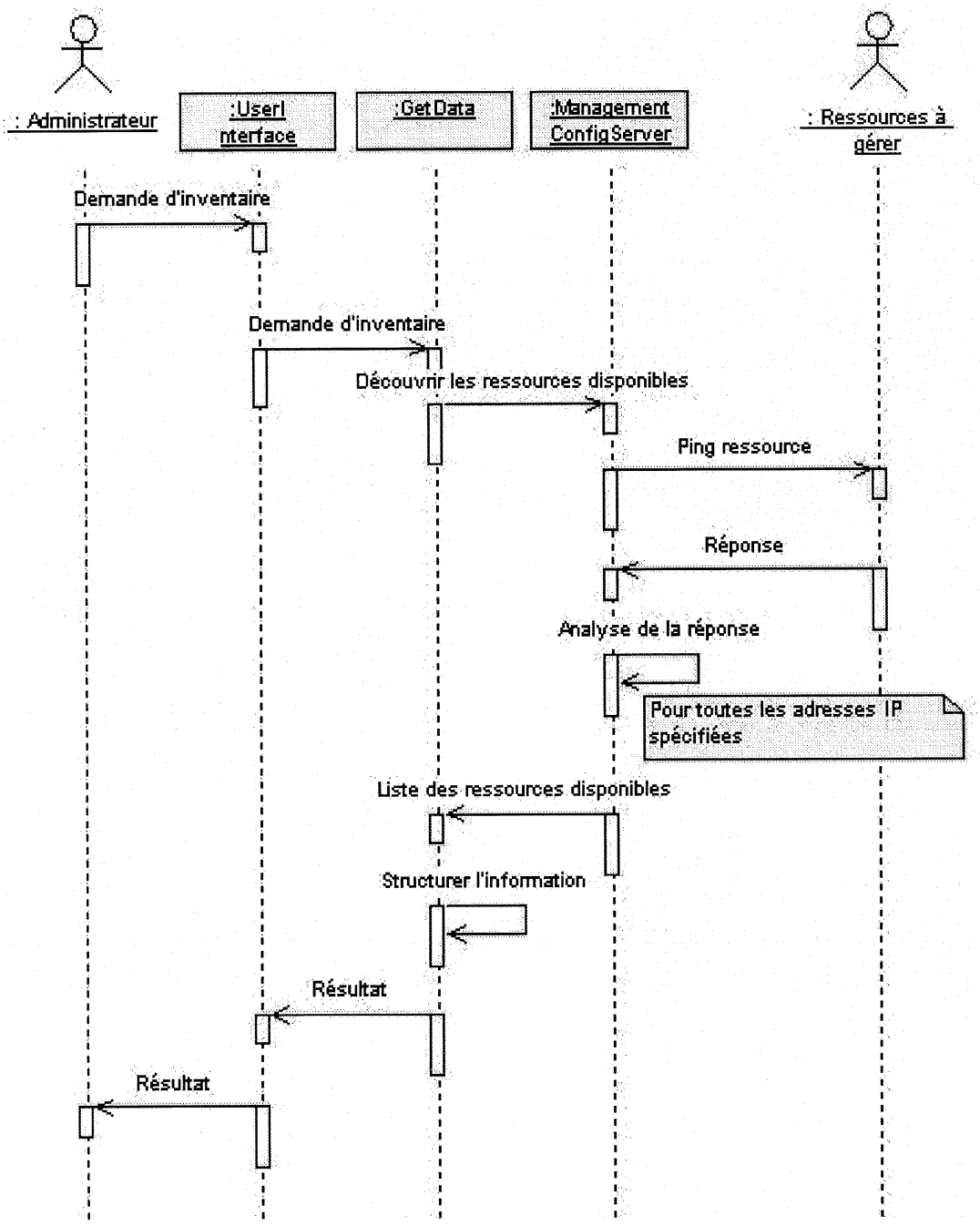

FIG. 25 – *Inventaire des ressources disponibles sur le réseau*

GetData. Cette dernière transmet la requête d'analyse de la configuration au composant distant *ManagementConfigServer*. Ce composant détermine la configuration actuelle et la compare à la précédente. Le résultat de la comparaison est communiqué à l'administrateur.

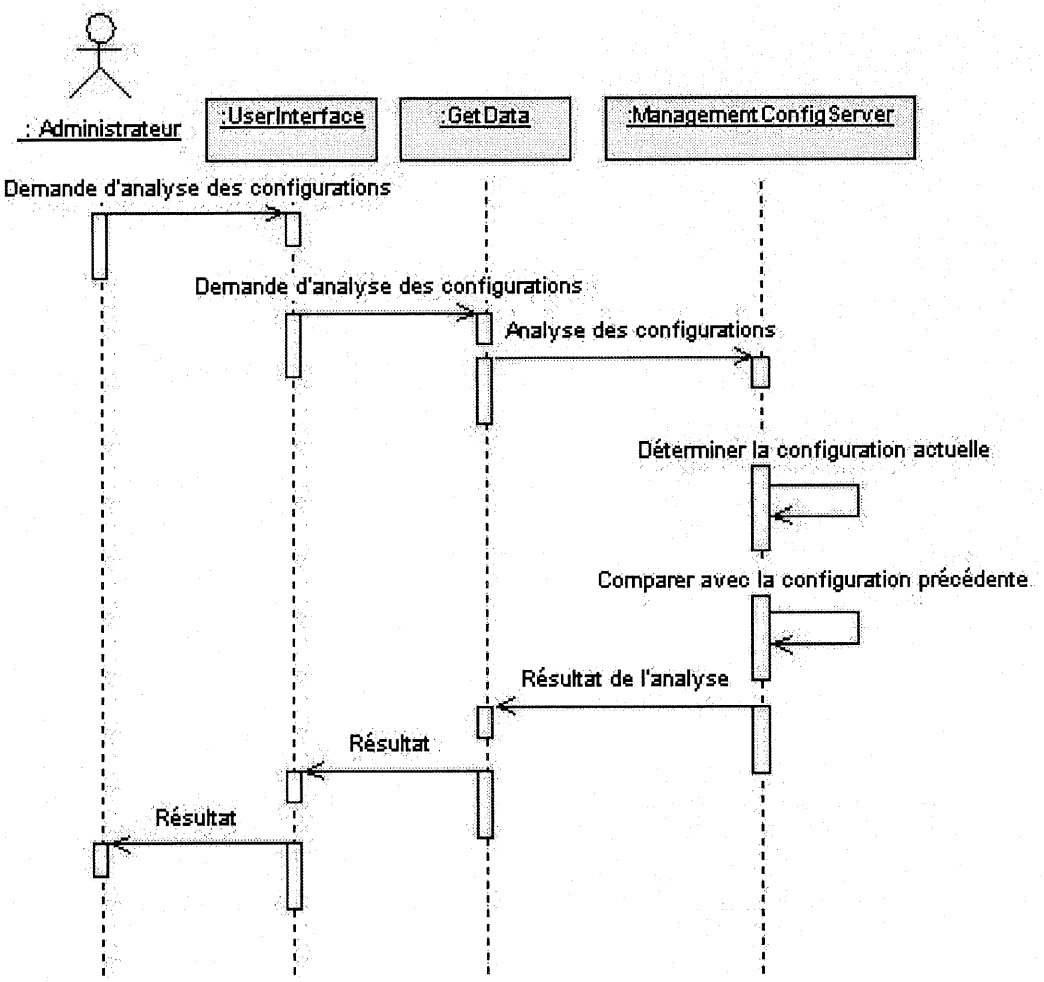

FIG. 26 – *Analyse des configurations*

3.2 Particularités de notre architecture

3.2.1 Transparence d'accès aux MIB SNMP

L'accès aux MIB SNMP se fait à travers le composant *ProxyServerCorba* qui joue le rôle d'intermédiaire entre les composants CORBA de l'application et la passerelle CORBA-SNMP. Cette séparation assure une transparence d'accès aux MIB et une indépendance par rapport à la passerelle CORBA-SNMP utilisée. Le changement de la passerelle n'influe pas sur les composants de l'application. Il suffit de mettre à jour l'implantation du composant *ProxyServerCorba* par rapport à la nouvelle passerelle. Après cette modification, le système fonctionnera à nouveau sans modifier les autres composants de l'application.

3.2.2 Détection dynamique des fonctions de gestion des réseaux

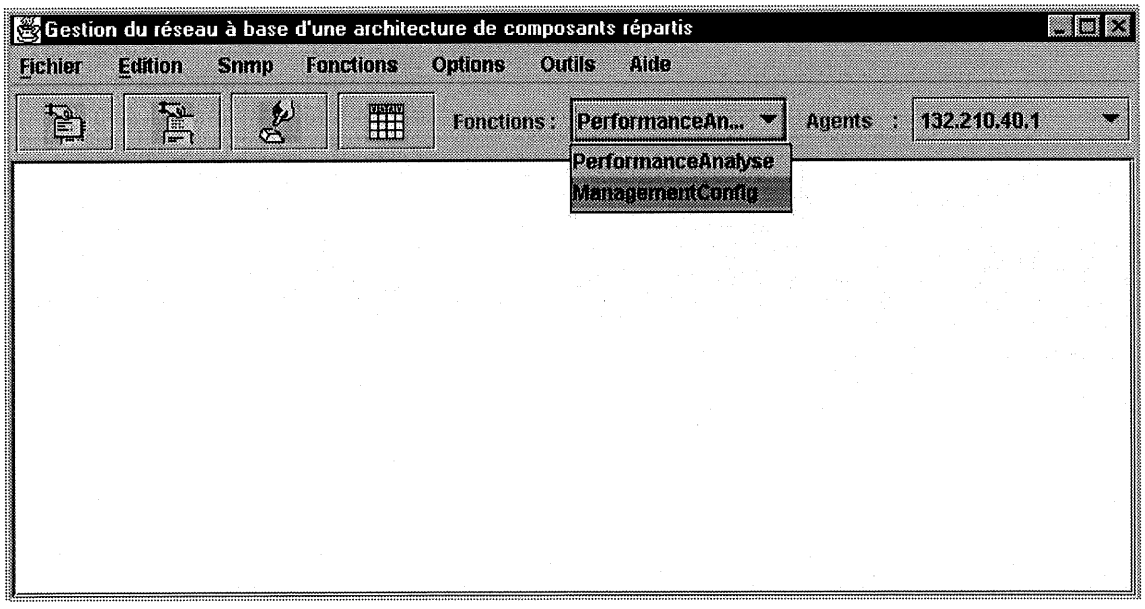

FIG. 27 – *Détection automatique des fonctions de gestion*

Dans notre approche, le *gestionnaire* améliore ses services en accédant à d'autres fonctions de gestion sur le réseau. Il les découvre pendant son exécution. Au démarrage, le *gestionnaire* lance un programme qui s'exécute en tâche de fond pour détecter l'arrivée des nouvelles fonctions (figure 27). Celles-ci peuvent être lancées sur plusieurs ordinateurs du réseau. Une fois qu'elles sont détectées, le *gestionnaire* active le menu correspondant, ce qui offre des fonctionnalités additionnelles à l'administrateur. Le *gestionnaire* détecte automatiquement la déconnexion de ces fonctions. Cela est assuré par un processus qui s'exécute en tâche de fond et vérifie périodiquement leur présence. Une fois déconnectées, les menus offerts par ces fonctions sont désactivés automatiquement au niveau du *gestionnaire.*

Afin d'intégrer de nouvelles fonctions de gestion dans notre application, le développeur doit respecter les interfaces IDL que nous avons définies pour ces fonctions. Par exemple, l'interface IDL de la fonction de gestion de la configuration, présentée ci-dessous, définit les opérations *Ping, DiscoverNode, IsSnmp, SearchNewOldNode, InfoAgentSnmp* et *DeplugFunction* (voir le paragraphe 3.1.3). L'opération *Ping* génère une exception en cas d'erreur. Chaque opération possède des paramètres d'entrée et de sortie que le développeur doit respecter pour implanter une de ces opérations. L'ensemble des opérations et des attributs est regroupé dans un module appelé *ManagementConfigApp.*

```
module ManagementConfigApp
{
typedef sequence <string>stringSeq ;
typedef sequence <stringSeq>stringMatrix ;
exception SnmpException
{
        string raison ;
} ;
```

```
interface ManagementConfig
{
boolean Ping(in string adrIp) raises (SnmpException) ;
stringSeq DiscoverNode(in string ip1,in string ip2,in string ip3) ;
boolean IsSnmp(in string adrIp) ;
stringSeq SearchNewOldNode(in stringSeq lastTIp,in stringSeq newTIp) ;
stringSeq InfoAgentSnmp(in string nodeIp) ;
void DePlugFunctions(in string name, in string service-id) ;
} ;
} ;
```

3.2.3 Détection dynamique des *agents* CORBA sur le réseau

Un *agent* CORBA est identifié par son adresse IP. Il s'enregistre auprès des services Nom et Vendeur de CORBA. Le *gestionnaire* détecte automatiquement la présence d'un *agent* sur le réseau. Cet *agent* offre une opération de gestion qui communique les caractéristiques de la ressource, telles que le type de MIB utilisé, le nom et l'adresse IP de la machine. Chaque *agent* CORBA représente une ressource sur le réseau.

Le *gestionnaire* détecte automatiquement la déconnexion d'un *agent* CORBA et met à jour la liste des *agents* disponibles sur le réseau (figure 28).

3.2.4 Définition des paramètres d'entrée de notre *gestionnaire*

Nous avons défini deux fichiers de configuration. Le premier fichier, *PrefSnmp.cfg,* contient les informations concernant l'environnement SNMP et les paramètres de l'ORB. L'administrateur peut changer ces paramètres en utilisant l'interface graphique du *gestionnaire* (figure 29). La description détaillée de ces paramètres se présente comme suit :

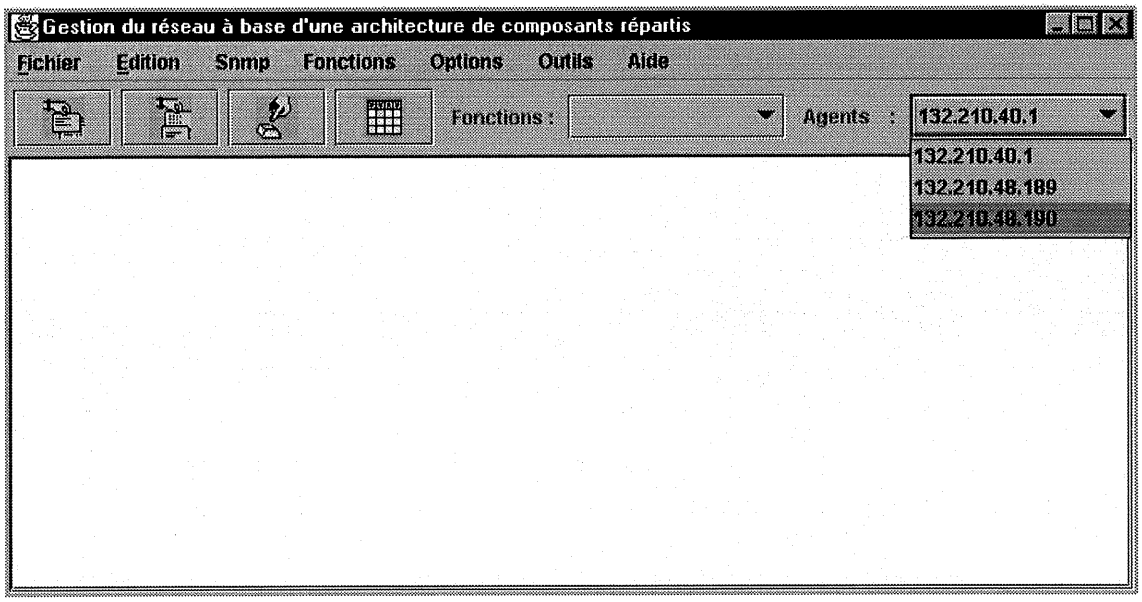

FIG. 28 – *Détection automatique d'un agent CORBA*

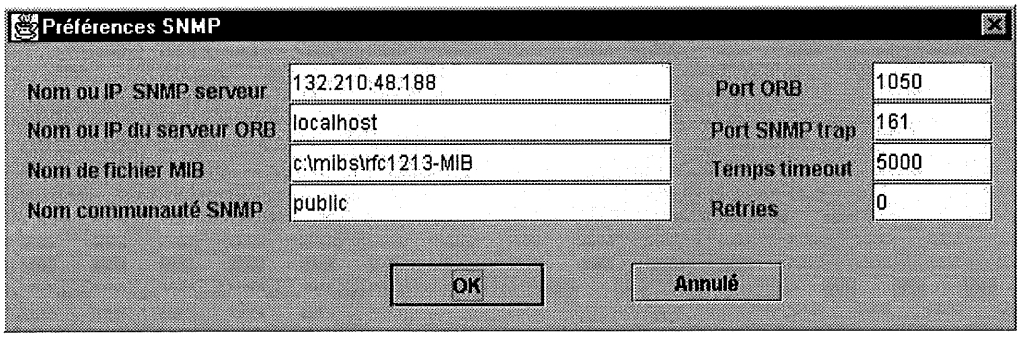

FIG. 29 – *Mise à jour des paramètres de la configuration de l'application*

- SnmpServer : adresse IP du serveur sur lequel la passerelle CORBA-SNMP est en cours d'exécution.
- ORBInitialHost : adresse IP du serveur sur lequel l'ORB est en cours d'exécution.
- ORBInitialPort : numéro de port de l'ORB.
- MibFile : nom du fichier qui contient la spécification de la MIB en IDL.
- PwdSnmp : mot de passe de la communauté SNMP.
- PortSnmpTrap : numéro de port via lequel le *gestionnaire* SNMP reçoit les alarmes.
- Tempstimeout : le temps d'attente avant la retransmission de la requête SNMP en cas d'absence de réponse de l'*agent*.
- Retries : nombre de retransmissions après le timeout.

Le deuxième fichier, *network.cfg,* contient les plages d'adresses IP qu'il faut explorer pour établir l'inventaire. Il est constitué de plusieurs lignes de données. Chaque ligne du fichier représente les valeurs décimales des trois premiers octets de l'adresse IP. Le troisième octet peut être représenté par le caractère « * » pour indiquer un intervalle de valeurs allant de 0 à 255. Nous présentons ci-dessous un exemple du contenu de ce fichier :

```
132 210 47
132 211 *
```

La donnée 132.210.47 indique que les adresses à explorer vont de 132.210.47.0 à 132.210.47.255. Par contre, la donnée 132.211.* indique que les adresses à explorer vont de 132.211.0.* à 132.211.255.* en variant le troisième octet de 0 à 255.

3.2.5 Apport des *agents* CORBA dans notre solution

L'utilisation des *agents* CORBA permet de diminuer le temps nécessaire pour établir un inventaire des ressources disponibles sur le réseau. Le traitement classique utilisé consiste à transmettre séquentiellement des requêtes *Internet Control Message Protocol* (ICMP) à toutes les adresses IP comprises dans un intervalle donné. Cela veut dire que si

une requête ICMP s'exécute en T secondes et que nous avons N adresses IP, la durée d'un processus d'inventaire est égale à (N x T) secondes. Par contre, avec les *agents* CORBA, toutes les ressources sont déjà inscrites auprès du service Vendeur. L'inventaire consiste à transmettre une requête au service Vendeur de CORBA pour récupérer la liste des *agents* disponibles. Le temps requis pour l'exécution de cette opération est égal au temps nécessaire pour l'appel d'une opération CORBA. D'après les tests de l'exécution de notre application sur un réseau local, la durée d'une seule requête Ping est approximativement égale à la durée d'une requête d'accès au composant Trader de notre application. Cela montre que l'utilisation des *agents* CORBA réduit considérablement le temps nécessaire à l'établissement d'un inventaire.

3.3 Implantation

Cette section présente les technologies disponibles pour l'implantation des applications de gestion distribuées, de même que l'implantation physique de notre application en spécifiant la technologie utilisée.

3.3.1 Technologies disponibles pour le développement d'applications de gestion

Des groupements de constructeurs et d'éditeurs de logiciel ont proposé plusieurs architectures pour le développement d'applications de gestion basées sur des environnements distribués.

Java *Management eXtensions* (JMX) (autrefois Java *Management API*) [15] est une architecture proposée par plusieurs organismes, entre autres, JavaSoft, BMC, CISCO et 3COM. Elle fournit des services et une API pour développer des solutions de gestion de réseau basées sur l'environnement Java et Java *Dynamic Management Kit* (JDMK) [21].

JMX fournit également des outils pour développer des solutions distribuées, modulaires et dynamiques pour la gestion des ressources et des services sur les réseaux. Il supporte les protocoles IIOP, CMIP et SNMP.

Web Based Enterprise Management (WBEM) [17] est une architecture née d'un regroupement d'efforts de plusieurs grandes compagnies, entre autres, Microsoft, Intel, Compaq et BMC. Leur objectif est la normalisation de la gestion par le WEB en utilisant des navigateurs et en assurant une transparence totale vis-à-vis des plates-formes, des systèmes et des réseaux à gérer. WBEM utilise le nouveau protocole de gestion de réseau *Hyper Media Management Protocol* (HMMP) et le nouveau modèle d'information *Common Information Model* (CIM).

AdventNet SNMP API [16] est une architecture définie par *AdventNet.* Elle fournit un environnement constitué d'un ensemble de classes Java formant ainsi une API destinée à faciliter le développement des applications de gestion. Cette architecture fournit une API pour SNMPv1, SNMPv2 et SNMPv3. Récemment *AdventNet* a intégré les supports de RMI et de CORBA dans son API, ce qui permet d'utiliser son API à distance en utilisant RMI ou CORBA.

JMX et WBEM poursuivent le même objectif, qui est de permettre le développement de solutions de gestion distribuées et indépendantes des plates-formes sous-jacentes, mais elles sont différentes dans leurs approches.

JMX est construite sur des protocoles de gestion et d'accès à l'information existante, et se base sur la portabilité et la flexibilité de Java pour offrir la transparence de la distribution. WBEM introduit un nouveau protocole et un nouveau modèle d'information qui étendent le protocole et le langage du WEB (HTTP/HTML) avec des propriétés spécifiques à la gestion des réseaux et des systèmes. La transparence de la distribution est assurée par un service de gestion de l'accès aux objets appelé *Hyper Media Object Manager* (HMOM). L'adoption de WBEM et sa normalisation dépendent de l'acceptation par l'industrie de ces nouveaux concepts. Des travaux d'intégration des protocoles SNMP

et CMIP vers WBEM sont en cours de réalisation en utilisant des passerelles entre ces différents protocoles.

Enfin, l'architecture de *AdventNet* est différente des deux approches précédentes, car elle offre des API qui sont liées essentiellement au protocole SNMP. Elle fournit des outils de développement graphiques pour améliorer la productivité du développeur dans le domaine de la gestion des réseaux utilisant SNMP.

Notre approche s'inscrit davantage dans la perspective de l'approche JMX, mais comme nous avons commencé notre développement avant la publication de la première version de JMX par Sun, nous avons opté pour l'API *AdventNet* qui était déjà disponible et opérationnelle.

3.3.2 Architecture détaillée de l'implantation

En nous basant sur les diagrammes de classes de notre application, nous avons implanté les composants CORBA *AgentCorbaServer*, *Trader*, *ManagementConfigServer*, *ProxyServerCorba* et les différentes classes qui constituent notre *gestionnaire* de réseau. Nous avons utilisé le langage Java 1.2 [12], l'API *AdventNet*, le service de nom de Java 1.2 et OrbixWeb [42] pour l'implantation de cette application.

Nous avons utilisé le service de nom de Java 1.2 pour la communication entre les composants de notre application. Après l'acquisition d'OrbixWeb par l'Université, nous avons testé le comportement des composants de notre application par rapport au service de nom d'OrbixWeb. Nous avons remarqué que des modifications sont nécessaires au niveau de nos programmes pour permettre à nos composants d'utiliser les services d'OrbixWeb.

Le démarrage de l'application consiste à exécuter les composants *PasserelleCorbaSnmp* (figure 30), *ProxyServerCorba* (figure 31) et *Trader* (figure 32). En fait, ces

composants sont représentés par des objets CORBA ; ils se connectent à l'ORB, s'enregistrent auprès du service Nom de CORBA et se mettent en attente des appels des clients. Ces composants sont exécutés sur des machines différentes en spécifiant l'adresse IP et le numéro de port de l'ORB.

```
java com.adventnet.snmp.corba.server -ORBInitialPort 1050
Factory is ready
Server is ready
Ready to receive client requests . . .
```

FIG. 30 – *Exécution du composant Passerelle CORBA/SNMP*

```
java ProxyServer -ORBInitialPort 1050
Initialisation de l'ORB
Resolution du service Naming
Connexion a la passerelle CORBA/SNMP de AdventNet
Creation de l'objet Proxy et son enregistrement sur l'ORB
Attente des invocations des clients.....
```

FIG. 31 – *Exécution du composant mandataire CORBA*

```
java TraderServer -ORBInitialPort 1050
Initialisation de l'ORB
Connexion au service de nom de l'ORB
Enregistrement aupres du service de nom de l'ORB
Attente des invocations des clients . . .
```

FIG. 32 – *Exécution du composant Trader*

L'exécution du *gestionnaire* affiche une interface graphique (figure 33) qui permet à l'administrateur d'utiliser les services des composants en cours d'exécution.

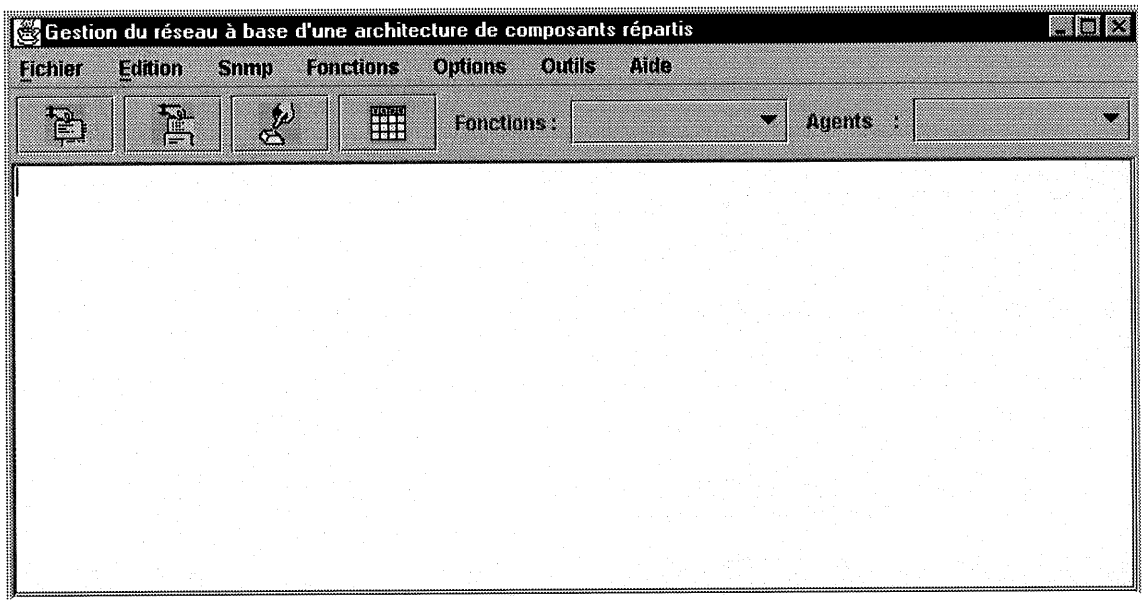

FIG. 33 – *Gestionnaire du réseau (interface utilisateur)*

Au lancement de la fonction de gestion *ManagementConfigServer* et de l'*agent* CORBA *AgentCorbaServer*, le *gestionnaire* les détecte automatiquement et active les menus (figure 34) nécessaires à l'exploitation de ces fonctions et *agents*.

Le *gestionnaire* offre également l'accès aux informations de gestion via les opérations SNMP *CorbaGetVar*, *CorbaSetVar*, *CorbaGetNext* et *CorbaGetTable* disponibles dans le composant *ProxyServerCorba* (figure 35). L'utilisation de ces opérations se fait par l'intermédiaire des interfaces graphiques (figure 36).

L'exécution de l'opération *inventaire* de la fonction de gestion de la configuration *ManagementConfigServer* donne la liste des ressources matérielles disponibles sur le réseau (Routeurs, Switch, PCs, Serveurs, etc.) (figure 37). Cette liste contient les noms et les adresses IP de ces ressources avec l'information sur la présence ou non d'un *agent* SNMP sur chaque ressource.

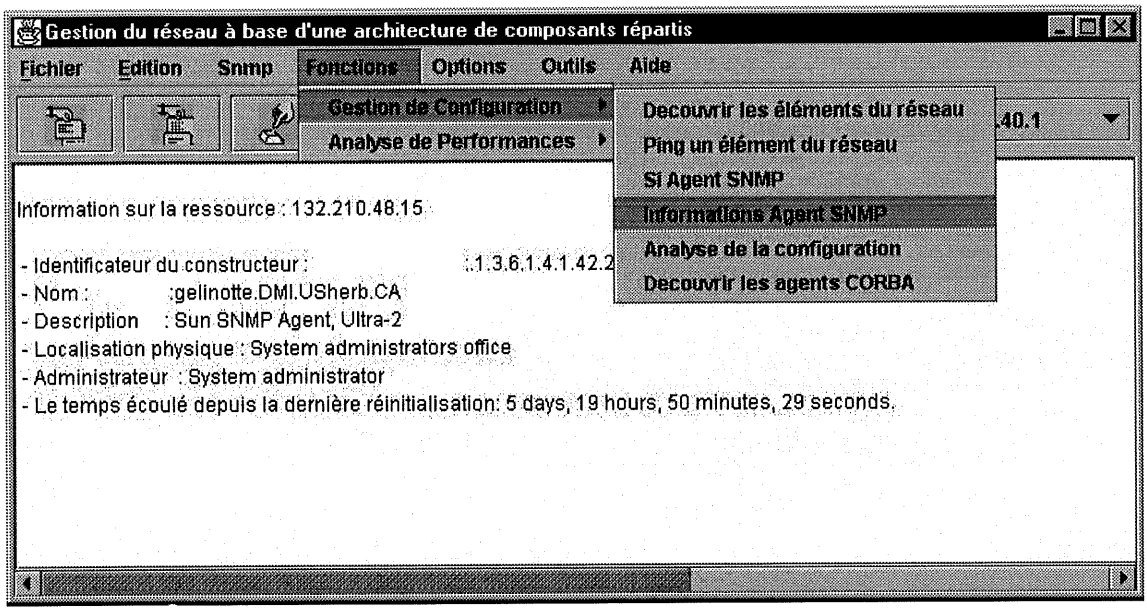

FIG. 34 – *Les opérations offertes par la fonction de gestion de la configuration*

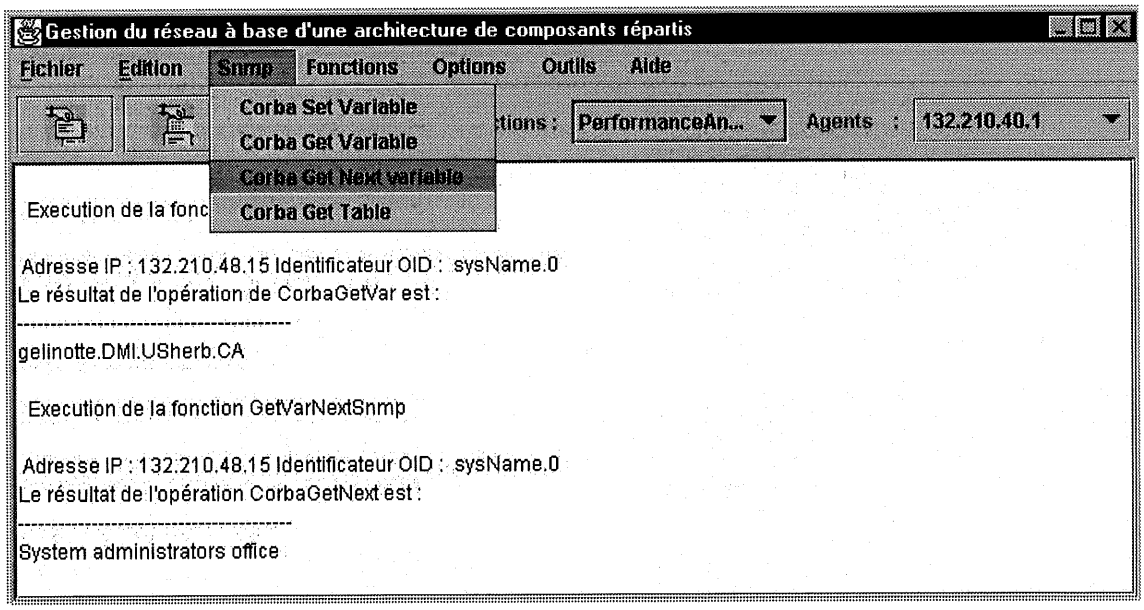

FIG. 35 – *Les opérations CORBA SNMP*

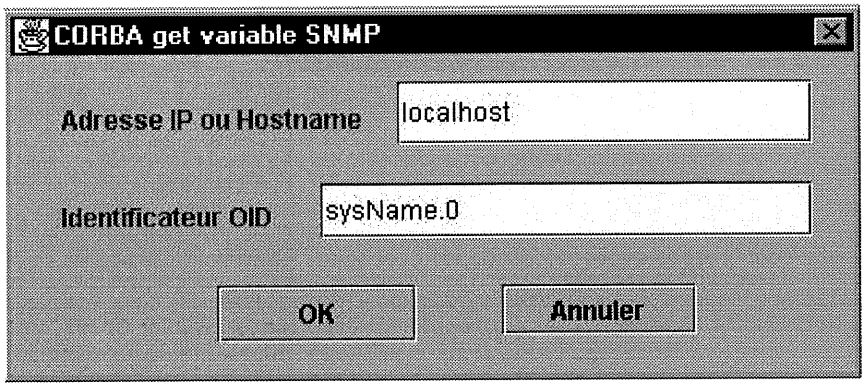

FIG. 36 – *L'opération CorbaGetVar*

Adresse IP	Nom Host	AgentSnmp
132.210.48.15	gelinotte2.DMI.USherb.CA	OUI
132.210.48.20	132.210.48.20	NON
132.210.48.52	pc018-2.DMI.USherb.CA	NON
132.210.48.54	pc018-4.DMI.USherb.CA	NON
132.210.48.55	pc018-5.DMI.USherb.CA	NON
132.210.48.56	pc018-6.DMI.USherb.CA	NON
132.210.48.57	pc018-7.DMI.USherb.CA	NON
132.210.48.141	chouette2.DMI.USherb.CA	OUI
132.210.48.188	lolly2.DMI.USherb.CA	NON
132.210.48.189	molly2.DMI.USherb.CA	NON
132.210.48.190	pc1010-21.DMI.USherb.CA	OUI
132.210.48.210	harfang4.DMI.USherb.CA	OUI
132.210.48.255	132.210.48.255	OUI

FIG. 37 – *Résultat de l'inventaire des ressources du réseau*

3.4 Discussion

Notre réalisation démontre l'apport de CORBA au niveau de l'intégration dynamique de nouvelles fonctions aux applications de gestion des réseaux, et au niveau de la transparence vis-à-vis des protocoles de gestion.

Dans notre architecture, la mission de gestion est distribuée sur plusieurs composants répartis sur le réseau. Cela permet à l'application de gestion de continuer à fonctionner même si une panne survient sur un de ses composants.

De plus, CORBA a apporté une flexibilité au niveau de l'expansion de l'application de gestion en permettant l'intégration dynamique de nouveaux composants, ce qui permet d'ajouter des fonctionnalités à cette application sans modifier les composants existants.

Notre application est fonctionnelle. Nous l'avons testée en exécutant les différents composants (fonction de gestion de la configuration, *agents* CORBA, Proxy, passerelle CORBA-SNMP) sur des ordinateurs distincts. Tous ces composants s'enregistrent sur le même service de nom de Java 1.2. Ils sont utilisés par le gestionnaire à partir d'une station de gestion. Il reste à tester le comportement des composants de notre application sur des ORB différents (OrbixWeb, VisiBroker, etc.).

Conclusion

Dans ce mémoire, nous avons développé une architecture pour faciliter le développement des applications de gestion distribuées, portables et extensibles. Cette architecture est constituée d'un *gestionnaire* de réseau, qui est représenté par un client CORBA, et d'un ensemble de composants répartis. Ces composants représentent des fonctions de gestion, des *agents* CORBA, une passerelle CORBA-SNMP et un *mandataire* contenant les opérations de base SNMP. La communication entre ces composants se fait à travers l'Intergiciel CORBA.

Contributions

Cette architecture est flexible. Elle permet d'intégrer de nouveaux composants fonctionnels à l'application de gestion (*plug and play*). Cela permet à l'administrateur de composer son application de gestion selon ses besoins. Une panne d'un composant de l'application n'affecte pas la fonctionnalité des autres composants.

Tous les composants de notre architecture peuvent être développés sur des plates-formes différentes et avec des langages différents.

Cette approche permet une mise en place distribuée des applications de gestion. De multiples instances de chaque composant de l'application peuvent coexister sur le réseau. Dans notre architecture, la répartition des tâches de gestion à travers des composants CORBA permet de séparer l'interface utilisateur et les fonctions de gestion. De plus, les

fonctions de gestion peuvent être branchées sur le réseau d'une manière dynamique, ce qui permet l'intégration de nouvelles fonctionnalités à l'application de gestion, selon les besoins de l'administrateur.

L'utilisation du composant *mandataire* dans notre architecture permet d'assurer une indépendance entre les composants de l'application et la passerelle CORBA-SNMP utilisée.

La représentation des ressources à gérer par des *agents* CORBA facilite la localisation de ces ressources sur le réseau. Ces *agents* déchargent l'application de gestion de certains traitements qui sont assurés sur la ressource.

Ce travail s'inscrit dans les tendances actuelles des applications sur Internet, à savoir la répartition transparente et les liaisons dynamiques entre les unités d'exécution d'une application.

Critiques du travail

Dans notre architecture, les interfaces IDL des fonctions de gestion doivent être connues au moment de la compilation des différents modules de l'application. Cela ne permet pas la détection et l'utilisation des fonctions non prévues dans les interfaces IDL de notre application.

L'accès à l'*agent* SNMP doit passer par le composant *mandataire* et la passerelle CORBA-SNMP, ce qui augmente le nombre de messages de contrôle sur le réseau.

Si une anomalie survient sur l'ORB, les composants de notre application ne pourront pas s'enregistrer, ce qui engendrera des problèmes au niveau de l'exécution de notre application.

Travaux futurs de recherche

Comme perspectives pour ce travail, nous pensons particulièrement à :

- Approfondir l'étude du concept de détection dynamique des fonctions de gestion, en utilisant les mécanismes d'appel *Dynamic Interface Invocation* (DII) et *Interface Repository* (IR) de CORBA.

- Développer la mission de l'*agent* CORBA en assurant, par exemple, la gestion des alarmes sur les ressources à gérer, ce qui évitera à l'*agent* SNMP de transmettre toutes les alarmes au *gestionnaire*. L'*agent* CORBA assurera le traitement local de ces alarmes. Il ne transmettra au *gestionnaire* que l'information pertinente.

- Faciliter l'accès à notre application en utilisant une interface HTTP et un navigateur Web. Cela permettra à l'administrateur d'utiliser notre application à partir de n'importe quelle station du réseau en utilisant un navigateur Web.

- Restreindre l'utilisation des fonctions de gestion aux administrateurs autorisés pour des raisons de sécurité.

Perspective

Actuellement, l'OMG prépare la future norme CORBA 3.0 qui assurera le passage du modèle objet au modèle de composants CORBA. Celui-ci prend en compte les aspects de diffusion et de déploiement d'applications distribuées [32]. Ce modèle propose une plate-forme de développement et d'exécution de composants CORBA. Il fournit des outils appropriés de développement et d'administration d'applications à base de composants répartis. Nous pourrons l'utiliser pour implanter notre architecture. Présentement, il n'existe pas d'ORB qui implante les spécifications de CORBA 3.0.

Bibliographie

[1] A. Ben Artzy, A. Chandna and U. Warrier. Network Management of TCP/IP Networks : Present and Future. *IEEE Network Magazine*, pages 35–43, 1990.

[2] B. Ban. Towards a Generic Object-Oriented Model Multi-domain Management. Proceedings of Coop'96, Workshop on System and Network Management, 1996.

[3] Bela Ban. Generic Management Model For Corba, CMIP and SNMP . Ph.D. thesis, University of Zurich, 1997.

[4] Sonny Rasmussen & Christoph Baumer. A CORBA to CMIP gateway : A marriage of management technology. Fifth International Conference on Intelligence in Services and Networks, 1998.

[5] Jean Marie Chauvet. *Corba Active X et Java Beans.* Eyrolles, 1997.

[6] D. Harrington, P. Mayer and B. Wijnen. An Architecture for Describing SNMP Management Frameworks. *RFC 2271*, 1998.

[7] L. Deri and Surfini. Network Management Ressources across the Web. Proceedings of 2nd Intl IEEE Workshop on Systems and Network Management, 1996.

[8] Luca Deri and Bela Ban. Static vs. Dynamic CMIP/SNMP Network Management using CORBA. *In Lecture Notes in Computer Science*, 1997.

[9] Robert Orfali et al. *The Essential Client/Server Survival Guide.* Wiley, 1996.

[10] W. Rosenberry et al. *Understanding DCE.* O'Reilly & Associates, Inc., 1994.

[11] Noême Simoni et Simon Znaty. *Gestion de réseau et de service.* InterÉditions, 1997.

[12] Geoffrey Lewis, Steven Barber and Eflen Siegel. *Programming with Java IDL.* John Wiley & Sons, Inc., 1998.

[13] P. Haggerty and K. Seetharaman. The Benefits of Corba-Based Network Management. *Communications of the ACM*, pages 73–79, 1998.

[14] http://developer.java.sun.com/developer.

[15] http://java.sun.com/products/JavaManagement/.

[16] http://www.adventnet.com.

[17] http://www.dmtf.org/wbem/.

[18] http://www.microsoft.com/dcom.

[19] http://www.omg.org.

[20] http://www.rdg.opengroup.org/mem_only/tech/sysman/jidm/index.htm.

[21] http://www.sun.com/software/java-dynamic/.

[22] McConnell Consulting Inc. RMON Methodology . 1997.

[23] J. Rumbaugh, I. Jacobson and G. Bouch. *The Unified Modeling Language Reference Manual.* Addison-Wesley, 1998.

[24] Jean-Marc Glib, Christophe Gransart et Phillipe Merle. *CORBA des concepts à la pratique.* InterÉditions, 1997.

[25] Jean Pavon, Jos Tomas, Yves Bardout and Linda Helene. CORBA for network and service management in the TINA framework. *IEEE Communication Magazine*, 36(3):72–79, 1998.

[26] Mohsen Kahani and Peter Beadle. Decentralized Approaches for Network Management. *ACM Computer Communication Review Journal*, 1997.

[27] P. Kalyanasundaram and A.S. Sethi. An application Gateway Design for OSI-Internet Management. *ISINM, San Francisco (Elsiever Science Publisher B.V. (North Holland))*, 1993.

[28] Markku Lankkahen. CORBA/SNMP management using gateway approach with IDL translation. *NOMS98*, 1:276–9, 1998.

[29] The Open Group Systems Management. Common Management Facilities. *CAE Specification C423*, 1997.

[30] Matt White, Carnegie Mellon and Smitha Gudur. An Overview of the AgentX Protocol. *The Simple Times*, 6(1), 1998.

[31] S. Mazumdar. Inter-Domain Management between CORBA and SNMP:Web based Management CORBA/SNMP Gateway Approach. IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, 1996.

[32] OMG TC Document orbos/99 02-05. CORBA Components, Joint revised submission. 1999.

[33] OSF-DME-PD-0394-2. DME Overview. *OSF online document*, 1992.

[34] OMG White paper. CORBA-based telecommunication network management system. Draft, 1996.

[35] Juan Pavon. Building Telecommunication Management Applications with CORBA. *IEEE Communication Surveys*, 1999.

[36] David Perkins and Evan McGinnis. *Understanding SNMP MIBs.* Prentice Hall PTR, 1997.

[37] S. Abeck, A. Clemm and U. Hollberg. Simply Open Network Management: An Approach for the Integration of SNMP into OSI Management Concepts. *ISINM, San Francisco (Elsiever Science Publisher B.V. (North Holland))*, 1993.

[38] S. Mazumdar, S. Bradly and D.W. Levine. Design of a Protocol Independent Management Agent to Support SNMP and CMIS Queries. *ISINM, San Francisco (Elsiever Science Publisher B.V. (North Holland))*, 1993.

[39] M. Siegle and G. Trausmath. Hierarchical Network Management: A Concept and its Prototype in SNMPv2. *JENC6*, pages 122/1–10, 1995.

[40] N. Soukouti and U. Hollberg. Joint Inter-Domain Management: CORBA, CMIP and SNMP. *IM, San Diego(Chapman & Hall)*, 1997.

[41] William Stallings. SNMP and SNMPv2, The infrastructure for network management. *IEEE Communications Magazine*, pages 37–43, 1998.

[42] IONA Technology. *OrbixWeb Programmer's Guide.* Iona Technology PLC, 1998.

[43] S. Waldbusser. Remote Network Monitoring Management Information Base. *RFC 1271*, 1991.

[44] ITU-T X.701. Information Technology - Open Systems Interconnection - Systems Management Overview. 1992.

[45] X/Open and NMF. Inter-Domain Management: Interaction Specification. *JIDM Working Group*, 1997.

[46] X/Open and NMF. Inter-Domain Management: Specification Translation, Preliminary Specification. *JIDM Working Group*, 1997.

[47] Y. Yemini and G. Goldszmidt. Network Management by delegation. 2nd International Symposium on Integrated Network Management, 1991.