Gestion et localisation de ressources sur un assistant numérique personnel

par

François Lévesque

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, janvier 2000

Le _____ , le jury suivant a accepté ce mémoire dans sa version finale.
date

Président-rapporteur:	M. Marc Frappier Département de mathématiques et d'informatique	_____
Membre:	M. Froduald Kabanza Département de mathématiques et d'informatique	_____
Membre:	M. Michel Barbeau Département de mathématiques et d'informatique	_____
Membre:	M. Richard St-Denis Département de mathématiques et d'informatique	_____

Sommaire

Ce mémoire traite les problèmes liés à la gestion et à la localisation de services sur un assistant numérique personnel (ANP). La localisation de services permet aux applications de découvrir les ressources d'un réseau offrant les services qu'elles ont spécifiés. Par le fait même, la localisation de services rend possible la configuration dynamique des applications afin qu'elles utilisent les ressources du réseau auquel l'ANP est connecté. Des protocoles de localisation de services existent pour accomplir cette tâche, mais ils ne sont pas disponibles pour les ANP.

La recherche qui fait l'objet de ce mémoire a conduit à la réalisation d'un système de gestion et de localisation de services pour un ANP. Ce système permet la spécification, la localisation et l'utilisation de services d'un réseau à partir d'un ANP. Il permet également la configuration dynamique des applications qui utilisent les services.

L'architecture de ce système est distribuée. Les ressources sont accessibles via les composants répartis de CORBA. Cette approche rend les clients indépendants de l'implantation des services offerts par les ressources. Elle permet d'uniformiser des interfaces pour l'accès aux ressources.

Remerciements

Tout d'abord, je veux remercier mes directeurs, messieurs Michel Barbeau et Richard St-Denis, qui m'ont permis de réaliser ma maîtrise. Cette maîtrise m'a permis de m'épanouir et de donner une nouvelle dimension à ma formation et à ma carrière. Je tiens aussi à les remercier pour l'aide financière qu'ils m'ont apportée durant cette maîtrise.

J'aimerais aussi remercier mes acolytes et amis, messieurs Steve Bernier et Denis Thibault, pour leur support moral pendant les temps difficiles ainsi que pour l'aide qu'il m'ont apportée lors de la réalisation de mes travaux.

Enfin, je désire remercier mes parents qui m'ont toujours encouragé dans mes études et tous ceux qui m'ont aidé de près ou de loin dans la poursuite de celles-ci.

Table des matières

Liste des figures

Introduction

Ce mémoire porte sur la gestion des réseaux et des services. La gestion des réseaux consiste à administrer les équipements et à surveiller le trafic du réseau afin que ce dernier demeurent actif et performant. L'étendue des réseaux d'aujourd'hui et la décentralisation des activités nécessitent que les outils de gestion de réseaux soient faciles à installer, à configurer et à utiliser. Quant à la gestion des services, elle consiste à permettre aux utilisateurs des réseaux de requérir facilement à la panoplie de services et de ressources rendus disponibles par les réseaux. De plus, elle doit rendre l'administration de ces services simple aux administrateurs vu le nombre important de ressources qu'un réseau peut posséder.

Les ordinateurs qui se connectent à un réseau sont de différentes tailles (e.g., ordinateur personnel, ordinateur portatif et assistant numérique personnel). Un assistant numérique personnel (ANP) est un ordinateur de poche de plus en plus populaire et son utilisation est restreinte, pour le moment, à des applications de production. L'accès aux ressources d'un réseau à partir d'un ANP n'est pas encore répandu. Par contre, puisque ce type d'appareil se transporte facilement et qu'il est de plus en plus utilisé en entreprise, il a tout avantage à utiliser les services et les ressources des réseaux.

Actuellement, il existe deux moyens pour exploiter les services d'un réseau pour un utilisateur d'un ANP. Prenons, par exemple, l'impression d'un document. Le premier moyen est de spécifier manuellement, à partir de l'application d'impression, l'adresse du

serveur d'imprimante du réseau. Ainsi, l'application pourra envoyer le document à transmettre à ce serveur. Toutefois, il faut que le serveur implante le protocole d'impression utilisé par l'application, sinon le document ne pourra pas être imprimé. Bien sûr, l'application sur l'ANP peut implanter plusieurs protocoles d'impression et l'utilisateur pourrait choisir celui à utiliser à partir d'une fenêtre de configuration. Cependant, en implantant plusieurs protocoles, la taille de l'application augmente alors qu'il est recommandé de garder la taille des applications d'un ANP la plus petite possible vu la faible quantité de mémoire disponible. L'autre moyen d'imprimer un document est de télécharger celui-ci sur un PC du réseau et d'imprimer à partir du PC. Cela implique qu'une application de téléchargement pour ANP (appelée conduit) soit installée sur un PC. On peut donc constater que peu importe le moyen utilisé, l'utilisateur doit effectuer manuellement un certain nombre d'opérations. Dans le cas de l'impression d'un document, il existe une troisième alternative. Il existe une application qui permet d'imprimer à partir d'un ANP. Pour ce faire, il faut relier le ANP, via un câble série, à une imprimante possédant un port série ou bien il faut que l'imprimante soit compatible IrDA. Cette solution nécessite que l'utilisateur se déplace à côté de l'imprimante pour imprimer.

La localisation de services consiste à découvrir des ressources offrant des services qui répondent à des besoins. Actuellement, il existe des protocoles qui donnent la possibilité de localiser des services en fonction de besoins de service spécifiés. Ces protocoles sont utilisés par les applications de façon interactive ou non interactive pour découvrir les services d'un réseau et obtenir un point d'accès aux ressources. Les points d'accès sont généralement des URLs qui pointent sur les ressources en question.

Problème

Selon nos connaissances actuelles, il n'y a pas d'implantation de ces protocoles pour les ANP. La localisation de services n'est donc pas possible sur un ANP. De plus, l'utilisation

des services exige, dans la plupart des cas, que les applications connaissent l'implantation des services ou les protocoles dont elles se servent. Cela implique que les applications doivent implanter une partie des services pour pouvoir s'en servir. L'usage des services d'un réseau à partir d'un ANP est donc peu répandu.

La recherche présentée dans ce mémoire a pour but de solutionner le problème de la localisation des services et de l'accès aux ressources d'un réseau à partir d'un ANP. Elle vise à minimiser autant que possible les opérations manuelles de la part de l'utilisateur pour utiliser les ressources d'un réseau.

Description de notre solution

La solution que nous proposons permet la localisation et l'utilisation des services d'un réseau à partir d'un ordinateur de poche de type ANP. Notre solution repose sur l'architecture de composants répartis CORBA. Notre système permet à un administrateur d'inscrire, modifier et retirer des ressources matérielles ou logicielles disponibles sur le réseau. Les applications spécifient les services dont elles ont besoin, puis les ressources du réseau correspondant à ces services sont découvertes dynamiquement à l'aide de notre système.

Ce genre de solution exige parfois des protocoles spécifiques pour communiquer avec les ressources découvertes. Cela implique qu'un grand nombre de protocoles de communication doivent être implantés sur l'ANP. Notre solution a l'avantage d'éliminer le besoin d'avoir tous ces protocoles sur un ANP puisque nous utilisons des composants répartis. Ainsi, aucun code spécifique n'est nécessaire aux applications pour communiquer avec les services. Les composants répartis offrent une interface uniforme et encapsulent les détails d'implantation.

Nous fournissons aux utilisateurs d'ANP un explorateur de services. Cette application permet de visualiser les ressources disponibles. Les besoins de service sont spécifiés via

une fenêtre de configuration d'une application, ou via l'explorateur de services.

Pour déclencher la localisation des services, il suffit que l'utilisateur démarre l'application de localisation de service en cliquant sur une icône. À ce moment, l'utilisateur doit spécifier le réseau sur lequel porte la localisation. Lorsque le réseau a été précisé, l'application localise les ressources et se termine. Les ressources peuvent maintenant être utilisées par les applications sans aucune autre opération de la part de l'utilisateur. La localisation des services peut aussi être effectuée à partir de l'explorateur de services plutôt que par l'application de localisation.

Pour la réalisation de notre système, nous avons utilisé l'ANP de 3Com. Nous avons opté pour un ordinateur de poche 3Com car il est présent sur le marché depuis quelques années, très populaire et fiable. Le système d'exploitation utilisé par l'ANP est PalmOS 3.0. Nous avons utilisé l'environnement de développement CodeWarrior 5.0 de Metro-Werks pour les applications qui s'exécutent sur l'ANP. Ce choix est basé également sur le fait qu'il est le plus populaire mais aussi parce qu'il est utilisé pour l'implantation du seul ORB actuellement disponible pour PalmOS, Mico [1]. Nous employons Mico dans notre solution.

Pour le système serveur, nous avons choisi d'utiliser le langage Java et le JDK 1.2 de SUN. Nous avons choisi Java parce que c'est un langage portable, donc il nous permet de rendre disponible les ressources des réseaux de n'importe quelle plate-forme pour laquelle il existe une machine virtuelle Java. Le ORB que nous avons utilisé du côté serveur est Visibroker 3.4 de Inprise [2] . Toutefois, d'autres ORB auraient pu également convenir.

Organisation du mémoire

La suite de ce mémoire comporte quatre chapitres. Le premier chapitre donne un aperçu du domaine de la gestion de réseaux et de services. Il présente les divers protocoles de gestion de réseaux et de localisation de services qui existent. Le deuxième chapitre

présente les modèles de composants répartis. Il discute également de l'utilisation du modèle CORBA pour la gestion des réseaux. Le troisième chapitre explique la réalisation de notre système de localisation et d'utilisation de services. Il décrit l'application type ainsi que la conceptualisation, l'analyse, la conception et l'implantation de notre système. Il conclut avec une évaluation de notre logiciel en identifiant ses limites et introduit les améliorations possibles ainsi que les sujets de travaux futurs. Le dernier chapitre est une conclusion de notre réalisation.

Chapitre 1

Gestion de réseaux et localisation de services

1.1 La gestion de réseaux

Avant les années 80, les systèmes de gestion de réseaux étaient presque inexistants ou utilisaient des protocoles propriétaires pour gérer les équipements de réseaux. Les réseaux devenant de plus en plus vastes et complexes, le besoin de la normalisation des protocoles de gestion de réseaux s'est fait ressentir. C'est alors que sont apparus des protocoles et des outils de gestion de réseaux tels que SNMP, CMIP, RMON et bien d'autres.

Les systèmes de gestion de réseaux ont pour but principal de surveiller et de contrôler les réseaux. La gestion des réseaux touche plusieurs aspects, c'est pourquoi elle peut être séparée en cinq domaines fonctionnels comme le fait la communauté OSI (Open Systems Interconnection) [53] : gestion de la configuration, gestion de la performance, gestion des fautes, gestion de la sécurité et gestion comptable. Les dispositifs qui peuvent être gérés dans un réseau sont appelés des noeuds. Les noeuds administrés sur un réseau sont les hôtes (e.g., station de travail, serveur de terminaux, imprimantes réseau), les équipements de raccordements (e.g., commutateurs, répéteurs, ponts, HUB, routeurs, passerelles) et

les médias (e.g., coaxiaux, paire torsadée, fibre optique).

Dans les sous-sections suivantes, nous allons décrire les principaux protocoles de gestion de réseaux utilisés aujourd'hui. Aussi, nous présenterons brièvement les autres normes qui existent en gestion de réseaux.

1.1.1 SNMP

SNMP a été le premier protocole de gestion de réseaux à apparaître dans les années 80. Il fût défini par un groupe de travail de l'IETF (Internet Engineering Task Force). Il se devait être une roue de secours aux difficultés de gestion inter-réseaux en attendant la création de protocoles plus efficaces et permettant d'effectuer des opérations plus complexes. Cependant, le seul protocole à apparaître beaucoup plus tard est celui de l'OSI, et même depuis qu'il est apparu, SNMP est devenu le protocole de gestion de réseaux le plus répandu parce qu'il est simple et efficace. SNMP est un protocole requête/réponse asynchrone basé sur UDP, il est donc utilisé pour administrer les réseaux TCP/IP.

Depuis sa création, il y a eu plusieurs versions du protocole. Heureusement, toutes les versions partagent la même structure et les mêmes composantes de base, ce qui facilite le passage d'une version à une autre.

Dans la prochaine sous-section, le modèle de gestion SNMP est présenté. L'architecture de SNMP et les améliorations faites dans les différentes versions sont présentées dans les sous-sections qui suivent.

Modèle SNMP de gestion de réseaux

Un système de gestion de réseaux basé sur SNMP comprend trois types de composantes : au moins une station d'administration (client), un agent (serveur) sur chaque noeud géré et un protocole réseau pour échanger des informations d'administration. Chaque agent utilise une MIB (Management Information Base) pour représenter les

informations du périphérique qu'il gère. Une application de gestion utilise le protocole SNMP pour agir sur la MIB d'un agent donné. Un agent peut se situer directement sur le périphérique géré, lorsque c'est possible, ou sur un ordinateur à proximité de l'équipement à gérer. Un agent qui gère un équipement autre que celui où il réside est appelé agent proxy.

Normalement, il y a un seul agent par noeud administré et il utilise le port 161 de la machine pour recevoir les requêtes. Lorsque plusieurs équipements gérés sont des composantes d'un même ordinateur, le modèle administratif de SNMP ne suit pas de norme. Il y a au moins quatre alternatives pour l'administration de plusieurs équipements sur une même machine. La première est d'avoir un seul agent sur la machine puis d'intégrer le code du nouvel agent dans l'agent existant à chaque fois qu'un équipement est ajouté. La deuxième alternative est que l'agent sur la machine utilise des routines. De cette façon, à chaque fois qu'un équipement est ajouté, il suffit d'installer une librairie que l'agent utilisera pour gérer cet équipement [58]. Ces deux premières manières nécessitent la compilation de l'agent. La troisième alternative est d'installer un agent par équipement géré et de faire en sorte que les agents utilisent des ports différents [58]. Toutefois, cela implique que la station gestionnaire connaisse les ports pour accéder à chaque agent. Enfin, la dernière alternative est d'utiliser la solution précédente ainsi qu'un agent général qui encapsule les autres agents [58]. De cette façon, le gestionnaire s'adresse à l'agent général par le port 161 et celui-ci envoi les requêtes aux bons agents sur le port correspondant. Les agents retournent les réponses à l'agent général qui les assemble en un seul message pour l'envoyer à la station d'administration. Cette dernière méthode, même si elle est la meilleure, peut causer des problèmes de normalisation si plusieurs protocoles doivent être supportés pour communiquer avec des sous-agents qui sont supportés par différents systèmes d'exploitation. Toutefois, il y a une solution à ce problème puisque l'IETF a proposé la norme AgentX, en janvier 1998, comme protocole de communication entre un agent général et des sous-agents [26].

SNMPv1

SNMP n'est pas seulement un protocole, il s'agit plus d'un cadre de gestion composé d'un langage de définition de données (SMI), d'une base d'information de gestion (MIB) et d'un protocole. Il offre aussi une certaine sécurité et permet d'effectuer des tâches administratives.

Chaque information définie dans une MIB est appelée objet dans le jargon SNMP. SNMP utilise les SMI (Structure of Management Information) pour donner l'ensemble des règles de définition des objets à gérer. Ces règles déterminent comment l'information à gérer est groupée et nommée, les opérations permises, les types de données permis et la syntaxe pour spécifier les MIB[1].

On peut voir la SMI comme étant similaire au schéma d'un système de base de données relationnelle. Une MIB ne conserve pas les valeurs des objets, elle est seulement une représentation pour accéder les objets. Les valeurs des objets sont conservées sur le périphérique géré ou dans la mémoire interne de l'agent et seulement l'agent sait comment les accéder.

Certains objets ont seulement une instance tandis que d'autres ont plusieurs instances. Les objets qui ont le même type d'instance sont organisés dans des tables conceptuelles des MIB. La composition de l'identité d'un objet et de son instance est appelée variable SNMP.

Les objets sont identifiés globalement dans SNMP, de façon unique, à l'aide d'une séquence de nombres entiers non négatifs organisés hiérarchiquement (figure 1). Cette séquence de nombre est appelée identificateur d'objet. Pour faciliter l'utilisation, un nom textuel est associé à chaque élément de la séquence (ou composante) d'un identificateur d'objet. La dernière composante du nom est utilisée par elle-même comme un raccourci

1. Il y a souvent confusion dans l'utilisation du terme MIB. Parfois, il est utilisé pour identifier l'ensemble de l'information de gestion alors que d'autres fois il désigne une partie de l'information de gestion accessible par un agent. Dans le deuxième cas, le terme segment de MIB serait plus approprié. Donc, une MIB est composée de plusieurs segments de MIB, chacun accessible par des agents différents [60].

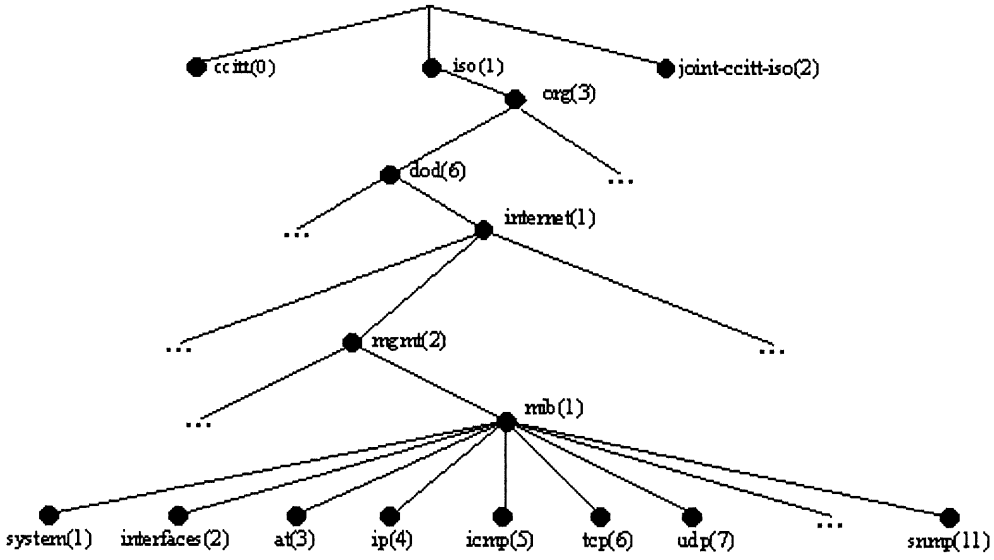

FIG. 1 – *Hiérarchie d'objets pour SNMP*

pour nommer l'objet. Puisque le nom raccourci n'est pas nécessairement unique, tous les noms textuels des objets définis sont, par convention, rendus uniques en utilisant un préfixe différent pour les objets dans chaque nouvelle MIB. SNMP utilise la forme encodée de la valeur numérique plutôt que le nom textuel. Les exemples suivants sont des identificateurs d'objets valides dans SNMP :

{ iso org(3) dod(6) internet(1) } ou 1.3.6.1

{ internet 4 } ou 1.3.6.1.4

{internet 2 mib(1) tcp 4 } ou 1.3.6.1.2.1.6.4

{ tcp 4 } ou 1.3.6.1.2.1.6.4

La syntaxe précise pour spécifier un identificateur d'objet dépend d'où il est utilisé. Le premier nom textuel dans l'identificateur d'objet peut être n'importe quel nom dans la hiérarchie en autant qu'il est unique dans le contexte où il est utilisé.

Les MIB et les objets gérés sont normalisés par l'IETF. Cependant, il se peut que des entreprises veuillent gérer des périphériques dont une MIB n'a pas été développée par l'IETF. Des extensions de MIB sont donc nécessaires pour permettre de gérer les

propriétés spécifiques d'un périphérique ou d'un système propriétaire non incluses dans une MIB standard. Il existe une branche entreprises(1) sous la sous-branche private(4) de la branche internet(1) qui regroupe les MIB pour les produits de différentes entreprises. Les entreprises qui veulent ajouter des MIB doivent obtenir un numéro sous cette branche pour identifier leur propre branche de la hiérarchie. Les numéros pour les branches des entreprises sont assignés par l'IANA (Internet Assigned Number Authority). Toutefois, le numéro affecté à chaque MIB sous la branche d'une entreprise est décidé par l'entreprise.

Une MIB dans SNMP est définie par une collection de définitions de modules qui peuvent être contenues dans un ou plusieurs documents. Un module correspond à un point enregistré dans la hiérarchie d'identificateurs d'objet de SNMP. Dans un module, il est possible d'importer des définitions d'éléments définies dans d'autres modules pour les utiliser. Dans un module, on peut définir des types, des valeurs (instances d'un type) et des macros (règles de grammaire). Les macros sont utilisées pour définir les types d'objets et de « traps ». Les « traps » sont des événements qui sont rapportés au gestionnaire par l'agent SNMP lorsqu'ils se produisent. La SMI et les modules sont définis avec le langage ASN.1.

Un compilateur de MIB est utilisé pour vérifier si les définitions des MIB sont dans la bonne forme. Il peut aussi servir à d'autres usages. Par exemple, générer le code de base en langage C pour un agent ou pour une application de gestion et produire une représentation graphique de la hiérarchie d'une MIB [58].

Le protocole SNMPv1 offre quatre opérations pour l'administration. Un gestionnaire peut retrouver ou modifier l'information de gestion accessible par un agent en utilisant les opérations GET, GETNEXT et SET. L'opération GET permet de récupérer la valeur de l'information de gestion d'un ou de plusieurs objets spécifiés, alors que l'opération GETNEXT permet de retrouver la valeur de l'information de gestion de l'objet suivant le ou les objets spécifiés. L'opération SET permet de modifier la valeur de l'information de gestion de l'objet spécifié. Un agent peut informer un gestionnaire qu'un événement s'est

produit avec l'opération TRAP. Les informations sont échangées sur le réseau à l'aide de message. Un message SNMP est constitué d'information d'administration (numéro de version, besoin en sécurité) et d'un PDU (Protocol Data Unit) qui sont les opérandes ou les résultats d'une opération SNMP. Pour transférer l'information entre deux machines et ce indépendamment de l'architecture des machines, la représentation externe BER (Basic Encoding Rules) est utilisée.

Le principal avantage de SNMP est que sa conception est simple, ce qui le rend facile à implanter sur un réseau. De plus, il ne nécessite pas beaucoup de ressources de la part des noeuds gérés du réseau. Ce qui lui donne sa force et sa simplicité c'est l'utilisation de la MIB pour gérer les objets. Cette simplicité lui permet d'être très extensible.

Le protocole SNMP est simple, toutefois il impose certaines limites à l'administration de l'information. Premièrement, il n'est pas très sécuritaire. La seule sécurité qu'il offre est l'authentification basée sur les chaînes de caractères. Comme l'information échangée n'est pas cryptée, elle peut facilement être captée et modifiée. D'autre part, l'information qu'il gère n'est pas assez détaillée ni bien organisée pour la gestion des réseaux des années 90. Il n'est pas adapté pour la récupération d'informations en masse, ce qui cause beaucoup de trafic sur un réseau. De plus, il manque certaines fonctionnalités telles que la création de nouvelles instances d'un objet de la MIB, l'exécution de commandes de gestion et la communication de gestionnaire à gestionnaire. Finalement, SNMP n'est pas fiable puisqu'il est bâti sur UDP. Ainsi, les messages « traps » ne sont pas acquittés et l'agent ne peut être certain qu'ils ont atteint la station gestionnaire.

SNMPv2

Cette version du protocole procure de nouvelles fonctionnalités [53, 47] et elle est mieux adaptée aux réseaux d'aujourd'hui. Tout d'abord, elle permet une spécification plus détaillée des variables, elle fournit de nouveaux types de données et elle permet d'ajouter ou de supprimer des lignes dans une table conceptuelle. SNMPv2 fournit aussi

deux nouvelles opérations pour l'échange d'information. L'opération GETBULK étend la fonctionnalité de GETNEXT en permettant que plusieurs valeurs soient retournées pour les objets sélectionnés dans la requête. Cette opération permet donc le transfert de données en masse améliorant ainsi l'efficacité et la performance. L'autre opération, INFORM, sert à l'échange d'information entre gestionnaires. Dans SNMPv2, un agent peut également jouer le rôle de gestionnaire. Cela peut permettre de faire une gestion hiérarchique du réseau et d'améliorer la performance. Enfin, la gestion des erreurs a été améliorée et des nouvelles MIB ont également été ajoutées.

Malgré toutes ses nouvelles fonctionnalités, le but premier de SNMPv2 était d'améliorer la sécurité et l'administration. Au niveau de la sécurité, la confidentialité des données, l'authentification et le contrôle de l'accès aux données devaient être définis. Toutefois, le modèle de sécurité que les auteurs du protocole voulaient implanter s'est avéré trop complexe et ils n'ont pas réussi à s'entendre sur un autre modèle [53, 58, 60].

SNMPv2 fut victime de l'énorme succès de son prédécesseur (SNMPv1) et du désaccord de ses auteurs sur certains aspects. Par conséquent, la version 2 de SNMP est théoriquement en vie, mais il existe probablement peu d'agents ou de gestionnaires SNMP qui supportent toutes les extensions de la version 2 [53]. Une version de SNMPv2 sans nouveau concept de sécurité, c'est-à-dire qui implante la sécurité de SNMPv1, est nommée SNMPv2c.

SNMPv3

Cette toute dernière version du protocole SNMP a été définie pour résoudre les problèmes au niveau de la sécurité et de l'administration connues dans la version 2 [47]. Le protocole SNMPv3 est en fait SNMPv2 auquel on a ajouté des fonctionnalités pour la sécurité et pour l'administration.

Les aspects de la sécurité qui sont spécifiés sont l'authentification et la confidentialité ainsi que l'autorisation et le contrôle d'accès aux MIB. Le modèle de sécurité implanté

dans SNMPv3 permet une protection contre quatre menaces potentielles. Ce sont la modification de l'information, la mascarade, la modification de flots de messages et la révélation d'information. L'algorithme de découpage sécuritaire et MD5 sont utilisés pour fournir l'intégrité des données afin d'assurer directement la protection contre la modification des données, de fournir indirectement l'authentification de l'origine des données et de permettre la protection contre la mascarade. La synchronisation sur un indicateur de temps aide à se prémunir contre la modification de flots de messages. Des mécanismes de synchronisation d'horloge sont inclus directement dans le protocole afin de ne pas avoir de dépendance de sources tierces pour la sécurité. Enfin, le chiffrement standard de données (DES) est utilisé pour la protection contre la révélation.

Dans SNMPv3, les noms des entités de l'architecture de SNMP ont été clarifiés et étendus. Ainsi, on retrouve maintenant cinq types d'applications [41], soit des générateurs de commandes, des récepteurs de commandes, des auteurs de notification, des récepteurs de notification et des transmetteurs proxy. Les diverses combinaisons de ces applications peuvent être associées aux entités des versions précédentes :

- une entité SNMP minimal avec un récepteur de commandes ou un auteur de notification correspond à un agent SNMP traditionnel ;
- une entité avec un transmetteur proxy est équivalent à un agent proxy ;
- un générateur de commandes ou un récepteur de notification inclus dans une entité correspond à un gestionnaire dans SNMPv2 ;
- une entité avec un générateur de commandes, un récepteur de notification en plus d'un récepteur de commandes ou un auteur de notification est analogue à une entité qui jouait le rôle d'agent et de gestionnaire dans SNMPv2 ;
- finalement, une entité qui comprend un générateur de commandes ou un récepteur de notification et possiblement d'autres types d'applications pour gérer potentiellement un grand nombre de noeuds correspondent à une station de gestion de réseau traditionnelle.

Cette nouvelle architecture permet la réalisation d'une gestion efficace dans une variété de configurations et d'environnements. Par le fait même, elle permet d'implanter des applications (agents) avec seulement les fonctionnalités requises et ainsi réduire les ressources qu'elles utilisent. SNMPv3 semble être tout à fait adapté pour gérer convenablement les réseaux d'aujourd'hui (de petite taille toutefois). Il ne reste qu'à savoir combien de temps il faudra pour que des implantations complètes soient disponibles.

1.1.2 CMIP

CMIP [2] (Common Management Information Protocol) est le protocole orienté connexion de gestion de réseaux OSI créé par l'ISO (International Standards Organization) [53]. Le but de la gestion OSI est d'apporter une solution complète aux problèmes les plus difficiles de la gestion de réseaux.

Une application de gestion de réseaux OSI est constituée de trois éléments de services d'application, soit la gestion fonctionnelle, les fonctions de gestion des systèmes et les services CMISE (Common Management Information Services Element) [32]. La gestion fonctionnelle comprend cinq domaines fonctionnels SMFA (System Management Functionnal Areas). Les SMFA s'appuient sur les services de 13 fonctions SMF (System Management Functions), définies par OSI, qui peuvent être utilisées par un ou plusieurs SMFA. Les SMF s'appuient sur l'élément CMISE, lequel est une combinaison des protocoles définis par les services CMIS (Common Management Information Services) et le protocole CMIP qui réalise ces services [53]. La figure 2 montre une vue d'ensemble de la gestion répartie de systèmes selon OSI.

Tout comme avec SNMP, un noeud géré dans la gestion OSI comprend aussi un agent et une MIB. Toutefois, une MIB OSI est très différente d'une MIB SNMP. OSI possède son

2. Le terme CMIP est souvent utilisé dans la littérature pour désigner la gestion de réseaux OSI tout comme SNMP peut désigner une architecture de gestion de réseau. Toutefois, CMIP est seulement le protocole de gestion de réseaux utilisé dans la gestion de réseaux OSI.

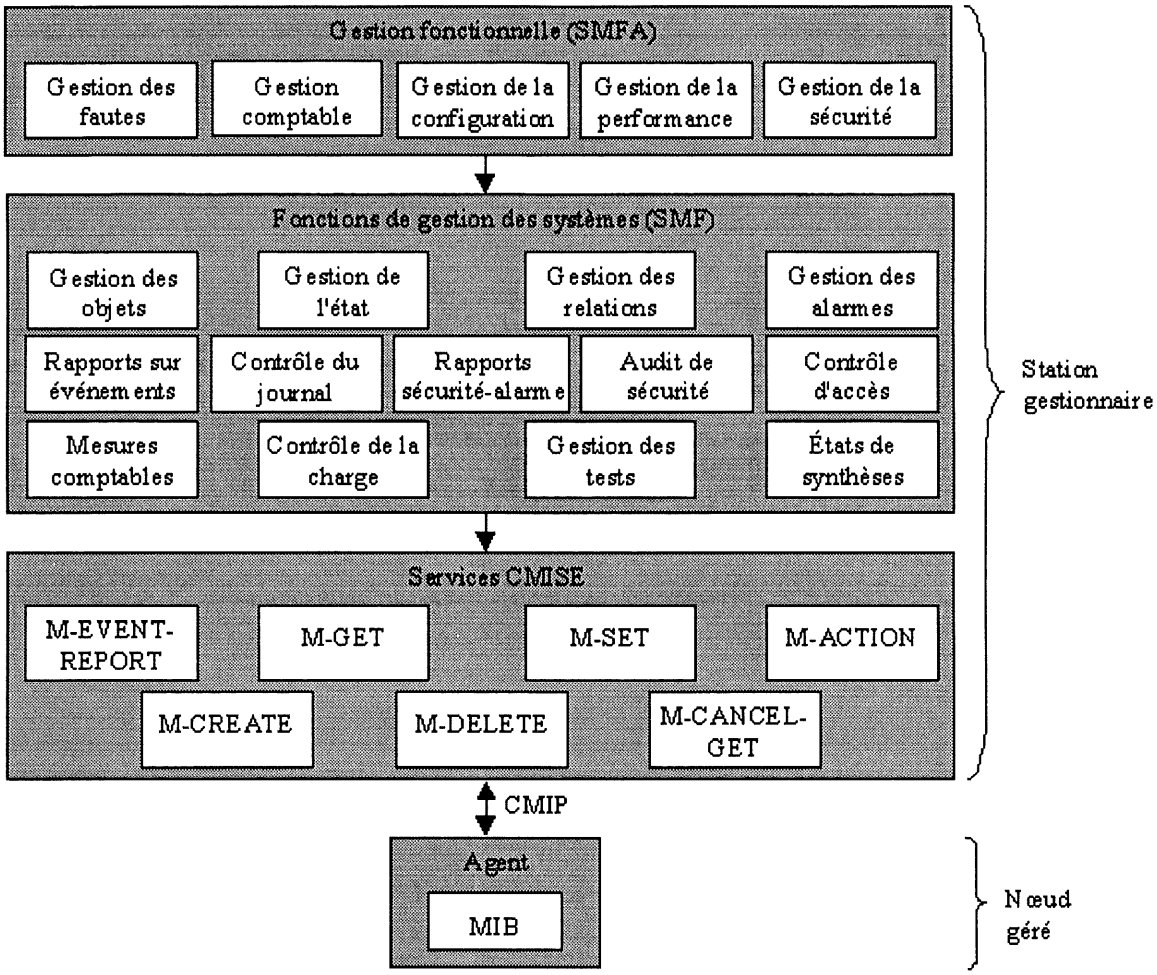

FIG. 2 – *Vue d'ensemble de la gestion répartie de systèmes selon OSI*

propre vocabulaire, CMI (Common Management Information), pour définir les règles de gestion de l'information d'une MIB. GDMO (Guidelines for the Description of Managed Objets), une extension du langage ASN.1, est la notation utilisée pour spécifier les objets de la MIB.

La notation GDMO est beaucoup plus puissante et complexe que celle utilisée par SNMP. Elle permet de définir les objets un peu à la manière de la technologie orientée objet. Pour chaque objet géré on peut définir des attributs, des opérations, des notifications et de l'héritage. Les attributs sont des variables qui représentent des propriétés de la

ressource incarnée par l'objet. Les opérations sont des actions qui peuvent être exécutées sur les attributs d'un objet ou sur l'objet lui-même. Les notifications sont des événements émis par un objet. Les types de notification définis sont les alarmes et les violations, les changements de la valeur d'un attribut, la création et la suppression d'un objet ainsi que les changements de l'état d'un objet. Enfin, l'héritage permet à une classe d'objets d'hériter de toutes les caractéristiques d'une autre classe d'objets. Tout comme dans la technique orientée objet, une instance d'une classe peut être créée dynamiquement pour obtenir un nouvel objet.

Le reste du modèle réparti de gestion OSI est semblable à celui de SNMP. Chaque noeud géré contient un agent sur lequel un gestionnaire agit pour retrouver ou modifier l'information du noeud via la MIB. La MIB est également représentée par une hiérarchie d'identificateurs d'objet. La communication de gestionnaire à gestionnaire est aussi supportée.

CMISE offre sept services, définis par CMIS, qu'un gestionnaire peut invoquer par des opérations du protocole CMIP. Ces services sont [32] :

- M-GET qui permet à un gestionnaire de retrouver les valeurs des attributs d'un ou de plusieurs objets gérés ;
- M-CANCEL-GET qui permet à un gestionnaire d'annuler une requête M-GET précédente ;
- M-SET qui permet au gestionnaire de modifier les valeurs des attributs d'un ou de plusieurs objets gérés ;
- M-ACTION qui permet à un gestionnaire l'exécution d'une opération sur un ou plusieurs objets gérés ;
- M-CREATE qui permet à un gestionnaire de créer un objet géré ;
- M-DELETE qui permet à un gestionnaire de supprimer un ou plusieurs objets gérés ;

- M-EVENT-REPORT qui permet à un agent d'envoyer spontanément des notifications au gestionnaire.

CMIS offre aussi beaucoup de flexibilité pour les requêtes. Les objets impliqués dans une requête peuvent être identifiés selon une portée. La portée marque le noeud de la hiérarchie de la MIB à partir duquel commence la recherche. Des critères de sélection sur les objets impliqués peuvent aussi être spécifiés dans une requête pour filtrer les objets. Lorsque plusieurs objets sont impliqués dans une requête de service, le gestionnaire peut indiquer le type de synchronisation (meilleur effort ou atomique) pour la réalisation du service.

CMIS/CMIP permet de gérer un réseau beaucoup plus efficacement que SNMP. Son plus grand avantage est que les objets gérés ne font pas que transporter de l'information du noeud à la station de gestion et vice-versa, mais ils peuvent aussi être utilisés pour effectuer des tâches. Ainsi, CMIP facilite la tâche à l'administrateur de réseau car il peut effectuer des tâches pour lui. D'autre part, la sécurité est intégrée dans le protocole CMIP.

Par contre, malgré le fait que le système de gestion de réseaux OSI est très puissant et apporte une solution complète aux problèmes de gestion de réseaux, il n'est pas très répandu. À cause de sa complexité, il utilise beaucoup de ressources sur les noeuds du réseau. Par conséquent, très peu de réseaux sont capables de supporter une implantation complète sans faire des modifications massives au réseau. Seuls les réseaux de télécommunications implantent actuellement l'architecture de gestion de réseaux OSI.

Il existe deux autres versions du protocole CMIP pour la gestion de réseaux [53]. CMOT (CMIP over TCP/IP) est une version réduite de CMIP pour les réseaux TCP/IP. CMOL (CMIP over LLC) est un petit CMIP conçu pour être utilisé directement au-dessus de la couche de liaison logique IEEE 802.2.

1.1.3 RMON

RMON (Remote Network MONitoring) est une extension à SNMP qui permet de surveiller un réseau afin de détecter les problèmes et de les résoudre. RMON peut agir sur l'information qu'il recueille sans impliquer la station de gestion. Les deux versions de RMON existantes sont décrites dans cette section.

RMON

RMON est un segment spécialisé d'une MIB SNMP qui étend considérablement les possibilités en gestion de réseaux. Un agent RMON, également appelé moniteur, est un agent intelligent qui implante une MIB RMON. Un agent RMON permet de surveiller minutieusement le réseau et de détecter rapidement les problèmes et aussi de les analyser afin de les résoudre. Il peut filtrer l'information qu'il recueille et agir sur elle sans avoir à impliquer directement la station de gestion pour chaque action, ni à inonder le réseau de transferts de données volumineux. L'information qu'on peut obtenir avec RMON concerne seulement les deux premières couches du modèle OSI [53]. La norme RMON est définie par l'IETF.

Plusieurs applications RMON peuvent utiliser un moniteur en même temps. L'administrateur de réseau peut configurer l'agent RMON pour offrir des vues différentes à différents membres de l'équipe de gestion. Un agent RMON collecte de l'information sur des segments de réseau. La surveillance continuelle permet de fournir l'information avant que les problèmes surviennent, laissant ainsi les gestionnaires distants prendre une approche proactive de gestion.

RMON définit des extensions à la MIB MIB-II pour informer un moniteur sur les données qu'il doit recueillir. Ces extensions sont des groupes d'objets où les objets représentent des commandes. Le moniteur exécute une commande lorsque le gestionnaire invoque l'opération SNMP SET sur un de ces objets. Chacun des groupes de la norme

RMON fournit un ensemble de variables de contrôle permettant à une station de gestion de diriger à distance l'opération d'un moniteur. Ces variables peuvent être considérées comme des tables d'état disant au moniteur ce qu'il faut collecter et comment il doit traiter les événements qu'il génère. Il faut se rappeler, qu'avec SNMP, il n'est pas possible de créer des objets dans une MIB ou de spécifier des actions en relation avec des objets. Avec RMON, les applications peuvent demander au moniteur de créer dynamiquement un ensemble de variables, de débuter une nouvelle activité de surveillance et même de créer des objets qui surveillent d'autres objets pas nécessairement reliés à RMON.

La MIB RMON comprend neuf groupes pour gérer les réseaux Ethernet et un groupe pour les réseaux Token Ring. Les groupes de RMON sont [70] :

- Statistics qui sert à comptabiliser des statistiques Ethernet (multicasts, fragments, collisions) ;
- History qui permet au gestionnaire de fixer un temporisateur pour l'enregistrement d'échantillons de statistiques Ethernet ;
- Alarm qui permet au gestionnaire de fixer un seuil sur des variables afin de déclencher une alarme lorsqu'elles prennent des valeurs sous ce seuil ;
- Host qui permet d'obtenir des statistiques pour un périphérique spécifique d'un LAN ;
- HostTopN qui permet de lister les meilleurs hôtes selon des paramètres prédéterminés par le gestionnaire ;
- Matrix qui permet de faire le suivi d'un nombre limité d'information du trafic entre deux usagers ;
- Filter qui permet à la station de gestion de sélectionner les paquets à observer sur une interface sous-réseau ;
- Capture qui permet la copie de paquets ou de portion de paquets dans une mémoire tampon à partir de canaux filtres ;

- Event qui définit des événements pouvant être le résultat de paramètres fixés dans d'autres régions de la MIB ;
- TokenRing qui permet de comptabiliser des statistiques propres aux réseaux Token Ring et de conserver des échantillons de statistiques [22].

Tous les groupes sont optionnels. Pour être conforme à RMON, un vendeur n'a besoin de supporter qu'un des neuf groupes.

En résumé, RMON est un protocole qui utilise des variables pour transmettre des instructions et des paramètres ainsi que pour contrôler l'information entre une station gestionnaire et un agent. Il permet de faire une analyse spécifique et profonde du trafic de données. RMON permet aussi de réduire la surcharge de gestion en limitant les intervalles d'interrogation et de transmission puisqu'il effectue la surveillance à la place de la station de gestion.

Puisque RMON est un sous-ensemble de SNMP, la plupart des vendeurs qui contrôlent le monde des réseaux l'ont incorporé à leurs produits. Par contre, RMON est une solution qui a des faiblesses dans des aspects importants [12]. D'abord, la quantité d'information qui est fournie n'est pas assez complète pour les administrateurs de réseaux qui doivent faire la résolution de problèmes complexes à distance. Ensuite, les mécanismes employés par une station de gestion centrale pour récupérer les données sont très lents et inefficaces en utilisation de largeur de bande. Enfin, un grand nombre de manufacturiers d'équipements de réseau choisissent d'implanter seulement une partie de la norme pour ne pas être obligé d'ajouter des coûts supplémentaires aux équipements qui sont dit conforme à RMON. Pour bien supporter RMON, un équipement nécessite un processeur RISC dédié.

RMON-2

RMON-2 ajoute neuf nouveaux groupes au-dessus de RMON. Il collecte des informations sur le fonctionnement de la couche réseau et de la couche application [22]. Ainsi, il

permet de construire des vues de bout en bout de la totalité du réseau, alors que RMON ne traite que de simples segments de réseau. Par le fait même, un agent RMON-2 est capable de fournir plus d'information lorsque les problèmes inter réseaux sont complexes. Il donne une meilleure compréhension du trafic entre deux segments de réseau en fournissant de l'information sur le trafic entre les sous-réseaux, la distribution du trafic des protocoles et les activités des applications.

Les nouveaux groupes de RMON-2 sont [22] :

- ProtocolDirectory qui liste les protocoles que le moniteur est capable de surveiller ;
- ProtocolDistribution qui comptabilise des statistiques du trafic pour chaque protocole et permet de fournir des informations sur l'utilisation des protocoles ;
- AddressMapping qui associe les adresses réseaux aux adresses physiques ;
- Network-LayerHost qui comptabilise des statistiques du trafic qui va ou qui provient de chaque nouvel hôte découvert ;
- Network-LayerMatrix qui comptabilise des statistiques du trafic sur les conversations entre les paires d'hôtes découverts ;
- Application-LayerHost qui comptabilise des statistiques du trafic qui va ou qui provient de chaque hôte pour chaque protocole ;
- Application-LayerMatrix qui comptabilise des statistiques du trafic sur les conversations entre les paires d'hôtes pour chaque protocole ;
- UserHistoryCollection qui permet au gestionnaire de fixer un temporisateur pour l'enregistrement d'échantillons de statistiques qu'il choisit lui-même ;
- ProbeConfiguration qui procure un moyen standard de configurer à distance les paramètres à sonder.

RMON-2 est économiquement plus efficace que RMON car il n'est plus nécessaire de placer des sondes sur chaque segment du réseau pour récolter l'information. Les sondes sont intégrés dans les routeurs et les hubs, d'où elles peuvent recueillir les informations

du trafic de la dorsale du réseau.

1.1.4 DMI

Les normes de gestion de réseaux présentées jusqu'à maintenant permettent d'administrer le réseau en général et les périphériques de réseau. Toutefois, elles ne sont pas vraiment adaptées pour faire la gestion des composantes d'une station ou d'un ordinateur. La norme DMI (Desktop Management Interface) [23] a été créée par DMTF (Desktop Management Task Force) pour répondre à ce type de gestion. La norme DMI est supportée par les manufacturiers de matériel informatique depuis 1996.

La norme DMI est une interface multi plates-formes indépendante du protocole et du système qui aide à gérer le matériel, les systèmes d'exploitation, les applications, le stockage et les périphériques d'un ordinateur. Elle est constituée d'une base de données MIF (Management Interface Format) et d'un agent DMI. La base de données MIF contient toutes les informations qui peuvent être gérées dans le système. L'agent DMI représente un ordinateur et ses ressources, et il est activé seulement sur demande. Lorsqu'il est activé, il charge le code nécessaire pour gérer une unité et collecte l'information pendant que les autres applications continuent de fonctionner.

Chaque manufacturier fournit un fichier MIF sur leurs produits. Ce fichier, qui est un simple fichier texte ASCII, contient les informations pertinentes qui peuvent être gérées pour le produit. Les attributs gérés sont décrits dans le langage MIF qui possède une grammaire définie. Lorsqu'un produit est installé dans un système, l'information de son fichier MIF est ajoutée dans la base de données MIF et il est alors disponible à l'agent DMI, et par le fait même aux applications de gestion.

La valeur de chaque attribut de la MIF peut être retrouvée dans la base de données MIF ou être fournie par un programme qui est exécuté lorsqu'elle est demandée. Un attribut peut avoir une simple valeur ou être contenu dans une table et parcouru en utilisant une clé indexée.

Avec la norme DMI 2.0, des événements peuvent être définis dans le fichier MIF d'un dispositif. On peut aussi spécifier les types d'événement pour lesquels on veut recevoir des notifications, ainsi que fixer des seuils de sévérité.

Un agent DMI possède deux interfaces : une interface aux composants et une interface de gestion. L'interface aux composants permet aux constructeurs de composants d'enregistrer leurs dispositifs et leurs interfaces dans DMI. L'interface de gestion fournit une API pour permettre aux applications de gestion de retrouver les attributs des composants, de modifier les attributs des composants ou d'obtenir la liste des composants gérés par DMI.

Finalement, il est possible d'utiliser la norme DMI avec le protocole SNMP ou d'autres systèmes de gestion répartis pour gérer à distance les composants d'un ordinateur ou d'une station.

1.1.5 MIB mobile

Pour les réseaux qui comprennent des équipements mobiles, les besoins de gestion de ces équipements ne sont pas tout à fait les mêmes que ceux des équipements fixes. Par exemple, il peut être important pour l'administrateur de réseau de savoir où se trouve un ordinateur portatif, l'état de l'énergie de sa pile ou s'il est branché au réseau avec une connexion avec fil ou sans fil. Il n'y avait pas de norme de gestion pour les équipements mobiles jusqu'à ce que MMTF (Mobile Management Task Force) propose quatre MIB de gestion de la mobilité.

Ces quatre MIB, les normes MIB System, MIB Adapter, MIB Link et MIB Extended System sont conçues pour être une extension à SNMP [10]. Elles définissent des objets qui servent à administrer différents aspects pour le support de la mobilité dans les réseaux. Chaque MIB a une fonctionnalité spécifique afin de rendre la gestion de réseaux plus personnalisée aux besoins des concepteurs d'applications de gestion d'environnement mobile de réseaux.

Les informations fournies par les MIB sont accessibles lorsque l'utilisateur de l'ordinateur portatif communique sur le réseau via une connexion avec ou sans fil.

1.2 La gestion et la localisation de services

Un service est une action qui peut être accomplie par une ressource d'un réseau. Une ressource est soit matérielle, par exemple un périphérique, ou soit logicielle tel qu'un serveur de fichiers ou un serveur d'imprimantes. La gestion des services est effectuée par les administrateurs des réseaux via des applications d'administration de services. Les principales opérations de gestion communes à tous les services sont la définition des ressources à rendre disponibles aux utilisateurs des réseaux ainsi que les autorisations pour chacune d'elles. L'activation et la désactivation de l'accès à des ressources ou des services sont également des opérations de gestion de services. Le reste des opérations consiste à personnaliser la configuration propre à chaque service.

La localisation de services consiste à découvrir les ressources d'un réseau qui offrent des services spécifiques. Les protocoles de localisation de services sont utilisés entre autres par les applications résidant sur des ordinateurs mobiles. Puisque les ordinateurs mobiles sont amenés à se déplacer et à se brancher sur différents réseaux, ces protocoles permettent aux applications d'obtenir des points de référence sur des ressources qui fournissent les services dont elles ont besoin pour fonctionner.

Les sous-sections qui suivent décrivent différents protocoles qui permettent de localiser les services qui existent sur un réseau.

1.2.1 DHCP

DHCP (Dynamic Host Configuration Protocol) [57] est un protocole qui permet à un nouvel ordinateur qui se connecte à un réseau de requérir et d'obtenir une configuration pour lui permettre de fonctionner sur le réseau. Ce protocole, développé par un

groupe de travail de l'IETF, est entre autres utilisé pour qu'un ordinateur puisse obtenir dynamiquement une adresse IP lorsqu'il est initialement branché à un réseau.

DHCP est conçu autour du modèle d'opération client-serveur. Le système client effectue sa requête pour obtenir des informations de configuration et le serveur lui soumet ces informations selon une séquence de transmissions à plusieurs étapes. Lorsqu'un serveur DHCP fournit des paramètres de configuration à un client, ce dernier a le choix de les accepter ou de les refuser. S'il les accepte, ces paramètres lui sont attribués pour une période de temps (qui peut être infinie) et le client a la possibilité de renouveler son droit d'utilisation de ces paramètres. En plus de l'adresse IP, les informations normalement distribuées aux clients DHCP sont le masque de sous-réseau, l'adresse du routeur par défaut et les adresses des serveurs de noms locaux ainsi que le nom des domaines qu'ils servent.

Pour obtenir les données de configuration, le client DHCP doit d'abord découvrir un serveur. Il effectue cette opération en diffusant un message à des récipiendaires locaux au port UDP 67. Un récipiendaire peut satisfaire la requête s'il est un serveur DHCP ou la transmettre à un serveur s'il est un relais DHCP, sinon la requête est perdue. Le serveur répond au client en lui envoyant directement une sélection de paramètres de configuration s'il se trouve sur le même sous-réseau que lui, sinon il envoie les paramètres à l'agent de relais qui lui a transmis la requête du client. Le client accepte seulement une offre d'un serveur parmi les réponses reçues et rejette les offres des autres serveurs. S'il accepte la configuration, le client demande au serveur d'enregistrer la configuration comme lui étant désignée. Finalement, le serveur envoi un accusé de réception au client pour l'assurer que l'opération d'attribution a réussie.

Un client peut également spécifier les paramètres de configuration qu'il désire lors du renouvellement de son adresse IP plutôt que de choisir parmi un ensemble d'offres. Si le serveur accepte la requête du client, il lui envoie un accusé de réception positif. Le client peut libérer son adresse IP (et sa configuration) à n'importe quel moment.

En résumé, DHCP permet à un ordinateur de se configurer dynamiquement pour un réseau. Cependant, DHCP ne permet pas d'accepter seulement une partie des paramètres offerts, c'est l'ensemble de la configuration qui doit être accepté ou rien du tout. Par exemple, un client ne pourrait pas accepter qu'une adresse IP lui soit attribuée et choisir un serveur de noms autre que ceux qui font partie de la configuration. De plus, la configuration est restreinte à un certain nombre de ressources (ou paramètres) et ne permet pas de spécifier les ressources à découvrir.

1.2.2 SAP

Le protocole SAP (Service Advertising Protocol) est employé par les nœuds d'un réseau qui fournissent des services pour annoncer leur service et leur adresse [9]. Ce sont les serveurs de services (e.g., les serveurs de fichiers, les serveurs d'imprimantes et les serveurs d'applications) qui utilisent ce protocole. Ce dernier leur permet d'annoncer leur présence lorsqu'ils sont démarrés et leur absence au moment où leur exécution est interrompue. Donc, le protocole permet d'ajouter ou d'enlever des services dynamiquement. La diffusion de messages est utilisée par les serveurs pour annoncer leur adresse.

SAP permet de découvrir les services disponibles, mais il ne permet d'obtenir des informations sur ces services. Il ne peut donc pas être utilisé pour découvrir des services qui possèdent des propriétés spécifiques. Dans un autre ordre d'idée, puisque ce protocole est utilisé par les serveurs de services plutôt que par les applications d'administration de services, aucune authentification ne peut être faite sur les utilisateurs qui utilisent les services, à moins que chaque serveur implante un mécanisme d'authentification. Ce protocole ne peut donc pas être employé dans tous les systèmes.

1.2.3 LDAP

Lightweight Directory Access Protocol (LDAP) est un protocole, conçu par L'IETF, qui sert à gérer et à retrouver des données qui sont regroupées dans des répertoires [49]. Un répertoire est une base de données spécialisée. LDAP est semblable et compatible avec le service de répertoire X.500 de l'ISO.

Le protocole de communication LDAP définit le transport et le format des messages utilisés par les clients pour accéder aux données d'un répertoire comme X.500. Le serveur LDAP qui reçoit les requêtes peut agir comme une passerelle entre les clients et le serveur X.500 ou un autre serveur qui implante le répertoire, ou bien il peut lui-même implanter le répertoire. Le protocole LDAP est supporté par plusieurs vendeurs de systèmes de différentes plates-formes et il est incorporé dans plusieurs produits. Actuellement, la version 2 du protocole est celle qui est la plus répandue, mais la version 3 de LDAP est définie et des implantations existent sur le marché. La version 3 n'ajoute que quelques extensions à la version 2. La description du protocole LDAP dans cette section se rapporte à la version 3.

Une API en langage C est définie pour être utilisée par les applications client afin d'envoyer des requêtes au serveur ainsi que pour recevoir des réponses. Bien que cette API ne soit pas normalisée, elle a été adoptée de facto par l'industrie. Cette API est implantée au-dessus du protocole TCP/IP. Un format de URL a aussi été normalisée pour envoyer des requêtes LDAP via le protocole HTTP. Pour envoyer une requête de recherche d'information, le client doit d'abord se connecter au serveur LDAP. Le port standard destiné au protocole LDAP est le port TCP 369.

Dans le protocole LDAP, chaque groupe d'information représentant un objet dans un répertoire est une entrée. Un objet décrit dans une entrée de répertoire représente habituellement une entité distincte (e.g., une personne, une imprimante, un serveur). Chaque entrée a un nom distingué (DN : Distinguished name) qui l'identifie de façon unique. Un DN est composé d'une séquence de noms distingués relatifs (RDN : Relative distinguished

name). Les entrées sont arrangées dans une structure hiérarchique, semblable à un arbre, basée sur leur nom distingué. Cet arbre d'entrées de répertoire est appelé arbre d'information de répertoire (DIT : Directory Information Tree). Chaque entrée contient un ou plusieurs attributs qui la décrit. Chaque attribut a un type ainsi qu'une ou plusieurs valeurs.

Les classes d'objets qu'un serveur de répertoire peut emmagasiner sont décrites dans un schéma. Le schéma défini aussi quels attributs elles doivent contenir, quels attributs sont optionnels et la syntaxe de chaque attribut. Chaque serveur LDAP possède un schéma de répertoire. Un schéma peut contenir la définition d'un DIT complet ou de seulement une partie d'un DIT. Si un serveur n'implante pas un DIT complet, il peut posséder une entrée qui réfère à un autre serveur pour certaines classes d'objets. Une telle entrée de référence contient l'URL d'un serveur. Les entrées de références permettent donc au DIT d'être réparti et même dupliqué sur plusieurs serveurs.

Le protocole LDAP permet plusieurs opérations pour gérer et rechercher des informations dans un répertoire. Les opérations qui sont possibles par un serveur LDAP sont la recherche d'entrées selon des critères spécifiés par le client, l'ajout d'une entrée, la destruction d'une entrée, la modification d'une entrée, la modification du nom distingué ou d'un nom distingué relatif d'une entrée et la comparaison d'une entrée.

Lorsqu'un client veut effectuer une requête de recherche, il peut spécifier dans quelle partie du répertoire la recherche doit être faite et quelles informations doivent être retournées. Cela implique que le client doit connaître le schéma de répertoire utilisé. Un certain nombre de paramètres doivent être spécifiés dans une requête de recherche. Tout d'abord, il faut indiquer le DN qui définit le point de départ de la recherche dans le DIT. Ensuite, il faut spécifier la portée de la recherche, c'est-à-dire la profondeur que la recherche peut atteindre dans le DIT à partir du point de départ. La profondeur peut être de demeurer au niveau du point de départ, d'aller au plus un niveau de plus que le point de départ ou bien l'arbre au complet. Après la portée, il faut spécifier le filtre de

recherche. Des conditions booléennes ainsi que des opérateurs relationnels sont utilisés dans le filtre de recherche pour spécifier les critères de sélection. Vient ensuite la liste d'attributs que le serveur doit retourner pour les entrées qui respectent les critères de recherche. Finalement, comme le protocole LDAP supporte l'utilisation d'alias, il faut spécifier si les alias doivent être résolus ou non, ainsi que la limite quant au temps de recherche ou à la taille des résultats.

Finalement, même si le but de ce protocole n'est pas réellement de permettre la gestion et la localisation de services, il peut quand même être utilisé pour retrouver des services selon des critères spécifiques. Si l'entrée pour un service particulier contient l'adresse du serveur pour ce service il sera alors possible d'utiliser le service. Toutefois, le client sera dépendant des particularités du service (e.g., protocole). De plus, la gestion hiérarchique des données réalisée par ce protocole n'est pas nécessaire pour la gestion de services, ce qui cause un problème de flexibilité inutile. Un autre inconvénient est qu'il faut établir une certaine configuration afin de connaître l'adresse du répertoire de départ ainsi que le schéma de répertoire employé pour pouvoir effectuer des recherches. La raison pour laquelle le protocole LDAP est quand même utilisé pour la gestion de services, même s'il n'est pas le plus approprié, est qu'il est un protocole très répandu. Par conséquent, les gens qui utilisent LDAP ne veulent pas s'incommoder d'un protocole de localisation de services puisqu'ils ont déjà un protocole qui peut faire le travail.

1.2.4 SLP

SLP (Service Location Protocol) est le protocole développé par l'IETF dans le but de découvrir les ressources et les services d'un réseau. Il peut être utilisé de manière interactive ou non interactive dans les applications. Les services et les ressources qui sont disponibles peuvent être enregistrés et dé-enregistrés dynamiquement via le protocole. De plus, SLP ne nécessite pas une administration centralisée. Il utilise la couche transport UDP et la couche TCP, lorsque c'est nécessaire, pour la transmission de données en

masse [72].

Ce protocole, qui effectue une gestion des ressources avec une approche non hiérarchique, comprend quatre entités différentes : des agents utilisateurs, des agents de service, des mandataires (proxies) et des agents de répertoire [55]. Un agent utilisateur représente une application et il est capable de comprendre les ressources et les services nécessaires à l'application. Un agent de service représente un service du réseau et rend celui-ci disponible aux agents utilisateurs. Quant au mandataire, il effectue les mêmes fonctions qu'un agent utilisateur ou qu'un agent de service pour les applications ou services qui ne sont pas écrits pour utiliser les fonctions de SLP. Enfin, les agents de répertoire gèrent les services qui sont organisés en répertoire. Un agent de répertoire peut gérer les services de plusieurs agents de service. Il utilise le port 427 pour recevoir les requêtes.

Le protocole SLP supporte la notion de portée, ce qui permet aux ressources de réseaux d'être regroupées en domaines administratifs. Puisque les agents utilisateurs sont personnalisés avec le nom de leur portée, chaque agent inclut sa portée dans ses requêtes de service, ce qui permet d'accéder aux services qui se retrouvent dans cette portée.

Dans SLP, les services sont décrits en donnant des valeurs à des attributs qui sont possibles pour chaque service. Pour découvrir un service et utiliser une ressource de ce dernier, un agent utilisateur doit d'abord s'adresser à un agent de répertoire. Afin qu'un agent de répertoire puisse être découvert, celui-ci peut envoyer un message multicast qui sera capté par les agents utilisateurs et les agents de service. Les autres moyens de découvrir un agent de répertoire sont soit par un fichier de configuration ou soit par une option du protocole DHCP. Lorsqu'un agent utilisateur connaît l'adresse d'un agent de répertoire, il peut lui envoyer des requêtes de service.

L'agent de répertoire doit connaître les services et les ressources disponibles pour être capable de répondre aux requêtes des agents utilisateurs. C'est le rôle des agents de service de s'inscrire auprès de l'agent de répertoire pour que le service qu'il représente soit publié

et accessible. Lors de l'inscription d'un service, tous les mots clés et les paires attribut-valeur qui décrivent le service sont spécifiées. De plus, un temps d'échéance est mentionné après lequel l'information pour le service sera détruite par l'agent de répertoire. L'information d'un service peut aussi être retirée par un dé-enregistrement explicite auprès de l'agent de répertoire. Enfin, les agents de service peuvent modifier les valeurs des attributs périodiquement.

Pour acquérir un point d'accès (handle) de service, un agent utilisateur envoie une requête sous forme de chaîne de caractères à l'agent de répertoire. La requête doit contenir le type de service ainsi que les mots clés et les valeurs des attributs désirés pour la ressource appropriée aux besoins de l'application. Par la suite, l'agent utilisateur reçoit une réponse de service qui contient un URL pointant sur l'agent de service et d'autres informations concernant le service.

Pour spécifier les mots clés et les attributs désirés dans une requête de service, plusieurs opérateurs logiques et de comparaison peuvent être utilisés. Ces opérateurs sont : AND, OR, =, >, <, >= et <=.

Finalement, il existe aussi des requêtes pour découvrir les informations pour un type de service ainsi que pour découvrir tous les services disponibles. De manière générale, une requête de service à la forme suivante [72] :

$$<srvtype>[.<na>]/[<scope>]/[<where>]/$$

où

$srvtype$ = type de service
na = autorité de nommage
$scope$ = portée administrative
$where$ = condition(s) désignant la (les) ressource(s) à sélectionner

Les conditions de sélection des ressources sont spécifiées entre des parenthèses et elles sont liées par un ou plusieurs opérateurs logiques selon une convention préfixée. Une

réponse à une requête de service à la forme suivante :

service :<*srvtype*> ://<*addr-spec*>

où

srvtype = type de service

addr-spec = URL de la ressource

Par exemple, la requête suivante peut être utilisée pour obtenir un point d'accès à une imprimante qui supporte le protocole lpr, qui est située au 12ième étage et qui imprime 12 pages par minute :

```
lpr//(& (PAGES PER MINUTE==12) (UNRESTRICTED_ACCESS)
    (LOCATION==12th FLOOR))/
```

Un exemple de réponse à cette requête pourrait être l'URL suivant :

```
service:lpr://igore.wco.ftp.com:515/draft
```

Finalement, toutes les requêtes sont véhiculées à l'intérieur d'un message SLP. Un message SLP possède un en-tête qui indique entre autres la version du protocole et le type de requête contenu dans le message.

1.2.5 RDP

Le protocole RDP (Resource Discovery Protocol) a pour but de fournir aux clients mobiles un moyen allégé de découvrir les ressources d'un réseau ou d'Internet [56]. Un client s'adresse à un serveur RDP pour requérir les ressources d'un réseau. L'obtention de l'adresse d'un serveur RDP ne fait pas partie du protocole, donc le client peut l'obtenir soit statiquement via un fichier de configuration ou dynamiquement via DHCP.

Le protocole RDP repose sur le protocole UDP et suit un modèle client-serveur requête-réponse. Le but de ce protocole est de découvrir des ressources et de les utiliser à partir d'un ordinateur mobile sans que l'usager n'ait besoin d'intervenir. Un client

formule une liste de URN (Uniform Resource Name) pour demander des ressources et obtient en réponse un ou plusieurs URL (Uniform Resource Locator) pour chacun d'eux. Il peut ensuite utiliser le ou les URL pour accéder aux ressources du réseau. RDP supporte aussi l'enregistrement ou le dé-enregistrement dynamique de ressources et permet au serveur de gérer les ressources automatiquement.

Le choix d'utiliser une forme de chaîne de caractères pour les requêtes/réponses fut basé sur la flexibilité, l'extensibilité et la lisibilité que procure cette approche. De plus, cette approche est plus facile à répandre et à utiliser du côté des administrateurs et des concepteurs qu'une forme numérique quelconque. En effet, les chaînes de caractères sont plus conviviales pour spécifier des attributs ou des mots clés pour la sélection parmi un nombre de ressources de même type.

Un URN qu'un client construit pour localiser une ressource à la forme suivante :

$$<service>:/[rp]/[na]/<scheme>/<key1>/<key2>/...$$

où

service = n2l ou n2c
rp = chemin de résolution
na = autorité de nommage
scheme = type d'URL (ressource)
keyN = mots clés décrivant l'URL

Le service indique le type de résolution. Lorsque n2l est spécifié, le serveur retourne le premier URL qui satisfait l'URN, tandis qu'avec n2c le serveur retourne une liste de URL. Le chemin de résolution est optionnel. Le client peut l'utiliser pour spécifier à quel serveur RDP la requête doit être envoyée. L'autorité de nommage est aussi optionnelle. Elle sert à spécifier l'organisation qui est autorisée à résoudre la requête et par le fait même le dictionnaire qui est utilisé pour définir le type d'URL. IANA, l'organisme qui assigne les différentes numérotations pour Internet, a déjà spécifié quelques types de ressources

connues telles que printer, mailto, http et nfs. Enfin, les mots clés sont utilisés pour la recherche des ressources dans la base de données. Une recherche est satisfaite lorsque le type d'URL et tous les mots clés correspondent.

Voici quelques exemples d'URN valides :

`n2l://ibm/nfs/rdp/src`

`n2c://ibm/http/research/homepage`

`n2l:///printer/local/postscript`

Des exemples d'URL valides en réponse à ces requêtes sont :

`nfs://slag.watson.ibm.com/src/rdp`

`http://www.research.ibm.com/`

`printer://dukprinz.watson.ibm.com/j1j25ps`

De façon similaire, un client peut enregistrer et dé-enregistrer des ressources en utilisant les requêtes reg et dereg. Ces requêtes peuvent aussi servir à modifier la description des ressources. Lorsque qu'une requête reg est reçue par le serveur et que la ressource existe déjà, les descriptions dans la requête qui n'existent pas sont ajoutées à l'enregistrement dans la base de données. De même, lorsqu'une requête dereg est reçue et qu'elle contient des descriptions, l'enregistrement dans la base de données est modifié pour enlever les descriptions spécifiées dans la requête. Si la requête ne contient pas de description alors l'enregistrement est détruit. Ces deux types de requêtes sont idempotents et aucun mécanisme de sécurité ne leurs sont associés. Le format des requêtes reg/dereg est le suivant :

reg | dereg :/[*rp*]/[*na*]/<*url*>;<*desc1*>;<*desc2*>...

où

rp = chemin de résolution

na = autorité de nommage

url = URL à enregistrer ou dé-enregistrer

$descN$ = description de l'URL

Chez le serveur, les ressources sont conservées dans un fichier de configuration. Ce fichier est lu au départ par le serveur et ensuite les ressources peuvent être modifiées par des requêtes reg et dereg. Les enregistrements ont la structure suivante :

<resource URL>
    <description1>
    <description2>
        ...
    <descriptionN>

Un exemple d'enregistrement pour une imprimante pourrait être :

printer://dukprunz.watson.ibm.com/j1j25ps
    name=j1j25ps
    location=j1-j25,j1j25
    queues=j1plam,j1color,j1foils
    os2 postsctipt personal printer color foils
    access via TCP/IP lpd lpr
    local=129.34.16/24,9.2.46.0/25

Lorsque le serveur reçoit une requête pour la localisation d'une ressource, une recherche linéaire est effectuée à travers la base de données pour l'association des mots clés. Il est à noter que chaque serveur RDP est indépendant et que des serveurs RDP ne coopèrent pas ensemble pour retrouver une ressource qui satisfait une requête.

1.2.6 JINI

Jini est une architecture qui permet de grouper des périphériques et des composants logiciels en fédération dans un système réparti dynamique [46]. Ces périphériques et

ces composantes logicielles deviennent des services. Jini fournit des mécanismes pour la construction de services, la recherche de services, la communication avec les services et leur utilisation dans les systèmes répartis.

Un système Jini est composé de clients, de services et de services de recherche appelés Lookup Service. Quatre protocoles sont aussi intégrés dans l'architecture pour son fonctionnement, soit le protocole de service, le protocole de découverte, le protocole d'adhésion et le protocole de recherche. Le protocole de service est utilisé par les services pour communiquer entre eux. Le protocole de service est un ensemble d'interfaces écrites en Java. Le protocole de découverte permet aux services de découvrir un Lookup Service après quoi ils peuvent se joindre au système en utilisant le protocole d'adhésion. Enfin, le protocole de recherche est utilisé par les clients de Jini pour rechercher les services qu'ils désirent.

Pour qu'un service puisse être utilisé par des clients, il doit d'abord être enregistré auprès d'un Lookup Service. Un ou plusieurs Lookup Services peuvent exister sur un réseau. Chaque Lookup Service assistera un ensemble de groupes appelés communauté [29]. Deux formes différentes du protocole de recherche sont disponibles. La première forme est utilisée lorsque le fournisseur du service ne connaît pas de Lookup Service. Cette forme de découverte permet deux types d'interaction : le service envoi des requêtes pour trouver des Lookup Services ou bien le Lookup Service envoi des requêtes pour s'annoncer auprès des services. Dans les deux cas, le multicast IP est utilisé pour l'envoi des requêtes. La seconde forme de découverte est utilisée par le fournisseur du service pour contacter un Lookup Service particulier. Cela implique que le fournisseur connaît le URL du Lookup Service désiré.

Lorsqu'un service a découvert un Lookup Service, il peut obtenir un objet proxy sur ce Lookup Service qu'il utilisera pour communiquer avec lui. Ensuite, le fournisseur du service utilise le protocole d'adhésion pour s'enregistrer dans le Lookup Service. Lors de l'enregistrement, le fournisseur fournit un objet proxy pour le service qu'il ajoute dans

le Lookup Service ainsi que des attributs qui décrivent le service. L'objet mandataire contient une interface Java au service, incluant les méthodes que les clients peuvent invoquer pour exécuter le service. L'objet proxy peut être en réalité soit un objet qui est sérialisé pour être transporté sur le réseau ou une référence RMI à un objet distant qui implante le service.

À partir du moment où le service se trouve dans un Lookup Service, il est prêt à être recherché et utilisé. Le protocole de recherche est utilisé par les clients pour retrouver un service. Pour ce faire, un client spécifie le type de l'objet Java ainsi que les attributs qu'il désire pour le service s'il le veut. Si un service qui correspond à la requête de l'usager est retrouvé, une copie de l'objet mandataire qui se trouve dans le Lookup Service est retourné chez le client. Ce dernier utilise l'objet proxy pour communiquer avec le service.

En résumé, l'architecture de Jini permet à des utilisateurs ou à des applications de récupérer des services dans un système distribué. Les services sont utilisés comme n'importe quels autres objets Java, peu importe leur implantation et où ils se trouvent sur le réseau. L'architecture Jini est simple à utiliser et elle peut profiter de tous les autres services disponibles dans l'environnement Java tels que le service d'événements et le service de transactions.

Chapitre 2

Les composants répartis et la gestion de réseaux

2.1 L'approche des composants répartis

La répartition d'un système ou d'une application sur plusieurs ordinateurs différents est possible depuis longtemps. À l'origine, la communication par socket était utilisée par les applications pour communiquer et échanger de l'information avec d'autres applications. Parmi ces applications, certaines agissaient comme clients et d'autres comme serveurs. L'utilisation des sockets sans aucun mécanisme d'encodage implique que le client et le serveur soient sur des plates-formes identiques.

Par la suite, on a voulu rendre transparente la localisation des différentes applications d'un système réparti afin que sa programmation soit sensiblement identique à celle d'un système centralisé. C'est alors que la technique d'appel de procédure distante, RPC (Remote Procedure Call) [25], est apparue. Cette technique permet de faire appel à une procédure qui s'exécute sur une autre machine de la même manière qu'une procédure qui s'exécute localement. De plus, la machine client et la machine serveur peuvent être d'architectures différentes.

L'appel d'une procédure distante est modélisé dans une procédure locale, mais son exécution est effectuée sur une autre machine. Pour définir une telle procédure, on utilise un langage de définition d'interfaces appelé Interface Definition Language (IDL). Ce langage sert à définir la procédure avec ses paramètres d'entrée et de sortie, et à partir de cette définition il est possible de générer la souche du client et la souche du serveur dans le langage de programmation désiré. Pour chaque procédure, la souche du client et la souche du serveur ont un numéro identique unique qui leur est affecté. La souche du client est une procédure qui effectue l'encodage des paramètres d'entrée de la procédure locale dans un format standard, envoie une requête contenant le numéro de la procédure et les paramètres encodés au serveur et décode les paramètres reçus dans le message retourné par le serveur pour les mettre dans les paramètres de sortie locaux. La souche du serveur est une procédure qui effectue le décodage des paramètres contenus dans le message qu'elle reçoit, exécute la procédure locale du serveur avec ces paramètres, encode les paramètres de sortie de la procédure dans le format standard et envoie le message de réponse contenant ces paramètres au client. Pour qu'un client puisse utiliser les services d'un serveur, il doit connaître la localisation du serveur, c'est-à-dire sur quel port il écoute. Afin de demeurer transparent, un serveur de liaison est utilisé pour connaître la localisation des services. Lorsqu'un serveur est démarré, il s'inscrit auprès du serveur de liaison pour enregistrer ses services. Quand un client démarre, il envoi un message au serveur de liaison pour obtenir le port du serveur qui offre le service désiré. Ensuite, lorsque le client fait appel à une procédure souche, la procédure distante est appelée en envoyant un message sur le port du serveur. Ainsi, grâce au serveur de liaison, l'application client n'a pas besoin d'être modifiée même si le serveur change de port.

Depuis quelques années, avec la prolifération des langages orientés objet, on a voulu appliquer la technique RPC mais aux objets plutôt qu'aux procédures. On voulait pouvoir utiliser des objets se trouvant sur d'autres machines de la même manière que s'ils étaient

sur la machine locale. C'est alors que sont apparues des architectures telles que Common Object Request Broker Architecture (CORBA) et Distributed Component Object Model (DCOM). Ces architectures sont appelées middleware pour composants répartis parce qu'elles font le lien entre les composants qui résident sur différentes plates-formes pour leur permettent de coopérer ensemble, comme s'ils étaient tous sur une même machine. Elles permettent aux programmeurs de toujours utiliser les composants de la même manière indépendamment du type des plates-formes, des langages de programmation et des protocoles de communication utilisés. Pour y parvenir, elles doivent automatiser certaines tâches de la programmation réseau telles que la localisation et l'activation des composants, l'encodage et le décodage des paramètres et le traitement des erreurs. Un composant est un objet local ou distant et le terme objet peut être employé pour désigner un composant.

Ces architectures ont pour but d'intégrer des systèmes d'information hétérogènes. La capacité d'abstraction est la clé de l'intégration des systèmes hétérogènes. Elle vise à simplifier l'intégration en identifiant les éléments communs du système d'information. Aussi, elle sert à rapprocher le système d'information du modèle des activités et des structures d'une entreprise. Enfin, l'abstraction permet de faciliter l'adaptation du système d'information en minimisant le travail de coordination entre le modèle et le système technique [18]. L'approche orientée objet est la technique employée pour construire et manipuler les abstractions.

L'interface d'un objet protège le monde extérieur des changements de sa structure interne. Il est parfaitement envisageable de substituer une implantation à une autre sans modifier l'interface de l'objet. Ce principe de substitution est la principale caractéristique des systèmes de composants répartis. Les systèmes de composants peuvent ainsi s'adapter facilement et rapidement aux spécifications changeantes et aux modifications.

Un composant peut à la fois être client et serveur selon qu'il envoie ou qu'il reçoive une requête d'un autre composant. Un composant client accède à un composant serveur par

une référence et à son implantation par l'entremise d'une interface. La référence évite au composant client d'avoir à connaître la localisation du composant serveur. L'implantation et la localisation des composants peut changer sans avoir modifier les applications qui les utilisent. Un serveur s'occupe de gérer le flot de requêtes et l'état des composants dont il est responsable. Il doit, si nécessaire, activer le composant à qui est destinée une requête reçue une requête et récupérer l'information sur son état. La mémorisation de l'état des composants est normalement prise en charge par un composant de l'architecture de distribution et peut reposer sur des fichiers, des bases de données ou tout autre mécanisme de persistance. Enfin, le serveur exécute la requête de l'objet et retourne les résultats au client.

Dans les systèmes de composants répartis, on peut invoquer des messages statiquement ou dynamiquement. L'invocation statique implique que l'interface d'un objet est connue par le client au moment de la programmation. Par contre, l'invocation dynamique permet d'envoyer des messages à un objet sans connaître ses interfaces au moment de la programmation. La découverte des interfaces d'un objet par un client se fait pendant l'exécution de ce dernier. Ce type d'invocation offre la possibilité de communiquer avec des applications développées indépendamment et ajoutées ultérieurement au système réparti.

2.1.1 CORBA

CORBA est une architecture pour la répartition d'objets dont les spécifications et la normalisation sont contrôlées par l'Object Management Group (OMG) [3]. Cette entité, créée en 1989, est un consortium de plus de 800 compagnies dont la mission est de promouvoir la théorie et la pratique de la technologie objet pour le développement de systèmes répartis. Son but est de fournir une architecture de cadres d'application, pour les applications orientées objet, basée sur des spécifications d'interfaces disponibles au grand public. La spécification de CORBA 2.0 est celle utilisée dans cet ouvrage puisque

c'est la version actuellement disponible sur le marché.

L'architecture de gestion d'objets de CORBA, appelée Object Management Architecture (OMA), définit des règles pour que l'interaction entre les composants soit indépendante des protocoles de réseaux utilisés. Cette architecture est composée d'un modèle objet et d'un modèle de référence [36]. Le modèle objet définit la sémantique de la spécification des caractéristiques des objets indépendamment de l'implantation choisie. La spécification des objets est effectuée avec un IDL et leur implantation dans un langage de programmation donné est produite par un compilateur IDL. Dans le modèle objet de l'OMA, un objet a une identité distincte et ses services peuvent être accédés à partir d'interfaces bien définies.

Le modèle de référence de l'OMA caractérise l'interaction entre les composants. Ce modèle comprend quatre catégories d'interfaces et le composant Object Request Broker (ORB) (figure 3). Les interfaces de services des objets fournissent des services généraux qui peuvent être utilisés par tous les objets répartis, peu importe leur domaine d'application. Une quinzaine de services ont été spécifiés jusqu'à maintenant par l'OMG, entre autres le service de noms, le service de gestion du cycle de vie des objets, le service de sécurité, le service de transactions et plusieurs autres [38]. Les interfaces d'outils courants ont un caractère horizontal et fournissent des services pour des outils aux applications des usagers dans la plupart des domaines. Les interfaces de domaine ont un caractère verticalt et elles comportent des services qui portent sur des domaines spécifiques d'applications. Des exemples de domaines pour lesquels l'OMG a fait des requêtes de propositions d'interfaces sont le domaine des télécommunications, le domaine médical et le domaine financier. Enfin, les interfaces d'application sont développées spécifiquement pour des applications données. Cette catégorie d'interface n'est pas normalisée, car l'OMG ne développe pas d'application mais seulement des spécifications. Un ORB permet aux composants de communiquer dans un environnement réparti de façon transparente. Il permet à des programmes d'invoquer des méthodes sur des composants tout en faisant

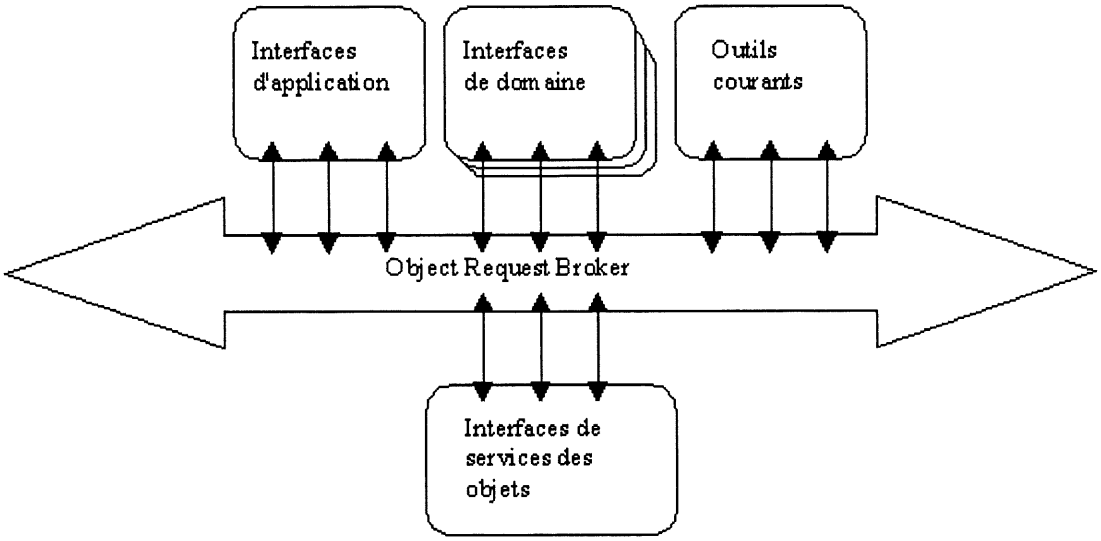

FIG. 3 – *Modèle de référence de l'OMA*

abstraction de leur localisation, du langage de programmation dans lequel ils sont implantés, de la plate-forme du système d'exploitation sur lesquels ils se trouvent et des protocoles de communication. Sa tâche consiste à fournir des services pour localiser un composant serveur qui implante un service indiqué dans une requête d'un composant ou programme client, à établir une connexion avec le composant serveur et lui communiquer les données de la requête, à activer et désactiver les composants ainsi qu'à générer et interpréter les références aux composants. Par conséquent, un programmeur peut s'occuper seulement des détails concernant le domaine de son application et ne pas s'occuper des détails concernant la programmation de bas niveau des systèmes répartis.

La figure 4 montre les composants de CORBA et les relations entre chacun d'eux. Le client est un programme qui invoque les opérations d'un servant. Le servant peut résider sur une autre machine ou sur la même machine que le client. Un servant est un ensemble d'objets qui implantent les services définis à l'aide du langage IDL. Le langage IDL de CORBA permet qu'une interface hérite des services spécifiés dans plusieurs autres interfaces (héritage multiple).

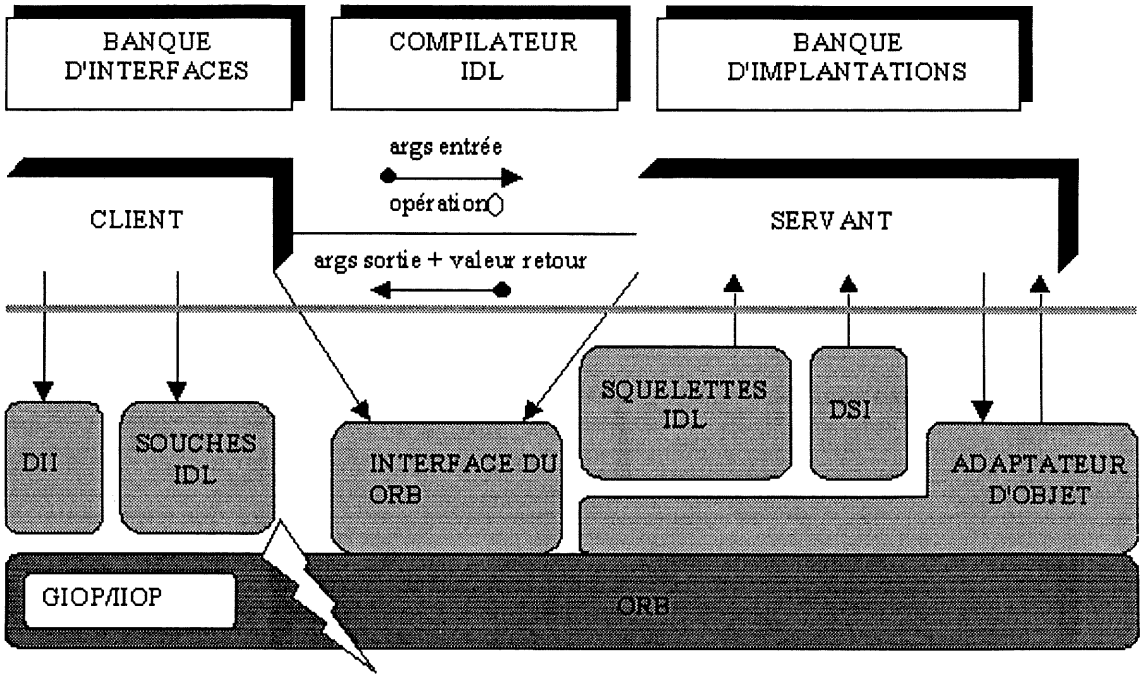

FIG. 4 – *Composants de CORBA*

Le compilateur IDL sert à générer l'interface des objets dans un langage de programmation à partir des interfaces IDL. Il met aussi à jour la banque d'interfaces et la banque d'implantations. La banque d'interfaces et la banque d'implantations sont des objets CORBA dont les opérations peuvent être invoquées comme tous les autres objets. La banque d'interfaces offre des services pour sauvegarder l'information sur l'implantation des objets par rapport à l'information IDL. Ainsi, les informations concernant l'association entre les types de données du langage IDL et ceux du langage de programmation peuvent être accédées et écrites lors de l'exécution. Elle permet aux applications de parcourir une hiérarchie d'informations IDL. Elle est particulièrement utilisée lors de l'invocation dynamique d'une opération. La banque d'implantations offre des services pour installer les implantations des objets ainsi que les règles pour activer et exécuter ces objets. Elle contient des informations qui permettent au ORB de retrouver et d'activer les implantations des objets. La plupart des informations dans la banque d'implantations

sont spécifiques à l'environnement d'exécution.

Comme cela a déjà été mentionné, un ORB est responsable de livrer au servant la requête qu'un client invoque par une opération et de retourner la réponse au client s'il y en a une. Un ORB est normalement implanté comme une librairie dynamique liée aux applications client et serveur. Le client et le servant utilisent des services standards fournis par un ORB via son interface. L'interface d'un ORB permet de dissocier les détails de son implantation des applications. Les opérations standards fournies permettent entre autres de convertir les références des objets en chaînes de caractères et vice versa, ainsi que de créer des listes d'arguments pour les requêtes faites à travers l'interface d'invocation dynamique.

Le client se sert d'interfaces statiques Static Invocation Interface (SII) pour invoquer des opérations sur les objets. Ces interfaces sont fournies par les souches IDL. Les squelettes IDL sont l'équivalent des souches du client chez le servant. Il y a une souche et un squelette pour chaque méthode d'une interface d'un objet. Ils sont produits par le compilateur IDL. En plus de fournir une interface, la souche encode les requêtes d'une application en une représentation commune pour la transmission à l'objet cible. Un squelette transforme la requête reçue dans la représentation commune en un format qui est approprié pour l'application. Les souches et les squelettes servent donc à faire le lien entre le ORB et le langage de programmation des applications afin que le ORB soit indépendant de tout langage de programmation.

Un client peut aussi invoquer les requêtes d'un objet sans avoir connaissance de ses interfaces au moment de la compilation. Cette capacité est possible grâce à l'interface d'invocation dynamique (DII). Des opérations sont fournies par tous les objets CORBA pour permettre de créer des requêtes dynamiques sur eux, ainsi que pour fournir les valeurs des arguments de ces requêtes. L'interface squelette dynamique (DSI) est équivalent à la DII du côté du servant. Cette interface permet à un ORB de livrer des requêtes à un servant qui n'a pas connaissance à la compilation de l'interface IDL qu'il implante. Un

client n'a pas besoin de savoir si un servant utilise un squelette statique ou dynamique.

Il existe trois façons d'invoquer les requêtes dynamiques d'un objet : l'invocation synchrone, l'invocation synchrone différée et l'invocation unidirectionnelle. Avec l'invocation synchrone, un client qui invoque une requête bloque jusqu'à ce qu'il reçoive une réponse, comme avec une souche statique. L'invocation synchrone différée permet à un client d'envoyer une requête, de continuer son traitement et de cueillir la réponse lorsqu'elle est reçue. Enfin, avec l'invocation unidirectionnelle, un client invoque une requête et continue son traitement puisqu'il n'y a pas de réponse pour cette requête. Ce type d'invocation est aussi supporté par les souches. La DII offre plus de flexibilité que les souches statiques, par contre son coût d'utilisation est beaucoup plus élevé que celui des souches. En effet, une requête dynamique peut impliquer qu'un ORB ait besoin des services de la banque d'interfaces pour obtenir de l'information sur les types des arguments et de la valeur de retour de la requête. Puisque que la banque d'interfaces d'un ORB est elle-même un objet CORBA, chaque requête pour un service peut en fait invoquer une requête sur un objet distant. Ainsi, la création d'une requête dynamique peut nécessiter plusieurs invocations distantes et augmenter le temps de traitement.

Lorsqu'une requête est reçue par un ORB, ce dernier doit déterminer qu'elle implantation objet peut traiter cette requête. Un ORB fait appel à un adaptateur d'objets pour effectuer cette tâche. L'adaptateur d'objets permet l'enregistrement d'un objet d'un langage de programmation quelconque comme une implantation d'un objet CORBA. Il est ensuite responsable de générer une référence pour chaque objet CORBA, de trouver le servant à qui une requête reçue par le ORB s'adresse et d'appeler la méthode appropriée sur cet objet. Si l'objet en question n'est pas actif lorsque la requête arrive, l'adaptateur d'objets peut l'activer. Il peut aussi activer, si nécessaire, le processus serveur dans lequel l'objet peut être activé. C'est particulièrement grâce à l'adaptateur d'objets que CORBA peut supporter diverses implantations d'objets. CORBA permet d'avoir plusieurs adaptateurs d'objets qu'on appelle Portable Object Adapter (POA). On retrouve

normalement un POA pour chaque langage de programmation.

Enfin, lorsqu'un client invoque une requête d'un servant, le ORB du client et celui du servant doivent nécessairement communiquer ensemble. Les protocoles inter-ORB sont utilisés pour la communication entre des ORB hétérogènes. Le protocole General Inter-ORB Protocol (GIOP) est une spécification abstraite de la représentation des données et d'un ensemble de formats de messages qui peuvent être utilisés sur n'importe quel protocole de transport. Le protocole Internet Inter-ORB Protocol (IIOP) spécifie comment GIOP peut être implanté sur le protocole de transport TCP.

2.1.2 DCOM

DCOM est l'architecture de distribution d'objets de Microsoft. Elle est une extension de COM qui permet l'interaction entre des objets qui s'exécutent sur des stations différentes d'un réseau. DCOM utilise une couche RPC objet au-dessus de DCE RPC pour supporter les objets distants. COM est une infrastructure d'intégration de composants qui interagissent dans un même processus ou entre plusieurs processus sur une même machine. Il est à la base de OLE, ActiveX, et d'autres services offerts pour les systèmes d'exploitation de Microsoft. Un COM est implanté dans une librairie dynamique (Dynamic Link Librairies) afin de pouvoir remplacer le code binaire sans affecter les clients [69].

Un objet COM n'est pas un objet au sens de l'orienté objet, mais plutôt un objet ActiveX. Il peut avoir plusieurs interfaces, chacune représentant une vue ou un comportement différent de l'objet. Un client COM peut interagir avec un objet COM en acquérant un pointeur sur une de ses interfaces et en invoquant les méthodes au moyen de ce pointeur. Les interfaces de COM ont la même norme de représentation de la mémoire que la table de fonction virtuelle de C++. Puisque cette spécification est au niveau binaire, elle permet d'intégrer des composantes binaires écrites dans différents langages de programmation.

Avec COM, la spécification des objets s'appelle une classe. Une classe implante

un ensemble d'interfaces qui sont définies par héritage simple d'autres interfaces. La spécification des interfaces peut être faite en IDL de Microsoft ou n'importe quel langage de programmation. Cela est possible parce que l'infrastructure de COM est complètement dissociée de l'outil utilisé pour créer et utiliser les composants COM.

Les classes et les interfaces COM sont identifiées par la même identification unique universelle utilisée pour identifier les interfaces DCE RPC. Toutefois, une classe n'a pas de liaison fixe avec un ensemble d'interfaces particulier. De plus, un objet (instance d'une classe) n'est pas obligé de supporter le même ensemble d'interfaces que les autres instances de la classe. Les classes des objets ont une identification globale unique (GUID). Un identifiant de classe (Class ID) permet d'identifier sans collisions les classes d'objets particulières dans un espace de noms décentralisé.

Pour utiliser un objet, le client appelle une des fonctions de la librairie COM pour créer un objet. La librairie COM recherche, à l'aide du registre du système, le code binaire approprié à l'objet, crée l'objet et retourne un pointeur sur l'interface au client. Le Service Control Manager (SCM) est une partie de librairie COM qui est responsable du contrôle des objets qui sont distants [24]. Si l'objet à créer réside sur une autre machine, il se connecte au SCM de la machine serveur et envoi la requête de création de l'objet. Pour la transparence, les noms des machines serveurs pour les objets sont dans le registre système.

La transparence du passage de la référence d'un objet d'une machine à une autre permet d'obtenir, via l'appel d'une méthode d'un premier objet, une référence à un deuxième objet et de l'utiliser ensuite sans l'intervention du premier objet. Lorsqu'un client fait appel à un objet et qu'il obtient une référence à un deuxième objet, COM décode la référence (identification de la machine, identification de l'objet) à l'objet, recherche l'identifiant de l'interface dans le registre du système afin de créer et d'initialiser un objet proxy. Le proxy se connecte à la machine et au deuxième objet. Le client reçoit alors un pointeur sur l'interface de l'objet proxy qu'il peut utiliser directement pour appeler le deuxième

objet [24].

Le nom d'une instance d'un objet est appelé un moniker et est lui-même un objet. Le moniker est responsable de trouver une instance d'un objet qui s'exécute. Si l'instance n'existe pas, il la crée et l'initialise. Si l'instance existe, il retourne une référence à son interface. Une instance d'un objet est en charge de créer elle-même un objet moniker et de le passer aux clients intéressés à se reconnecter à lui ultérieurement. Elle enregistre aussi son moniker avec la librairie COM. Le système maintient une table des noms des instances d'objets qui s'exécutent.

Le protocole de communication de DCOM, appellé ORPC (Object RPC), est une extension du protocole standard de RPC. Il spécifie comment les appels sont effectués à travers le réseau et comment les références aux objets sont représentées et maintenues. Le protocole ORPC utilise les paquets standards de RPC avec des informations additionnelles spécifiques à DCOM. Le langage de définition d'interfaces de Microsoft (MIDL) peut être utilisé pour générer automatiquement le code nécessaire pour transférer les données sur le réseau.

Le langage IDL de Microsoft est une extension du IDL de DCE RPC. Il supporte tous les types de données du langage C, mais ne définit pas un ensemble de types de données communs accessible par tous les langages de programmation comme avec le langage IDL de CORBA. Ainsi, des interfaces et des types de données définis et accessibles en C ou C++ ne sont pas accessibles dans d'autres langages. MIDL sert à générer la souche, qui est le code du client, et le proxy, qui est le code du serveur.

Les clients et les objets COM peuvent utiliser une librairie de types pour emmagasiner ou découvrir lors de l'exécution les interfaces d'autres objets. L'interface d'invocation dynamique de COM s'appelle l'interface d'expédition. Un client construit une structure de données pour sa requête et invoque une de ses méthodes avec la structure. L'objet doit implanter l'interface IDispatch, incluant la méthode invoquée avec la structure.

Même si l'architecture de DCOM est très différente de celle de CORBA, elle a le

même but, soit de permettre d'utiliser des objets à distance de façon transparente. Elle a, elle aussi, des mécanismes pour l'invocation dynamique de méthodes, la gestion des erreurs et des exceptions, l'identification et la persistance des objets, l'activation et la désactivation des objets. Toutefois, DCOM a le désavantage d'être étroitement lié aux systèmes d'exploitation et aux autres produits de Microsoft. Les composantes de son architecture sont difficiles à discerner, car elles sont toutes construites comme une extension d'autres composantes existantes qui n'ont pas nécessairement la même utilité au départ. Cela fait qu'il n'est pas vraiment extensible puisqu'il devient dépendant de la plate-forme utilisée.

2.1.3 Autres architectures

Il existe d'autres architectures de composants répartis telles que ILU de Xerox [4] et RMI de SUN [5]. ILU est disponible seulement pour les systèmes d'exploitation UNIX et il accepte des composants écrits dans différents langages de programmation. ILU n'est pas très connu car la communauté UNIX est plus présente dans le monde universitaire et de la recherche que dans les compagnies. RMI permet de répartir des composants Java et d'appeler leurs services à partir d'une application Java. Son utilisation est donc restreinte à ce langage.

2.2 CORBA et le temps réel

Jusqu'à maintenant, il n'existe pas de spécification de CORBA pour le temps réel. Toutefois, puisque des applications dans certains domaines ont besoin de tenir compte de contraintes de temps, l'OMG a émis un RFP (Request For Proposal) pour des spécifications visant à rendre CORBA capable de supporter le temps réel [37]. Dans un premier temps, nous allons présenter les principales demandes de l'OMG pour CORBA temps réel. Ensuite, nous analyserons les réponses qui ont été soumises au RFP par différentes

compagnies. Finalement, nous allons montrer d'autres solutions pour CORBA temps réel qui résulte de recherches faites dans des universités et qui n'ont pas été soumises comme réponse au RFP de OMG.

2.2.1 Requête de l'OMG pour CORBA temps réel

Puisque l'OMG a un besoin urgent de spécifications pour le temps réel, seulement quelques caractéristiques ont été énoncées dans le RFP comme devant faire parties des spécifications. L'OMG demande qu'un ORB temps réel comprenne la planification par priorité fixe, le contrôle sur les ressources du ORB pour la prévisibilité de bout en bout et la flexibilité des communications.

Afin que CORBA puisse supporter le temps réel, l'OMG convient que le comportement d'un ORB à l'exécution doit être prévisible et être contrôlé en fonction du temps du monde réel. Le contrôle peut être exercé par les applications de façon directe ou indirecte. Avec le contrôle direct, les applications indiquent leurs besoins de temps au ORB et aux services via une interface. En ce qui concerne le contrôle indirect, il se fait en gérant les ressources qu'un ORB et les services utilisent lors de l'exécution. Les ressources allouées peuvent être des ressources de traitement (e.g. cycles de processeur, éléments de synchronisation), d'espace mémoire ou de communication.

Puisque dans un système temps réel il peut parfois y avoir conflit entre l'ordonnancement des événements par priorité ou par échéancier, l'OMG préconise la technique de planification par priorité fixe pour résoudre ce problème. Cette technique consiste à affecter une priorité aux tâches au moment de la conception pour que leur ordre d'exécution soit contrôlé et prévisible lors de l'exécution. Avec cette technique, les priorités sont affectées par les applications et le ORB ne peut les modifier. Toutefois, il se peut qu'il se produise une inversion de priorité (c'est le cas lorsqu'une tâche avec une priorité élevée est bloquée en attendant qu'une tâche avec une priorité plus faible termine de s'exécuter) en utilisant cette technique, mais l'OMG considère que cette inversion de priorité est

strictement limitée et ne considère pas d'autres méthodes pour l'instant.

Pour les systèmes temps réel qui n'ont pas besoin d'une garantie absolue, la planification par priorité fixe est trop restrictive. L'OMG suggère donc de contrôler la gestion des ressources, comme deuxième technique, pour atteindre le degré de prévisibilité dont les systèmes ont besoin.

Finalement, en plus de pouvoir contrôler les ressources de communication, un ORB temps réel doit être capable d'accepter des protocoles de transport spécifiques. Les applications temps réel doivent être en mesure de pouvoir ajouter au ORB leur propre protocole de transport, et cela de manière à ce que se soit portable.

2.2.2 Solutions pour CORBA temps réel en réponse au RFP de l'OMG

Dans cette section, nous allons discuter des différentes réponses qui ont été soumises à OMG comme solution pour CORBA temps réel. Les cinq compagnies ou groupes de compagnies qui ont travaillé sur le sujet sont Objective Interface System [44], un groupe formé de Iona Technologies et Northern Telecom [71], un groupe que nous allons appeler AHLOST (Alcatel, Hewlett-Packard, Lucent Technologies, Object-Oriented Concepts, Sun Microsystems et Tri-Pacific) [11], un groupe formé de Visigenic Software et Highlander Communications [20] et enfin Lockeed Martin Federal System [43].

Nous n'allons pas présenter en détail les différentes solutions ainsi que chacune de leur API. Nous allons tout d'abord donner un aperçu de chacune des solutions. Ensuite, nous allons présenter seulement les grandes parties de ces solutions et nous concentrer sur les points les plus importants. Puisque certaines solutions ont des similitudes avec d'autres solutions, nous allons présenter les solutions selon un ordre prédéterminé. Lorsque les solutions pour un point donné seront communes pour plusieurs compagnies ou différentes pour une compagnie quelconque nous le mentionnerons.

Solution de Objective Interface Systems Inc.

La solution de Objective Interface System respecte le temps réel logiciel et le temps réel matériel. Elle nécessite que le ORB temps réel soit implanté sur une plate-forme temps réel. Les spécifications ont été écrites en se basant sur les extensions temps réel POSIX. La solution tient compte du fait qu'un système CORBA temps réel doit être capable d'interagir avec des systèmes CORBA non temps réel sans qu'il y ait d'impact sur les contraintes de temps des composants temps réel. Puisque le temps d'exécution des tâches est primordial pour la prévisibilité des systèmes temps réel, la spécification mentionne que toutes les implantations d'un ORB temps réel devraient fournir leurs métriques dans leur documentation. Les métriques sont les temps d'exécution des méthodes des objets.

La spécification parle surtout des priorités des threads, de la planification en fonction de la priorité, du verrouillage selon la priorité, du contrôle synchrone et asynchrone de threads ainsi que des ressources.

Solution de Iona Technologies et Northern Telecom

Cette solution considère qu'un système temps réel est souvent synonyme de système à haute performance. Il est donc important que les contraintes de temps soient respectées dans le contexte du système.

Une interface *Attrlist* est utilisée pour donner des valeurs aux différents attributs pour le temps réel des composants CORBA. Les valeurs sont toutes attribuées avec des chaînes de caractères et elles sont converties par le ORB dans le bon format (numérique, booléen ou chaîne de caractères). C'est le ORB qui accepte l'attribution des valeurs pour les attributs des différents composants et qui distribue ces valeurs aux bons composants. Nous ne trouvons pas que c'est une bonne façon de procéder et nous n'en voyons pas l'avantage. Premièrement, c'est contraire aux concepts de l'orienté objet, il n'y a pas d'encapsulation par objet. Ensuite, ça demande un surplus de répartition de la part du

ORB pour donner les valeurs aux attributs des composants alors que le programmeur pourrait le faire directement.

Les auteurs mentionnent que ce n'est pas une nécessité qu'un ORB temps réel soit multithread (à l'interne), que c'est simplement une stratégie d'implantation pour atteindre certaines caractéristiques. Ils fournissent donc aucune interface pour la création ou l'utilisation de ressources telles que les threads et les primitives de synchronisation. Cela implique qu'une application qui se sert d'un ORB respectant leur spécification n'est pas portable puisqu'elle doit utiliser directement les primitives du système d'exploitation pour accéder ces ressources.

Leur ORB est optimisé pour les événements asynchrones. Chaque événement a son propre thread dédié dans un pool de threads. Cela permet d'assurer que l'événement sera expédié dès qu'il survient.

La spécification couvre beaucoup de points tels que la planification, la priorité globale, le contrôle des ressources, le POA temps réel, les liens entre objets, l'interaction temps réel des protocoles et les protocoles connectables.

Solution du groupe AHLOST

Cette solution est très complète. Elle couvre à peu près tout ce qui touche la construction de système temps réel avec CORBA. C'est une solution qui est très flexible et performante car plusieurs modules qui sont greffés au ORB sont configurables.

Une possibilité qui est présente directement à travers l'interface de cette solution et qui ne l'est pas dans les autres solutions est l'invocation asynchrone de requêtes. Le nouveau mot clé async est ajouté dans le langage IDL pour permettre aux clients d'indiquer qu'une méthode est asynchrone. Deux modèles d'invocation sont supportés. Le premier consiste à ce que le client passe un objet à la méthode invoquée qu'il pourra interroger plus tard pour obtenir le résultat de la requête. L'autre modèle consiste à fournir, lors de l'appel asynchrone d'une méthode, un objet contenant une méthode qui sera appelée par

le serveur lorsque le résultat de la requête sera prêt. Dans les deux cas, l'objet à fournir, dont l'interface est décrite dans la soumission, est un ResponseHandler.

Comme nous l'avons mentionné plus tôt, un ORB dérivé de cette solution est très flexible. Il y a deux éléments qui donnent une grande flexibilité au ORB : les références d'objets et le mécanisme d'intercepteur. Une référence d'objet est constituée principalement d'un identificateur de composant et d'un ou plusieurs composants de données. Les références d'objets permettent à un ORB de laisser les applications emmagasiner des données arbitraires. Les références d'objets offrent entre autres les possibilités :

- que les servants puissent emmagasiner certaines informations et les retrouver lorsque le client effectue une requête ;
- d'emmagasinage temporaire d'information de n'importe quel type qui doit être retrouvée plus tard pour être utilisée ;
- de passage générique d'information lors de l'invocation des méthodes d'un objet (n'importe quelles informations peuvent être passées en paramètres aux méthodes).

Un intercepteur est un objet dont plusieurs de ses opérations sont appelées par le ORB à différents niveaux d'une invocation. Les objets intercepteurs sont fournis par les applications. Ils sont comme une sorte de mécanisme d'événements ou d'appels de méthodes spécialisées. Selon les auteurs, les intercepteurs sont d'une importance critique pour la réalisation de la norme CORBA temps réel car ils permettent aux ORB d'être plus flexible dans l'implantation des stratégies temps réel.

Plusieurs catégories d'intercepteurs sont définies pour un ORB temps réel. Il y a les intercepteurs pour les clients, les intercepteurs pour les servants, les intercepteurs pour les POA, les intercepteurs pour les requêtes, les intercepteurs pour le transport, les intercepteurs pour les threads, les intercepteurs pour les initialisations et les intercepteurs pour les messages. Lorsqu'il y a plusieurs intercepteurs d'une même catégorie, ils sont chaînés ensemble et exécutés en séquence. Chaque intercepteur doit avoir une priorité d'affectée pour permettre aux applications de définir l'ordre d'exécution des intercepteurs.

Les intercepteurs peuvent utiliser des contextes de service pour échanger des données entre eux. Un contexte de service est équivalent à une référence d'objet.

Nous ne parlerons pas beaucoup des autres éléments de cette solution dans cette section car elle est similaire à celle de l'Université de Washington qui est présentée dans la sous-section 2.2.3. Cela est parfaitement normal car l'Université de Washington a collaboré à l'élaboration de cette solution.

Solution de Visigenic Software Inc. et Highlander Communications

Ces deux compagnies assument qu'un ORB temps réel doit être implanté sur un système d'exploitation temps réel et qu'un protocole temps réel doit être disponible pour que les invocations de requêtes soient en temps réel. CORBA temps réel doit être configurable pour que les applications temps réel puisse utiliser les composants de CORBA non temps réel sans que cela ait un impact sur le ORB temps réel. Elles mentionnent aussi qu'une spécification CORBA temps réel doit être implantable autant sur un environnement CORBA minimal que sur un environnement CORBA complet.

La solution à cette spécification est un nouveau module, le Common Object RealTime Service, qui permet aux applications d'exercer le contrôle et de spécifier des conventions temps réel pour le ORB et les objets de l'application. Les attributs des conventions temps réel peuvent être fixés à divers niveaux de portée, autant du côté du client que du côté du serveur. Les niveaux de portée possibles sont le ORB (portée au niveau du processus), les différentes conventions (portée au niveau de l'objet), l'interface *Current* (portée au niveau du contexte de programmation ou de l'invocation) et les règles de planification de threads (portée au niveau du RTPOA).

Solution de Lockheed Martin Federal Systems

Le groupe de cette compagnie qui a travaillé sur la solution de CORBA temps réel mentionne que le problème avec le temps réel c'est la façon dont on l'interprète. On peut

le voir comme la prévisibilité de bout en bout, comme l'ordonnancement d'exécution de tâches ou comme le contrôle de l'allocation des ressources. Puisque cette perspective est correcte et que les solutions ne sont pas mutuellement exclusives, ils ont développé quatre interfaces différentes pour les applications temps réel selon les différentes perspectives.

Leur solution est axée surtout sur la priorité des tâches, sur la gestion des ressources et sur les communications flexibles. Ils parlent aussi du planificateur encapsulé qu'ils utilisent pour encapsuler certains services du système d'exploitation local. Le planificateur encapsulé n'est pas un ordonnanceur, mais il offre les services nécessaires pour la planification des tâches. Enfin, ils mentionnent comment devrait être un POA temps réel.

Caractéristiques temps réel traitées dans les solutions

La priorité

Une priorité est une caractéristique temps réel qui est attribuée à une entité planifiable pour déterminer le moment où elle sera exécutée. Les entités planifiables utilisées dans CORBA temps réel sont les threads.

Priorité des threads Les différents systèmes d'exploitation multi-tâches ont leur propre modèle de priorité. Puisque les threads qui s'exécutent dans le système CORBA peuvent s'exécuter sur différentes plates-formes, il faut trouver un moyen de conserver leur priorité relative sur chacune des plates-formes.

Pour Objective Interface System, un intervalle de priorité doit être disponible dans l'implantation. Les priorités sont associées à celles du système d'exploitation sur lequel le ORB est implanté. Une priorité ayant un numéro élevé indique une priorité élevée. Chaque fois que les threads rivalisent pour obtenir une ressource, la ressource est allouée au thread ayant la priorité la plus élevée. Un thread peut avoir une priorité de base et une priorité active. La priorité de base est celle avec laquelle le thread a été créé. La priorité active est celle qui indique la priorité courante du thread et elle est utilisée, par

exemple, pour la planification et l'obtention des ressources. La priorité active peut refléter la priorité de base ou n'importe quelle priorité héritée de d'autres sources. À n'importe quel moment, la priorité active est le maximum de toutes les priorités dont le thread hérite à cet instant.

À la création d'un thread, celui-ci hérite de la priorité de son créateur. Lors d'un appel à un objet de contrôle de la concurrence, un thread hérite de la priorité maximum de cet objet. Dans tous les cas, la priorité cesse d'être héritée aussitôt que la condition qui a menée à l'héritage n'existe plus. La priorité du thread est alors réévaluée.

Pour Iona et Northern Telecom, il faut que la priorité soit associée à un modèle normalisé de priorité afin qu'une priorité ait la même signification sur chacune des plates-formes. Toutefois, la priorité des threads devrait être associée à un modèle normalisé de priorité seulement si elle doit être partagée par un autre ORB qui s'exécute sur une plate-forme où le système de threads a un modèle de priorité différent. L'association de la priorité normalisée à celle du système d'exploitation local devrait être faite par un service de planification temps réel qui n'est pas spécifié dans leur soumission.

Nous ne pensons pas que leur solution soit convenable, car pour connaître la plate-forme de l'autre ORB avec qui un ORB communique, des échanges supplémentaires de messages sont nécessaires. Or, dans un système temps réel on cherche à limiter les échanges afin d'assurer le respect des contraintes de temps. Cette solution serait acceptable seulement si le ORB était configuré à l'avance pour connaître les plates-formes des ORB avec lesquelles il peut communiquer. Dans ce cas, on évite de faire des transformations inutiles et on peut économiser du temps, ce qui n'est pas à négliger selon le type d'application temps réel.

Visigenic et Highlander Communication utilisent eux aussi un modèle normalisé de priorité pour associer les priorités des threads aux priorités spécifiques du système d'exploitation local. Un intervalle de 256 priorités, de 0 à 255, est offert aux clients CORBA, où 255 est la priorité la plus élevée. Toutefois, le ORB n'a pas libre accès à tout l'intervalle

de priorité du système d'exploitation pour faire son association de priorité. Il a seulement un certain intervalle de priorité de réservé parmi l'ensemble de priorités disponibles sur le système d'exploitation. Cela implique qu'une application temps réel non CORBA qui a besoin d'exécuter une tâche avec une priorité basse pourrait être privilégiée à une application CORBA temps réel qui a besoin d'exécuter une tâche avec une priorité élevée. En effet, cette situation peut survenir si l'intervalle de priorité réservé sur le système d'exploitation pour le ORB contient des priorités plus faibles que celles réservées pour les autres applications. On peut donc conclure que cette façon d'attribuer les priorités du système d'exploitation n'est pas très équitable et peut causer l'inversion de priorité. Il ne devrait donc pas y avoir d'intervalle de priorités réservé.

Lockheed Martin utilise aussi un modèle normalisé de priorité semblable à celui de Highlander Communications et Visigenic Software, à l'exception que ces priorités peuvent être associées à des priorités de l'ensemble des priorités du système d'exploitation. Une priorité peut être associée de façon indépendante à chaque méthode d'un objet client. Le client fournit au ORB ses priorités et celui-ci les passe au serveur.

Une différence qu'on peut remarquer dans leur solution est qu'ils utilisent deux sortes de priorités : la priorité d'exécution et la priorité de communication [43]. Ils font une distinction entre les deux sortes de priorité pour assurer que les ressources de communication peuvent être contrôlées de manière prévisible. La priorité de communication sert entre autres lorsque le protocole de communication utilisé permet d'associer une priorité au message envoyé ou qu'une priorité peut être associée à une connexion à un serveur.

Priorité globale Objective Interface System utilise la priorité pour donner des priorités relatives aux threads. Pour que l'ordre relatif des priorités soit préservé à travers les hôtes, les processus et les threads dans l'ensemble des programmes qui s'exécutent sous un ORB temps réel, les priorités sont transportées dans les requêtes entre les clients et les serveurs (GIOP est utilisé pour le transport de la priorité). On peut penser que cette manière de procéder est équitable, mais elle ne l'est pas vraiment.

C'est ce qu'on remarque dans la spécification de Iona et Northern Telecom. La raison est simple : une machine avec un processeur rapide peut exécuter une requête à basse priorité plus rapidement qu'une requête avec une priorité élevée sur une machine avec un processeur lent. Ainsi, selon la vitesse des processeurs des différents noeuds du système, une requête avec une priorité donnée peut ne pas être traitée au moment où elle devrait l'être à l'intérieur du système.

Pour diminuer ce phénomène d'inversion de priorité, Iona et Northern Telecom ne permettent pas explicitement de spécifier des priorités. Ils proposent plutôt de fournir aux clients un ensemble d'interfaces parmi lesquelles une application peut choisir une interface impliquant une certaine priorité dont elle a de besoin. C'est le ORB qui associera la bonne priorité pour l'interface en fonction du modèle normalisé de priorité et des priorités des autres tâches planifiées dans le système. Une telle interface n'est pas spécifiée dans leur réponse au RFP.

Dans la solution de ALHOST, il n'y a rien de mentionné quant à la propagation de la priorité et à cet aspect de la priorité globale.

Lockheed Martin se sert de messages GIOP pour passer la priorité d'un noeud à un autre. La priorité est incorporée dans le champ du protocole réservé aux paramètres de contexte du service d'un objet. Ce qui est intéressant dans cette façon de faire est que si le message est envoyé d'un ORB temps réel à un ORB non temps réel, le message ne sera pas invalide pour le système non temps réel. Dans le cas inverse, la priorité la plus faible sera donnée au service demandé. Par conséquent, leur solution permet aux ORB temps réel et aux ORB non temps réel de communiquer ensemble.

Inversion de priorité Highlander et Visigenic mentionnent qu'un ORB temps réel doit avoir des limites bien définies sur la durée de temps pendant laquelle l'inversion de priorité survient dans le ORB. Afin de minimiser et de donner des limites à l'inversion de priorité, les auteurs indiquent qu'une attention particulière devrait être portée à certains

mécanismes lors de la conception du ORB. Ces mécanismes sont :

- l'utilisation des files d'attentes basées sur la priorité ;
- l'utilisation de l'héritage de la priorité des mécanismes de verrouillage pour protéger l'accès aux ressources et aux structures de données partagées par les threads ;
- la réduction du couplage entre les threads pour accéder aux ressources et aux structures de données partagées en segmentant avec attention ces structures de données.

Lockheed Martin ajoute que lorsque le blocage inter processus est utilisé, l'héritage réparti doit être utilisé et les priorités de communication doivent être respectées. Ce mécanisme est propre à leur solution car Lockheed Martin est le seul à définir une priorité de communication.

Planification en fonction de la priorité Dans la solution de Objective Interface Systems, deux règles sont utilisées pour déterminer le thread qui doit être sélectionné pour l'exécution lorsqu'il y a plus d'un thread de prêt. Il s'agit du modèle de répartition de threads et de la règle de répartition spécifique au thread.

Le modèle de répartition spécifie une planification préemptive basée sur les files de threads ordonnées selon la priorité des threads. Chacun des processeurs d'une machine à une file pour chaque valeur de priorité. À un certain moment, chaque file d'un processeur contient exactement l'ensemble de threads ayant cette priorité qui sont prêts à être exécutés sur ce processeur. Aucun thread qui est dans la file ne s'exécute sur un processeur à cet instant. Un thread peut être dans les files de plus d'un processeur.

Chaque processeur à un thread courant qui est celui en exécution sur ce processeur. Lorsqu'un thread est sélectionné par un processeur pour être exécuté, il est enlevé de toutes les files des autres processeurs dans lesquelles il se retrouve. Le thread sélectionné est toujours celui à la tête de la file non vide ayant la plus haute priorité. Un nouveau thread courant est sélectionné aussitôt qu'il y a une file non vide ayant une priorité plus élevée que celle du thread courant ou lorsque le thread courant nécessite de retourner dans la file de threads.

La règle de répartition spécifique au thread est soit FIFO (First In First Out) avec priorité ou circulaire avec priorité. Cette règle s'applique lorsque les threads sont insérés ou enlevés des files. Les deux règles ont les mêmes effets sur les threads à l'exception que la règle de répartition circulaire avec priorité empêche un thread qui à la même priorité qu'un autre thread de monopoliser le processeur. Si le thread courant a été exécuté plus qu'une certaine durée de temps, le thread est inséré à la fin de la file et le thread à la tête de la file est alors exécuté.

De leur côté, Iona et Northern Telecom stipulent qu'un ORB temps réel devrait utiliser l'ordonnanceur fourni par le système d'exploitation plutôt que d'introduire son propre ordonnanceur. Cela évite l'inefficacité causée par la compétition entre les deux ordonnanceurs. Ils appuient leur déclaration en disant qu'il n'est pas réaliste qu'un ordonnanceur puisse être défini complètement portable, efficace et approprié pour tous les domaines d'applications.

Ils ne fournissent pas d'interface pour la création et la gestion des threads. C'est le ORB qui s'occupe d'effectuer cette tâche à partir d'un ensemble de propriétés temps réel configurables qui sont disponibles dans une interface de configuration générique.

Les interfaces de la solution de Highlander et Visigenic permettent de choisir selon quelle priorité la planification doit être faite, mais ne permet pas de choisir le mécanisme de planification à utiliser. Probablement que c'est le mécanisme de planification du système d'exploitation qui est utilisé.

Dans la solution de Lockheed Martin, ce qui est moins intéressant, c'est que la gestion des files d'attente pour l'exécution des tâches en fonction de la priorité est la responsabilité des applications. En principe, la gestion des files selon la priorité dans un système temps réel s'effectue quasiment toujours de la même manière, donc cette responsabilité devrait appartenir au ORB. Pour permettre plus de flexibilité, un ORB pourrait être configuré pour décider du type de planification à utiliser, comme le permet la solution de ALHOST.

Autres utilisations de la priorité Objective Interface Systems effectue le verrouillage des threads en fonction de la priorité à l'aide d'un objet de contrôle de la concurrence (mutex, sémaphore, barrière (sémaphore privé à deux étapes)). Lorsqu'un thread s'exécute dans une région protégée par un objet de contrôle de la concurrence, il peut être supplanté seulement par les threads dont la priorité active est plus élevée que celle de l'objet de contrôle de la concurrence.

La priorité a aussi un impact dans leur solution, il se situe au niveau du contrôle asynchrone des threads. Le contrôle asynchrone de threads est fait à l'aide de deux opérations sur les threads. Ces opérations sont la suspension et la réactivation du thread. Lorsqu'un thread est suspendu de façon asynchrone, sa priorité de base devient plus petite que la plus petite priorité permise pour un thread par un processeur. Ainsi, un thread suspendu ne peut pas être acheminé à aucun processeur, car sa priorité est plus petite que celle de n'importe quel autre thread en attente. Lorsqu'un thread est réactivé, sa priorité est réévaluée en fonction des priorités dont il hérite.

Contrôle des ressources

La définition de ressource est sensiblement la même pour toutes les réponses de RFP. Une ressource est n'importe quoi qui peut être alloué à une activité et récupéré par la suite.

Objective Interface Systems décrit seulement les besoins pour la mémoire. Puisque l'allocation dynamique de mémoire n'est pas recommandée pour les systèmes temps réel, il suggère comme technique d'utiliser un pool de mémoire.

D'après Objective Interface Systems, l'implantation d'un ORB doit fournir un pool de mémoire par défaut qui peut être utilisé par toutes les opérations en mémoire du ORB et par toutes les applications. Ce pool doit permettre l'allocation et la libération prévisible dans le temps. Cela implique l'utilisation d'algorithmes de gestion de pool qui ne fragmentent pas la mémoire. Un ORB doit permettre à l'usager de définir la taille

initiale du pool. Comme toutes les autres ressources, le pool doit supporter l'héritage de priorité et le verrouillage selon la priorité par les objets de contrôle de la concurrence. Enfin, un ORB doit permettre à l'usager de décider si la taille du pool peut être agrandie dynamiquement.

Iona et Northern Telecom gèrent trois types de ressources : les threads, la mémoire et les ressources de communication. L'allocation d'une ressource est contrôlée par un mutex, fourni par le système d'exploitation, afin qu'elle soit allouée à une seule activité. Une interface qui définit les opérations de contrôle d'une ressource abstraite est fournie. Cette interface peut être spécialisée pour différentes ressources concrètes.

Pour gérer les threads, ils utilisent des pools de threads. On peut créer autant de pools de threads que l'on désire. Lors de son initialisation, un ORB doit créer un pool de threads général. Tous les objets qui ne sont pas associés à un pool spécifique lors de leur création sont associés au pool général. Il y a aussi un pool de threads pour les événements asynchrones. Il y a des règles qui servent à gouverner les pools de threads. Premièrement, chaque objet est associé à un pool à sa création. Deuxièmement, un pool peut être associé avec n'importe quel nombre d'objets de tout type. Troisièmement, chaque thread dans un pool est créé initialement avec une pile de taille finie et une priorité. Enfin, la dernière règle, lorsqu'une requête est invoquée sur une interface, un thread avec suffisamment de pile et de priorité est recherché dans le pool associé. S'il y en a un de disponible, la requête est expédiée au thread et exécutée sous le contrôle de l'ordonnanceur du système d'exploitation. Lorsque la requête est complétée, le thread est retourné dans le pool. S'il n'y a pas de thread qui convient dans le pool, un nouveau thread peut être créé si c'est possible (si la limite de threads du pool n'est pas atteinte). Autrement, un thread qui convient peut être utilisé ou créé dans le pool général. Si ça ne fonctionne pas, la requête est conservée dans une file jusqu'à ce qu'un thread soit disponible ou qu'elle devienne obsolète.

En ce qui concerne la mémoire, un pool de mémoire de taille fixe est alloué lors de

l'initialisation d'un ORB. La mémoire de ce pool est surtout utilisée pour les tampons servant au transfert des données sur le réseau. La taille et le nombre de ces tampons sont spécifiés au ORB lors de son démarrage.

Enfin, la seule ressource de communication dont la gestion est expliquée est la file d'attente de messages. Chaque file d'attente d'un client ou d'un serveur est indépendante, il n'y a pas de gestion répartie des files d'attente à travers les différents processus. Les serveurs peuvent avoir une limite sur le nombre de requêtes qui attendent pour être exécutée. Lorsque la limite d'une file est atteinte, le comportement qui se produit est déterminé par le modèle d'interaction utilisé par les processus. Les clients peuvent avoir une limite sur le nombre de requêtes qui attendent pour être exécutées par un serveur. Ils peuvent aussi décider de ce qui arrive si la limite est dépassée, par exemple suspendre l'exécution en attendant que la ressource soit disponible ou lancer une exception.

Dans la solution de ALHOST, les ressources contrôlées sont les threads, les files d'attente de requêtes, les ressources de communication et la mémoire. La façon de procéder pour le contrôle de ces ressources est similaire à celle mentionnée dans les travaux de l'Université de Washington dans la section 2.2.3. La principale différence dans leur solution, comparativement à celles des autres réponses au RFP, c'est que les ressources sont créées par une fabrique (un objet qui est responsable de la création de d'autres objets). Il existe une fabrique pour chaque type de ressources. Une interface est disponible pour créer les différentes fabriques afin qu'elles coopèrent selon une même règle de planification.

Du côté de Visigenic et Highlander Communications, une distinction est faite entre les applications pour lesquelles il est possible de déterminer un maximum de ressources nécessaires et de les allouer à l'avance, et celles pour lesquelles cela n'est pas possible. Ainsi, un ORB temps réel doit permettre aux applications de spécifier des limites d'utilisation des ressources et de pré-allouer des ressources du ORB. Cependant, ils ajoutent que la liste des ressources qu'on peut contrôler est difficile à énumérer à cause des différents

systèmes d'exploitation temps réel qui existent. Aucune interface pour le contrôle des ressources n'a été définie pour l'instant dans leur soumission.

Finalement, dans la solution de Lockheed Martin, un gestionnaire de ressources est utilisé pour surveiller quelles applications utilisent les ressources et quelles applications sont en attente de ressources. Les applications s'adressent au gestionnaire de ressources pour acquérir ou relâcher une ressource. Le gestionnaire de ressources peut obliger une application à relâcher une ressource. Les auteurs considèrent que le contrôle des ressources est dépendant de l'application. Ainsi, le gestionnaire de ressources n'est pas défini dans le module d'interfaces de CORBA, mais dans un module d'interfaces au niveau de l'application.

Liaison entre objets

Une liaison doit être établie entre un client et un servant lorsque le client invoque des opérations sur le servant. Une liaison peut être effectuée de manière implicite ou explicite. Une liaison implicite implique qu'on ne sait pas si la liaison est établie lors de la réception d'une référence d'objet ou lors de l'utilisation de cette référence. Une liaison explicite implique que c'est l'application qui effectue elle-même la liaison entre le client et le servant. Pour le temps réel, une liaison établie signifie que des ressources de communication suffisantes sont disponibles pour la connexion entre les objets.

Il y a seulement les solutions de ALHOST, Highlander et Visigenic, et Lockeed Martin qui adaptent les liaisons entre objets pour le temps réel. La solution de ALHOST permet aux applications de spécifier des paramètres de qualité de service pour les liaisons. De leur côté, Highlander et Visigenic permettent de spécifier un temps d'échéance pour une connexion explicite. Toutefois, on ne peut pas indiquer avec quel objet on veut établir une liaison, ce qui semble indiquer que leur solution est incomplète. Finalement, Lockheed Martin permet aux applications de contrôler les liaisons en permettant non seulement de spécifier des paramètres de qualité de service, mais aussi en permettant de spécifier les

paramètres utilisés lorsque la connexion avec les sockets est établie. Cette solution est sensiblement équivalente à celle de ALHOST.

Protocoles connectables

Iona et Northern Telecom suggèrent de ne pas rendre connectable les protocoles de transport, mais plutôt la combinaison de protocoles d'interaction/transport. La raison est qu'il est très difficile de définir une interface abstraite entre le protocole d'interaction et le protocole de transport car chaque protocole de transport possède différentes caractéristiques. Il est plus facile de rendre connectable la combinaison des deux types de protocole car seule l'interface du protocole d'interaction doit être normalisée. Les vendeurs de ORB n'ont qu'à fournir une librairie pour chaque combinaison. Le protocole d'interaction servant à la communication entre des ORB temps réel, dont l'interface n'est pas spécifiée, devrait supporter certains mécanismes qui sont nécessaires pour le temps réel tels que le support de différentes synchronisations, le contrôle de flux et le passage de priorité globale. La sélection d'une combinaison de protocoles se fait en configurant un objet *REO* à la création d'un objet.

ALHOST propose une interface de communication ouverte qui définit les interfaces communes pour des protocoles connectables. Cette interface comprend deux parties, soit l'interface de transfert de message et l'interface d'opération distante. L'interface de transfert de message supporte les protocoles orientés connexion avec garantie de livraison. L'interface d'opération distante supporte les protocoles qui implantent les concepts d'appel de procédure distante. Ces deux dernières interfaces permettent l'utilisation d'un grand nombre de protocoles. Il est aussi possible de combiner les deux fonctionnalités pour permettre à une interface d'opération distante de supporter l'interface de transfert de message. L'interface de transfert de message est implantée selon les patrons de conception (design pattern) *Connecteur*, *Accepteur* et *Réacteur*. Pour brancher un protocole, il suffit de fournir des objets qui respectent les interfaces abstraites mentionnées. Cette

solution permet de brancher n'importe quel protocole, mais restreint un ORB à utiliser un seul protocole puisqu'il ne peut y avoir plus d'un objet avec un même identificateur.

Lockheed Martin procède d'une manière un peu différente. Dans leur façon de procéder, c'est l'application qui a le contrôle de la gestion des connexions entre le client et le servant. Deux interfaces sont disponibles pour permettre à l'application, via l'interface d'un ORB, de créer ou de terminer une connexion. L'interface commune que les protocoles doivent utiliser est celle des sockets Unix. Donc, pour utiliser n'importe quel protocole, il suffit d'encapsuler l'interface du protocole avec l'interface des sockets. Pour choisir le protocole qu'on veut utiliser, il suffit de spécifier les valeurs correspondant au protocole dans les paramètres famille, type et protocole lors de la création d'un socket. Des interfaces sont spécifiées pour permettre à l'application de fixer les paramètres pour chaque méthode de communication entre le client et le servant. Ce qui est bien de cette solution c'est que, contrairement à la solution précédente, celle-ci permet non seulement de brancher n'importe quel protocole, mais aussi d'utiliser plus d'un protocole à la fois.

POA temps réel

Certaines solutions nécessitent des modifications à la spécification du POA pour le temps réel. Comme le POA est impliqué dans la création ou l'activation des objets, il est normal qu'il soit modifié, entre autres pour être capable de spécifier des paramètres de qualité de service pour les objets.

Iona et Northern Telecom, de même que ALHOST, ne spécifient pas une extension au POA actuel de CORBA car leur solution implique qu'il y a trop de modes d'opération qui ne sont pas appropriés pour le temps réel. En effet, certains modes d'opération introduisent du non déterminisme ce qui compromet la prévisibilité de bout en bout. De plus, des opérations qui permettent l'association d'un pool de threads ou d'autres attributs de configuration temps réel aux objets lors de leur création sont nécessaires. C'est pourquoi ils optent pour la spécification d'un nouveau POA temps réel.

Le nouveau POA temps réel n'utilisent pas le POAManager, l'AdapterActivator, ainsi que le ServantManager car le délai et le non déterminisme produits lorsque ces objets effectuent de la création et de l'activation dynamique ne sont pas convenables pour le temps réel. Les différences et les similitudes entre le POA temps réel et le POA actuel sont mentionnées dans la spécification de Iona et Northern Telecom. Les deux solutions (Iona et ALHOST) se ressemblent beaucoup.

Lockheed Martin considère que les interfaces POA n'ont pas besoin d'extension pour le temps réel. Toutefois, ils font remarquer que ce n'est pas toutes les implantations de POA qui sont convenables pour le temps réel. Selon eux, les implantations convenables sont celles qui :

- utilisent le moins possible des segments de code critique afin de réduire le blocage de processus ;
- réduisent la mise en file d'attente des requêtes lorsqu'elles ne peuvent être traitées par l'application ;
- fournissent des mécanismes aux applications pour contrôler la priorité d'exécution des composants du POA.

Contrairement aux deux solutions précédentes, celle-ci n'élimine pas l'utilisation du POAManager, de l'AdapterActivator et du ServantManager. Chacun de ces objets est modifié pour incorporer un planificateur encapsulé afin de permettre aux applications de contrôler la priorité. Avec cette capacité, une application peut alors réduire le besoin de mise en file d'attente au niveau du POA ainsi que le besoin de section critique.

Autres exigences

Un système CORBA temps réel ne peut pas utiliser TCP, par le fait même IIOP, comme protocole de transport car il n'est pas prévisible [44]. Il faut donc utiliser un moyen de transport connectable qui peut être spécifique à l'application.

ALHOST mentionne que certains services de CORBA doivent être modifiés et optimisés pour améliorer leur performance dans les systèmes temps réel. Lockheed Martin indique que la plupart des services actuels de CORBA ne sont pas adaptés pour le temps réel car ils ne supportent pas la mise en file d'attente et la distribution basée sur la priorité, ainsi que la propagation de la priorité globale aux autres services. Ces services devraient donc être modifiés pour que les systèmes temps réel soient prévisibles.

2.2.3 Autres solutions à CORBA temps réel

Travaux de l'Université de Washington

L'Université de Washington effectue énormément de recherche sur l'utilisation de CORBA pour développer des applications distribuées temps réel. Elle a conçu TAO, un ORB temps réel qui est basé sur le cadre d'application ACE, lui-même conçu par l'Université de Washington. ACE est un cadre d'application orienté objet pour le développement d'applications réparties. TAO est utilisé par la compagnie Boeing à St-Louis, pour développer des applications pour la prochaine génération de systèmes d'aviation.

Caractéristiques essentielles pour le temps réel

TAO est conçu à partir de certaines caractéristiques que le groupe de recherche a établies comme étant essentielles pour un système de composants répartis temps réel [64]. Ces caractéristiques sont les suivantes :

1. Les applications doivent pouvoir spécifier leurs besoins de qualité de service á un ORB dans les opérations qu'elles effectuent à l'aide du langage IDL.
2. Un ORB temps réel doit être capable de planifier des ressources du système d'exploitation et du réseau ainsi que de supporter leurs options de qualité de service pour assurer une garantie de bout en bout aux applications.

3. Un ORB doit être capable d'utiliser un certain nombre de protocoles de communication personnalisés et optimisés pour les besoins spécifiques des applications et des environnements de réseaux.
4. Le démultiplexage et la répartition des requêtes temps réel doivent être optimisés pour améliorer la prévisibilité du ORB.
5. Un ORB temps réel doit optimiser la gestion de la mémoire pour ne pas dégrader sa performance. Les copies de données entre les différentes couches de son architecture ainsi que l'allocation dynamique de mémoire doivent être minimales.
6. La conversion des données de l'application dans un format portable qui est indépendant du langage de programmation et de l'architecture de la machine doit être optimale. Pour réduire les coûts de conversion, il est possible de faire des compromis entre utiliser du code compilé versus du code interprété pour la conversion. Le code compilé encodé est efficace mais nécessite beaucoup de mémoire, tandis que le code interprété est plus lent mais plus compact.
7. Un ORB temps réel doit permettre aux applications de choisir le degré de performance désiré afin de réduire les conflits entre les besoins de performance élevée et les besoins de déterminisme pour le temps réel.

Les caractéristiques 1, 3, 6 et 7 se situent au niveau des spécifications et font ou pourraient faire partie des spécifications de l'OMG pour CORBA temps réel. Les autres caractéristiques sont au niveau de l'implantation et dépendent seulement des vendeurs de ORB qui implantent les spécifications.

Implantation

TAO utilise une interface réseau au-dessus de ATM pour optimiser les sous-systèmes d'E/S conventionnels des systèmes d'exploitation afin de prévoir le traitement des protocoles à un débit très élevé. Les plates-formes sur lesquelles TAO est disponible sont

VxWorks, Chorus, Windows NT, Solaris et les implantations POSIX 1003.1c.

Des composants ont été ajoutés ou modifiés à CORBA afin de répondre aux besoins pour le temps réel. Tout d'abord, un sous-système d'E/S gigabit temps réel qui accepte la qualité de service a été ajouté pour permettre au ORB et aux applications d'accéder aux ressources de bas niveau du réseau et du système d'exploitation. Un nouveau protocole, RIOP (Real time Inter-ORB Protocol), spécialise le protocole GIOP pour permettre aux applications de transférer leurs paramètres de qualité de service. Les paramètres de qualité de service sont les attributs d'un objet particulier utilisé dans toutes les invocations de requêtes temps réel. Cela permet de garder la compatibilité avec les implantations IIOP qui ne supportent pas les extensions de qualité de service de TAO. Le protocole RIOP est conçu pour se confondre avec GIOP sur une variété de réseaux et peut être personnalisé pour les besoins spécifiques des applications.

Un adaptateur d'objet temps réel a été conçu pour planifier et expédier les requêtes en temps réel. Cet adaptateur d'objet peut être personnalisé pour implanter des mécanismes qui expédient les requêtes aux clients selon les règles de planification temps réel spécifiques aux applications. Pour être capable d'offrir cette flexibilité, l'adaptateur d'objet contient une référence d'objet à un planificateur d'exécution qui implante la planification des requêtes. Il suffit de changer la référence d'objet sur un autre planificateur d'exécution pour changer de règle de planification. On se rappelle que cette idée a été soumise dans la spécification [11] pour le RFP de CORBA temps réel.

L'adaptateur d'objet de TAO peut utiliser deux mécanismes d'expédition, soit l'expédition RTU (Real Time Upcall), soit l'expédition temps réel par thread. Leur utilisation dépend du genre d'application. TAO fournit aussi deux mécanismes de démultiplexage qui peuvent traiter les requêtes des clients de manière efficace et prévisible, peu importe le nombre de connexions actives, l'implantation des servants et les opérations définies dans les interfaces IDL. Ces mécanismes sont le schéma de découpage parfait et le démultiplexage actif [64].

Enfin, des composants spécifiques aux applications ont été ajoutés pour optimiser les sources de surcharge (mentionnées aux items 5 et 6 des caractéristiques essentielles au temps réel) et pour fournir des caractéristiques afin de supporter la qualité de service de bout en bout des applications et des services de CORBA.

Extension du service d'événements Dans un système temps réel, une des caractéristiques importantes est de pouvoir réagir aux différents événements qui surviennent en temps réel. Les recherches de l'Université de Washington sur ce point particulier visent à déterminer comment associer les résultats des couches réseau et système d'exploitation à CORBA.

TAO étend le service d'événements de CORBA pour supporter le temps réel en ajoutant des mécanismes requis par les applications temps réel. Les mécanismes ajoutés sont la planification et l'expédition d'événements temps réel, le traitement d'événements périodiques et un mécanisme de filtration et de corrélation d'événements [42].

Afin de permettre la planification et l'expédition d'événements temps réel, l'interface du service d'événements a été étendue pour permettre aux producteurs et aux consommateurs d'événements de spécifier leurs besoins et leurs caractéristiques d'exécution en utilisant des paramètres de qualité de service. Le mécanisme d'expédition du canal d'événements intègre ces paramètres de qualité de service avec les règles de planification temps réel du système pour déterminer l'ordre d'expédition des événements et les stratégies de préemption.

Le service d'événements temps réel de TAO permet aux consommateurs d'événements de spécifier des échéances de dépendance pour les événements qui se produisent périodiquement. Ainsi, un consommateur qui n'a qu'un certain temps pour effectuer son traitement peut exiger d'être averti si sa dépendance à un événement n'est pas satisfaite dans la limite spécifiée.

Enfin, le mécanisme de filtration et de corrélation, ajouté au service d'événement de CORBA, permet aux consommateurs de spécifier avec la logique OU et ET leurs

dépendances aux événements. Lorsque ces dépendances sont satisfaites, le service d'événements temps réel expédie tous les événements impliqués aux consommateurs. Ce mécanisme réduit la charge du réseau en éliminant les événements filtrés dans le canal plutôt que chez les consommateurs.

Le canal d'événements de TAO peut être configuré de manière statique ou dynamique. La configuration du canal peut comprendre tous les mécanismes ou seulement ceux nécessaires, tout dépendant des besoins des applications temps réel.

Les besoins en performance En plus de pouvoir spécifier les besoins en qualité de service, la plupart des applications temps réel ont également besoin de performance. Donc, toutes sources possibles de surcharge dans l'architecture de CORBA doivent être réduites ou éliminées. Dans TAO, plusieurs de ces sources de surcharge ont été réduites, en plus de celles que nous avons déjà mentionnées. Les optimisations qui ont été faites sont au niveau de la stratégie d'invocation des méthodes, du démultiplexage des requêtes, de la communication entre ORB [33] et des architectures multithread [63].

Stratégie d'invocation des méthodes L'optimisation effectuée à ce niveau est la diminution de la latence. Pour y parvenir, on a fait une utilisation optimale de la mémoire, réduit les appels de fonction intra-ORB, réutilisé les requêtes DII et minimisé l'encodage ainsi que le copiage de données nécessaire pour fournir les paramètres aux requêtes.

Démultiplexage des requêtes La plupart des ORB effectuent un démultiplexage par couche qui cause une augmentation de la latence. TAO utilise le schéma de découpage parfait et le démultiplexage actif sans couche comme stratégie de démultiplexage [33]. Cette dernière donne une performance meilleure que les autres stratégies de démultiplexage utilisées dans les ORB. Toutefois, elle nécessite que le client soit préconfiguré pour avoir connaissance de chacun des servants avec chacune de leurs opérations ou qu'un protocole soit défini pour gérer dynamiquement les points d'accès aux servants.

La communication entre ORB Le protocole IIOP implanté dans TAO est optimisé

pour minimiser la latence de la même manière que la plupart des autres optimisations de la latence. Les sept principes d'optimisation utilisés sont : optimisation pour le cas le plus commun, élimination des surcharges gratuites, remplacement des méthodes générales par des méthodes spécifiques plus efficace, précalcul des valeurs lorsque c'est possible, conservation des états redondants pour accélérer les opérations coûteuses, passage de l'information entre les couches et optimisation pour la cache du processeur [34]. TAO utilise ILP (Integrated Layer Processing) pour assurer que les protocoles de bas niveau manipulent les données dans les mêmes unités que celles spécifiées par l'application et diminuer le passage de données de couche en couche [33].

Les architectures multithread Plusieurs architectures multithread peuvent être utilisées dans les ORB pour améliorer la performance. L'Université de Washington a évalué plusieurs de ces architectures afin de trouver celle qui est la mieux appropriée pour le temps réel [63]. Elle a conclu que les architectures thread-per-request, thread-per-connexion et thread-per-servant utilisées dans la plupart des ORB conventionnels ne sont pas appropriées pour le temps réel à cause du non déterminisme et du mauvais balancement de la charge qu'elles impliquent.

Elle a aussi évalué les architectures utilisant un pool de threads. Ces architectures s'avèrent meilleures pour le temps réel car la surcharge causée par la création des threads est éliminée. Parmi les architectures évaluées, on retrouve l'architecture Worker Thread Pool et l'architecture Leader/Follower Thread Pool. Ces architectures ont cependant été mises à l'écart car elles peuvent causer des changements de contexte excessifs, de la synchronisation excessive pour gérer les files de requêtes ainsi que l'inversion de priorité des requêtes.

L'Université de Washington a finalement évalué des architectures hybrides composées de différentes architectures mentionnées précédemment. Il s'agit de l'architecture Threaded Framework et de l'architecture Reactor-per-thread priority. La première architecture, basée sur le pattern de la *Chaîne de Responsabilité*, a l'avantage d'être flexible car elle

comporte un mécanisme de thread filtre et permet à une application d'installer son propre thread filtre. Un filtre est un composant d'une application qui effectue un certain nombre de tâches. Par contre, elle a le désavantage de réduire le débit et d'augmenter la latence car un verrou est acquis chaque fois qu'une requête doit être insérée dans la file du pool de thread approprié.

TAO implante donc l'architecture Reactor-per-thread priority qui est basée sur le pattern *Réacteur*. Un groupe de réacteurs est associé à un groupe de threads s'exécutant à différentes priorités. Un réacteur reçoit les requêtes et les transmet à un gestionnaire de connexions approprié, qui à son tour lit les requêtes et les expédie au bon servant. Cette architecture minimise l'inversion de priorité et le non déterminisme. Elle réduit également la surcharge qui est due aux changements de contexte et à la synchronisation, en plus de supporter des techniques de planification et d'analyse qui associent la priorité avec le débit. Toutefois, elle réduit le parallélisme puisque toutes les requêtes d'un client pour un réacteur sont traitées par un seul thread.

Service de planification temps réel La planification est la tâche qui est au cœur d'un système temps réel. TAO supporte cette tâche dans un service particulier comme tous les autres services offerts par CORBA. Le ORB est conçu pour supporter les applications temps réel qui ont des besoins de qualité de service déterministe ou statistique [64]. Les applications temps réel déterministes doivent respecter des échéances de temps périodique, tandis que les applications temps réel statistiques peuvent tolérer des fluctuations mineures dans les garanties de planification et de robustesse.

Le service de planification temps réel de TAO est responsable d'allouer le minimum de ressources du système nécessaires pour respecter les besoins de l'application. Le service de planification est un objet CORBA qui a les responsabilités d'analyser la faisabilité de la planification, d'affecter des priorités aux requêtes et de coordonner les changements de mode. Un mode est un ensemble fixe de paramètres reliés à la planification.

Pour supporter la spécification de qualité de service, TAO utilise un langage IDL

étendu pour le temps réel (RIDL). Le langage RIDL comprend une interface *RT_Operation* et sa structure *RT_Info* pour permettre de spécifier au service de planification de TAO les besoins en UCT des tâches effectuées par les opérations de l'application. Les besoins de qualité de service pour l'UCT qu'il est possible d'exprimer avec la structure *RT_Info* sont : le temps de la pire d'exécution, le temps d'exécution typique, le temps d'exécution avec cache, le temps minimum entre deux itérations successives d'une opération, l'importance d'une opération, le temps de quantum et les informations de dépendance qui est une liste de structures *RT_Info*.

TAO permet de supporter plusieurs stratégies de planification statique ou dynamique pour les applications temps réel qui ont des besoins déterministes [31]. La stratégie de planification statique actuellement supportée est RMS (Rate Monotonic Scheduling). EDF (Earliest Deadline First) et MLF (Maximum Laxity First) sont les stratégies de planification dynamique supportées. TAO supporte aussi la stratégie de planification hybride MUF (Maximum Urgency First). MUF supporte le déterminisme de l'approche RMS et la flexibilité des approches dynamiques telles que EDF et MLF. L'utilisation de cette dernière stratégie est particulièrement favorisée dans TAO car elle surmonte les limites de chacune des stratégies en combinant des techniques de chacune.

Le but de la conception de cette stratégie de planification est de permettre aux applications de maximiser l'utilisation totale des ressources, de préserver les garanties de planification pour les opérations critiques et d'avoir la flexibilité de varier leurs besoins en planification. Le service de planification de TAO est conçu pour assurer que les opérations critiques de CORBA respectent leurs échéances, même lorsque l'utilisation totale excède la limite possible.

Recommandations

Bien que de permettre aux applications de spécifier leurs besoins en qualité de service et d'effectuer la planification des requêtes, un ORB temps réel se doit néanmoin d'assurer que les applications ne seront pas victime de non déterminisme et que la priorité de leurs requêtes ne sera pas inversée. L'Université de Washington a effectué une série d'évaluation et a émis certaines recommandations pour diminuer le non déterminisme et limiter l'inversion de priorité dans les ORB temps réel [65].

1. Les ORB temps réel devraient éviter l'établissement de connexion dynamique. Pour cela, elle recommande que l'OMG définisse dans sa norme pour CORBA temps réel un API qui contrôle le pré-établissement de connexions.
2. Les ORB temps réel devraient éviter de multiplexer les requêtes de priorités différentes sur une connexion partagée. Ils devraient permettre aux concepteurs d'application de déterminer eux-mêmes si les requêtes de priorités différentes doivent être multiplexées sur une connexion partagée.
3. Les ORB temps réel devraient minimiser l'allocation dynamique de mémoire.
4. Les architectures multithread des ORB temps réel devraient être flexibles, efficaces et prévisibles. Les ORB temps réel devraient fournir une API qui permet aux concepteurs d'application de sélectionner l'architecture multithread désirée.
5. Les architectures de ORB temps réel devraient être guidées par un jeu d'essais de performances empiriques pour permettre leur évaluation.

Travaux de l'Université du Rhode Island

Les travaux effectués par l'Université du Rhode Island consistent à apporter une extension temps réel aux ORB existants [73]. Le ORB utilisé pour leurs travaux est Orbix 2.0.1 MT de Iona Technologies Inc. Pour leurs recherches, le ORB a été installé sur une station de travail Sparc exécutant Solaris 2.5. Les chercheurs ont adapté le ORB

pour le temps réel en lui permettant de supporter l'expression de contraintes de temps dynamique. Le ORB supporte seulement le temps réel logiciel.

Caractéristiques essentielles pour le temps réel

D'après les chercheurs, pour supporter le temps réel dans CORBA, il y a deux catégories de besoins à combler : les besoins de l'environnement d'exploitation (systèmes d'exploitation et réseaux) et les besoins du système d'exécution de CORBA. Dans l'environnement d'exploitation :

1. les horloges des ordinateurs doivent être synchronisées ;
2. le délai pour l'échange d'un message doit être limité ;
3. la planification doit être basée sur la priorité des requêtes ;
4. tous les composants sous-jacents à l'environnement CORBA doivent implanter l'héritage de priorité.

Dans le système d'exécution de CORBA, les besoins sont au niveau du ORB et des composants qui offrent les services. Il faut que :

1. un type standard soit défini pour spécifier le temps absolu et le temps relatif ;
2. les informations relatives au temps réel soient transmises lors de l'invocation des méthodes ;
3. la priorité des tâches soit globale à travers le ORB et relative à chacune des autres tâches ;
4. tous les services de CORBA gèrent la priorité des tâches avec des files de priorité ;
5. les événements puissent être relatif au temps de d'autres événements ;
6. les services relatifs au temps réel utilisent l'héritage de priorité ;
7. le mécanisme d'exception de CORBA soit étendu pour générer des exceptions relatives au temps réel ;

8. les vendeurs documentent les pires temps d'exécution de toutes les exécutions dans leur produit ;
9. le ORB garantisse les contraintes de temps spécifiées par le client ou qu'il génère une exception.

Implantation

Tout d'abord, le IDL de CORBA a été étendu pour ajouter de nouveaux types pour le temps réel. De plus, une librairie de code a été ajoutée aux souches des clients et aux squelettes des servants pour permettre la manipulation des nouveaux types de données.

Les chercheurs ont appelé TDMI (Timed Distributed Method Invocation) le nouveau concept d'appel de méthodes avec des contraintes de temps. Une nouvelle structure RT_Environment est utilisée pour spécifier les contraintes de temps telle que l'importance, l'échéance et la période. Un nouvel objet de type RT_Manager est utilisé pour fixer les attributs de la structure RT_Environment, démarrer la TDMI et compléter la TDMI.

Lors d'une TDMI, le système d'exécution de CORBA temps réel fixe une priorité transitoire à la TDMI à l'aide du service de priorité globale et démarre un temporisateur pour l'échéance de la requête. Le nouveau service de priorité globale assure que la priorité transitoire est significative relativement aux autres priorités transitoires dans le système CORBA temps réel. Lorsqu'une TDMI est terminée, un appel d'une méthode spéciale doit être effectué pour redonner au client la priorité qu'il avait avant d'effectuer la TDMI et arrêter le temporisateur.

Un nouveau service de temps global est implanté pour que les clients et les serveurs utilisent le même temps. Le protocole NTP (Network Time Protocol) est utilisé pour synchroniser les horloges sur les différentes machines du système réparti.

D'autre part, le service de contrôle de la concurrence de CORBA a été modifié pour fournir le verrouillage avec l'héritage de priorité. Ainsi, une TDMI avec une priorité élevée

peut obtenir un verrou sur une ressource du système avant les autres TDMI de priorité moins élevée, même si elles ont demandé un verrou sur la même ressource avant elle.

Service d'événements temps réel Un service d'événements temps réel est implanté pour donner une priorité à la livraison des événements et pour donner le temps d'occurrence des événements. La priorité des événements est basée sur la priorité transitoire des producteurs et des consommateurs. Elle est importante pour maintenir la planification globale des priorités. La livraison du temps auquel les événements surviennent est importante pour permettre d'établir des contraintes de temps relatives à ces événements. L'implantation du service d'événements temps réel est basée sur le multicast de IP et prend avantage de l'environnement multithread de Solaris 2.5.

Service de planification temps réel Leur ORB implante également un nouveau service de planification temps réel basé sur la priorité transitoire des requêtes. La priorité transitoire est calculée en fonction de l'attribut d'importance et ensuite de l'attribut d'échéance. Ces attributs doivent être fournis lors d'une TDMI. La planification globale au système CORBA temps réel est donc de type EDF (Earliest Deadline First) avec un niveau d'importance. L'association de la priorité transitoire avec les priorités du système d'exploitation est effectuée par un processus démon temps réel qui s'exécute sur chaque noeud du système. Ce processus s'occupe également d'augmenter la priorité à mesure que le temps avance. Toutefois, cette fonctionnalité particulière peut ne pas être exécutée si la règle de planification utilisée ne la nécessite pas (par exemple RMS plutôt que EDF).

Opinion

Puisque les recherches de l'Université du Rhode Island ont porté sur l'ajout d'une extension aux ORB existants pour qu'ils supportent le temps réel, les ORB résultants n'offrent pas les meilleures performances possibles. Beaucoup de surcharges inutiles sont présentes dans les ORB à cause des nombreux démultiplexages des requêtes. Vu que le ORB de base n'est pas temps réel, il est trivial que le temps d'exécution d'une requête

temps réel dans un tel système est de même durée ou plus long que celui d'une requête non temps réel. Ce ORB peut donc être utilisé dans les systèmes qui nécessitent l'expression de contraintes de temps, mais ne peut pas être utilisé dans les systèmes qui ont des besoins de performance.

Travaux de l'ENST de Bretagne

L'ENST a développé COREMO [19], un modèle d'extension temps réel à CORBA. Leur approche consiste à garantir à un client que sa requête est acceptée ou rejetée avant même qu'elle soit exécutée. Si une requête est acceptée par le servant alors il est certain qu'elle sera traitée à temps.

Un client peut spécifier quatre contraintes de temps lors d'une requête. Ces contraintes sont le temps d'échéance, un taux de tolérance (pourcentage de temps alloué comme dépassement de l'échéance), l'état critique d'une requête et un critère d'estimation (les choix de critères sont : pessimiste, optimiste et moyen, mais aucune explication n'est donnée quant à leur signification). Lorsqu'une requête arrive au servant, celui-ci calcule la priorité de la requête en fonction des contraintes et décide si la requête peut être traitée ou non. Le servant renvoi une réponse au client relative à l'acceptation ou au rejet de la requête.

Le servant se sert d'un vecteur de stratégies pour déterminer son comportement concernant les aspects temps réel tels que la stratégie de planification à adopter, les requêtes à retenir et le mode d'exécution des requêtes (en série ou en parallèle). Cinq stratégies de planification peuvent être utilisées dans COREMO. La meilleure stratégie à utiliser dépend du type de méthode qu'on veut favoriser. Des informations nécessaires à leur modèle, telles que le vecteur de stratégie des servants et le temps d'exécution des méthodes au moment de l'installation, ont été ajoutées aux fichiers IDL temps réel.

Cette extension temps réel à CORBA est de base. Elle offre un minimum de support pour le temps réel et est utilisable que par les applications voulant faire du temps réel

logiciel. Néanmoins, le principe de déterminer à l'avance si une requête peut être exécutée ou non est intéressante. Cela permet à une application de réévaluer rapidement ces critères de qualité de service si sa requête est rejetée et d'envoyer à nouveau sa requête.

2.3 CORBA et la gestion de réseaux

La complexité croissante et l'hétérogénéité des réseaux ainsi que la progression du traitement réparti rend la gestion de réseaux plus importante et plus complexe. Aucune des normes de gestion présentées dans la section précédente ne permet de gérer tous les réseaux, systèmes et ressources des applications. Toutefois, puisque les systèmes de gestion de réseaux existants ont nécessité de gros investissements de la part des compagnies, ils sont là pour rester encore longtemps. C'est pourquoi plusieurs compagnies et institutions de recherche essaient de simplifier la gestion de réseaux en uniformisant la façon de gérer les réseaux. Ils veulent aussi que les systèmes de gestion soient extensibles pour être capable de s'adapter rapidement aux changements et de gérer des réseaux vastes.

La popularité croissante de CORBA dans l'industrie a amené certaines personnes à utiliser ce middleware pour intégrer ensemble les diverses normes de gestion de réseaux. L'environnement de CORBA est très approprié pour les applications de gestion puisqu'une grande majorité des systèmes répartis d'aujourd'hui utilisent déjà cet environnement. Par conséquent, les applications de gestion peuvent utiliser les composants (ORB) déjà en place pour communiquer avec les différents éléments du système de gestion. De plus, un autre signe que CORBA est approprié pour la gestion de réseaux est que le groupe de travail JIDM (Joint Inter-Domain Management) a établi la spécification d'une méthodologie pour traduire le langage de définition d'objet de CMIP, GDMO, et le langage de définition de MIB SNMP, ANS.1, en IDL de CORBA [39]. Par ailleurs, l'OMG a un groupe qui travaille spécialement sur la gestion des réseaux de télécommunication avec CORBA.

2.3.1 Systèmes de gestion de réseaux basés sur CORBA

Il existe plusieurs systèmes de gestion de réseaux (privés et commerciaux) basés sur CORBA qui ont été développés pour répartir la gestion et les composants des systèmes, permettre l'interopérabilité des systèmes de différents vendeurs et rendre transparent les normes de gestion utilisées.

Approches d'association

Diverses approches sont utilisées pour accéder aux objets gérés dans les protocoles standards en se servant de CORBA. Une d'entres elles est de créer des objets CORBA identiques à ceux de la norme de gestion comme le fait la spécification de JIDM. Toutefois, cette approche est statique (chaque objet est compilé avec l'application client) [28] et parfois les objets ont besoin d'être modifiés pour qu'ils respectent le modèle orienté objet [50]. Une deuxième approche statique connue est de créer des objets qui représentent les vrais objets gérés dans le réseau [50]. Ces objets sont formés de plusieurs objets d'un protocole de gestion.

Une autre approche consiste à utiliser un objet générique pour représenter n'importe quel objet de n'importe quelle norme de gestion [28]. Il suffit de donner l'identificateur de l'objet dans la MIB pour l'interroger ou le modifier. Cette approche est dynamique car peu importe l'objet désiré, il est toujours manipulé au moyen d'un objet générique, donc les objets n'ont pas besoin d'être connus au moment de la compilation de l'application. Enfin, une dernière approche consiste à utiliser un objet simple pour retrouver ou modifier les valeurs des objets d'une norme de gestion [28]. Il suffit encore une fois de connaître l'identificateur de l'objet dans la MIB. Cependant, les objets CORBA de cette approche respectent un peu plus le concept orienté objet.

La différence entre les deux approches dynamiques est que, même si leurs objets sont génériques, ceux de la première méthode peuvent représenter des objets autres que ceux

appartenant directement au protocole de gestion, alors que la deuxième méthode ne le permet pas. Au point de vue des concepts de l'orienté objet, la dernière approche est équivalente à la première, sauf qu'elle est dynamique plutôt que statique. Les approches dynamiques ont l'avantage de réduire la taille des applications de gestion et de les rendre flexibles. Par contre, même si elles ne nécessitent pas de connaître comment utiliser le protocole de gestion du réseau, elles ne le camouflent pas. En effet, l'application de gestion doit connaître la nomenclature des objets dans le protocole de gestion pour être capable de les contrôler.

Passerelles

Certains de ces systèmes utilisent des passerelles pour le passage des requêtes au format IDL en requête au format du protocole de gestion installé sur le noeud géré du réseau. Les passerelles sont aussi responsable de recevoir les notifications émises par les agents du protocole de gestion et de les transmettre aux bons objets de gestion dans les applications. Selon le type d'approche utilisée pour la représentation des objets dans le système de gestion, les passerelles nécessaires n'ont pas toutes la même architecture.

Les passerelles qui sont les plus répandues sont celles pour la première approche statique. La raison probable est que les gens pensent que c'est la bonne façon de procéder puisqu'un groupe de travail est derrière cette méthode. Cependant, ce sera probablement différent d'ici quelques années. De plus, les passerelles de ce type sont très flexibles car elles peuvent être modifiées de manière automatisée lorsque de nouveaux objets sont ajoutés au protocole de gestion (grâce à l'algorithme de traduction de JIDM). CORBA/CMIP Gateway [21], OpenSnmp [6], la passerelle utilisée dans [48] et la passerelle définie dans [52] sont des passerelles de ce type.

Les passerelles qui permettent d'implanter la deuxième approche statique sont moins répandues car leur développement demande plus d'effort et elles sont moins flexibles (l'ajout d'un nouvel objet dans le protocole de gestion nécessite qu'un programmeur

modifie manuellement la passerelle). Par contre, le modèle objet des systèmes de gestion de réseaux implantés avec la deuxième approche statique et ce genre de passerelle représentent mieux le réseau et est beaucoup plus compréhensible. ProSphere [40] est un système de gestion de réseaux qui utilise une passerelle de ce type.

GOM est une passerelle composée d'un modèle objet dynamique comprenant des objets proxy et adaptateurs ainsi qu'une banque d'interfaces étendue [28, 15]. Les objets proxy sont les objets génériques de la première approche dynamique. Les adaptateurs servent de pont entre le modèle objet CORBA et celui du protocole de gestion requis. La banque d'interfaces contient des informations pour chaque objet, entre autres quel adaptateur utiliser pour la conversion des requêtes dans le bon protocole de gestion. Cette passerelle ne fait qu'étendre l'architecture de CORBA.

Pour la deuxième approche dynamique, il y a quelques passerelles disponibles. Liaison [28, 27], jSNMP Enterprise [7] et celle décrite dans [54] en sont des exemples. La plupart de ces passerelles sont composées d'un serveur proxy et d'une ou plusieurs sous-passerelles. Le serveur proxy reçoit une requête en format IDL, détermine sur quelle sous-passerelle elle doit être transmise et l'envoi à la sous-passerelle. La sous-passerelle s'occupe de transformer la requête IDL en requête du protocole de gestion.

2.3.2 Apport de CORBA à la gestion de réseaux

CORBA, grâce à son architecture et aux nombreux services qu'elle fournit, simplifie le développement d'applications de gestion de réseaux. CORBA permet aussi d'ajouter facilement de nouvelles fonctionnalités dans les systèmes de gestion de réseaux.

Parmi les services de CORBA qui peuvent servir dans les applications de gestion, on peut penser aux services de nommage, d'événements et de sécurité. Le service de nommage permet de donner un nom à une référence d'objet et de résoudre le nom plus tard pour obtenir la référence d'objet. Cela offre la possibilité de donner des noms contextuels aux références d'objet dans les applications et aussi de définir un arbre de noms dans le

domaine CORBA pour identifier les objets dans le domaine du protocole de gestion [52].

Avec le service d'événements, les événements (traps ou notifications) générés par les éléments du réseau peuvent être reçus par le ORB et livrés aux applications de gestion [40, 54] de manière uniforme, peu importe le protocole de gestion de réseaux qui en est l'auteur. De plus, ce service rend possible la coopération entre objets puisqu'il permet la notification d'objet à objet. Enfin, le service de sécurité facilite l'administration de la sécurité du réseau puisqu'il supporte l'authentification, les listes de contrôle d'accès, la confidentialité et la certification [53].

Les fonctionnalités que CORBA permet d'ajouter dans les systèmes de gestion de réseaux sont entre autres l'accès multiple aux éléments du réseau géré [67], la possibilité de voir le réseau comme une base de données objet répartie [68] et l'ajout dynamique d'objets de gestion. Une façon de contrôler l'accès multiple sur les objets gérés est présentée dans [67], mais l'équivalent peut aussi être atteint en utilisant le service de contrôle de la concurrence de CORBA.

Par ailleurs, la possibilité de manipuler les objets gérés du réseau comme les objets d'une base de données objet peut faciliter grandement la gestion. En effet, les bases de données sont présentes dans les systèmes informatiques depuis tellement longtemps qu'à peu près n'importe quel informaticien est capable de les utiliser. Ainsi, avec cette approche, le développement d'applications de gestion ne nécessite pas de connaissance particulière de la part des programmeurs. Cela rend donc le développement de telles applications moins coûteux en temps, en ressources humaines et en argent. Les services de relations, de cycle de vie, d'interrogations, de transactions, de contrôle de la concurrence, de persistance et d'événements jouent un rôle important dans la réalisation du cadre d'application permettant de voir le réseau comme une base de données objet [68].

Finalement, CORBA permet l'ajout dynamique de nouveaux objets pour la surveillance d'un réseau. Nous présentons brièvement dans cette section deux méthodes pour ajouter de nouveaux objets.

Première méthode Prenons le cas du protocole de gestion SNMP. Normalement, lorsqu'on veut ajouter un nouvel objet, il faut ajouter l'objet dans la MIB et modifier l'agent dans certain cas. Cela est très fastidieux et désagréable car il faut recompiler la MIB et l'agent, puis tester l'agent à nouveau pour voir si tout fonctionne bien, sans impact sur les services déjà fournis par l'agent.

La façon de procéder avec CORBA pour éviter ces inconvénients est d'abord de mettre des objets CORBA dans le ORB pour chaque objet SNMP, comme le suggère JIDM. Chaque objet sait à quel agent SNMP s'adresser pour obtenir ou modifier l'information qu'il représente. Lorsqu'on veut ajouter un nouvel objet on peut procéder de deux façons. On peut d'abord modifier la MIB pour ajouter un nouveau segment qui définit le nouvel objet et ajouter un nouvel agent SNMP sur le périphérique qui traite le nouveau segment de MIB ajouté. Puisqu'on ne modifie pas l'agent existant, on a besoin de tester que le nouveau service fourni par l'agent ajouté. Ensuite, on ajoute un nouvel objet CORBA dans le ORB qui représente le nouvel objet SNMP ajouté et qui sert seulement de passerelle. L'autre façon consiste tout simplement à créer un objet CORBA qui s'occupe de fournir la fonctionnalité désirée en interrogeant l'agent SNMP. De cette façon, la fonctionnalité pour l'objet est implantée dans l'objet CORBA plutôt que dans l'agent SNMP. Toutefois, cette façon de faire peut engendrer beaucoup de trafic sur le réseau pour maintenir l'objet à jour tout dépendant du type de fonctionnalité qu'il fournit. Dans les deux façons, les applications de gestion peuvent découvrir dynamiquement les objets ajoutés au moyen du service de vendeur (trader).

Cette méthode pourrait même être étendue pour permettre aux clients SNMP d'accéder les nouveaux objets CORBA en utilisant un agent proxy et un service de pages jaunes. Les clients SNMP pourraient envoyer des requêtes SNMP à l'agent proxy. Ce dernier en utilisant un service de pages jaunes pourrait déterminer à quel objet CORBA la requête est destinée et lui envoyer (en format IDL). Que l'objet CORBA soit une passerelle ou qu'il implante directement une fonctionnalité ne ferait aucune différence.

Deuxième méthode Ce qui peut être intéressant, c'est de permettre à une application de gestion de créer dynamiquement un objet pour compiler une statistique qu'elle désire, et même de recevoir un avertissement si cette statistique atteint une certaine valeur. La méthode précédente ne permet pas cela car elle nécessite que quelqu'un programme l'objet ajouté et voir même modifie la MIB et ajoute un agent. La méthode qu'on définit ici permet de créer dynamiquement ce genre d'objet.

La méthode consiste à utiliser des objets génériques préalablement inscrits dans un ORB pour définir de nouveaux objets. Plusieurs objets génériques différents peuvent être créés pour représenter chacun un modèle de statistiques différent. Lorsqu'une application veut créer un objet pour compiler une statistique précise, elle n'a qu'à créer une instance de l'objet générique qui implante le modèle de la statistique. Ensuite, l'application utilise les services de cet objet pour indiquer les variables qui doivent servir aux calculs de la statistique, la méthode du calcul de la statistique et même le seuil de sévérité de cette statistique pour que l'objet génère une notification. Cependant, encore une fois, cette méthode peut générer beaucoup de trafic sur le réseau. Comme on peut le constater, cette méthode permet de créer dynamiquement de nouveaux objets. Ces nouveaux objets peuvent même être inscrits dans le ORB pour que d'autres applications puissent les utiliser.

2.3.3 Avantages de CORBA pour la gestion de réseaux

La section suivante résume les avantages offerts par CORBA présentés dans les sections précédentes ainsi que d'autres bénéfices. En premier lieu, l'utilisation de CORBA est un avantage pour la la gestion de réseaux car il permet d'intégrer les différentes normes de gestion existantes sous une interface uniforme [40] et de migrer facilement les systèmes de gestion de réseaux vers la technologie de répartition d'objets [50]. En second lieu, CORBA permet de créer des cadres d'applications extensibles [40]. Ajouter un nouvel élément dans le réseau ou ajouter/modifier des propriétés pour un équipement de réseau

consiste seulement à ajouter de nouveaux objets ou à modifier des objets existants. Cela peut permettre de changer de technologie de réseau facilement. En dernier lieu, CORBA permet de développer des solutions expansibles [40]. Un système de gestion de réseaux peut avoir à manipuler un grand nombre d'objets. La répartition des objets permet de ne pas causer de problèmes de performance, de limiter les pertes d'information et de ne pas placer trop de demandes de gestion sur les éléments du réseau.

On peut ajouter à cette liste d'avantages ceux que CORBA offre à tous les types d'applications. Tout d'abord, CORBA permet de créer des applications clients de petite taille. Le code de l'application est découplé du code de l'interface utilisateur [40]. Les applications clients peuvent être encore plus petites en utilisant une approche dynamique d'association d'objets. Ensuite, CORBA permet d'utiliser plusieurs services déjà définis pour les environnements répartis. Enfin, les autres avantages que CORBA apporte est qu'il permet d'écrire les applications de gestion dans n'importe quel langage de programmation [40, 59], qu'il est très répandu donc les programmeurs qui connaissent le IDL sont faciles à trouver [59], que les outils de développement évoluent rapidement et ne sont pas trop coûteux [59] et qu'il permet de faire une gestion basée sur le Web [28, 40, 54].

2.3.4 Désavantages de CORBA pour la gestion de réseaux

Malgré tous les avantages qu'elle offre, l'architecture de CORBA a tout de même quelques désavantages pour la gestion de réseaux. Tout d'abord, son protocole de communication orienté connexion est moins léger en utilisation de largeur de bande que SNMP ou CMIP [40]. Par contre, selon le type de système de gestion de réseaux qu'on implante, on peut inverser la situation. En effet, il est possible de regrouper des requêtes s'adressant à plusieurs agents différents dans une seule requête jusqu'à une certaine destination (par exemple un proxy) et de les envoyer ensuite une par une aux agents concernés. On peut par la suite faire la même chose, des agents au client, avec les résultats des requêtes. Ainsi, on peut économiser de la largeur de bande pour une grande partie de la communication

sur le réseau.

Un autre désavantage est la fréquence d'invocation des primitives [59]. La surveillance d'un réseau nécessite l'obtention continuelle d'information significative. Comme CORBA est un paradigme client-serveur, cela peut générer beaucoup de trafic sur le réseau.

2.3.5 Solution idéale

Nous pensons que la solution idéale serait que chaque périphérique possède un ORB. Le ORB pourrait contenir divers objets qui permettent de gérer le périphérique. Pour ajouter un nouveau service ou une nouvelle information, il suffirait de modifier un objet existant ou d'ajouter un nouvel objet dans le ORB. Grâce au service de vendeur, il serait possible de découvrir dynamiquement ces objets et de les utiliser. Les objets logeant directement sur le périphérique, il n'y aurait que de petits messages qui seraient échangés sur le réseau, c'est-à-dire l'appel d'une méthode et le résultat de l'exécution de la méthode.

Cependant, les équipements de réseau (e.g. hub, routeur) ne possèdent pas assez de mémoire et de processeur assez puissant pour que ce soit possible pour l'instant. Par contre, avec l'évolution de la technologie et la baisse des prix du matériel, de plus en plus de fournisseurs améliorent leurs équipements sur ce point. Donc, cette solution sera probablement réalisable dans un avenir prochain.

D'autres solutions, autres que CORBA, sont disponibles pour la gestion de réseaux. Certaines de ces solutions n'ont pas les désavantages de CORBA, mais elles n'ont pas tous ses avantages non plus. Ces solutions sont présentées dans la prochaine section.

2.4 Autres approches apparentes à CORBA pour la gestion de réseaux

Les approches qui sont apparentes à CORBA pour la gestion de réseaux sont celles qui permettent d'ajouter dynamiquement de nouveaux services ou fonctionnalités. Ces approches ne doivent pas nécessiter la modification des agents existants sur les périphériques à gérer. Les seules autres approches qui permettent cela, selon nos connaissances actuelles du domaine, sont celles qui sont basées sur le paradigme de code mobile.

Le paradigme de code mobile consiste à déplacer un programme à l'endroit où il doit être exécuté. Il existe trois types différents de code mobile [13] : le code sur demande, l'évaluation distante et les agents mobiles. Dans le code sur demande, un client possède les ressources nécessaires pour réaliser un service, mais ne possède pas le code pour le service. Le client doit demander à un serveur qu'il lui envoie le code nécessaire pour réaliser le service. L'évaluation distante est exactement le contraire du code sur demande. Le client possède le code pour un service mais les ressources nécessaires se trouvent sur le serveur. Dans ce cas-ci, le client envoie le code nécessaire chez le serveur pour qu'il l'exécute. Quant aux agents mobiles, ce sont des programmes autonomes qui implantent les services et qui sont capable de se déplacer, pendant leur exécution, sur un autre noeud du réseau et de continuer leur exécution. Donc, les agents mobiles possèdent le code pour réaliser un service mais ne possèdent pas les ressources.

Deux approches de code mobile sont de plus en plus populaires dans le domaine de la gestion de réseaux, il s'agit de la gestion par délégation et des agents mobiles.

2.4.1 Gestion par délégation

Le terme gestion par délégation est apparu au début des années 90 avec Yemini et Goldszmidt [35]. Depuis ce temps, plusieurs autres prototypes de recherche basés sur la

gestion par délégation ont fait leur apparition [66, 75], dont quelques-uns sont décrits dans [66].

La gestion par délégation est basée sur l'approche d'évaluation distante. Elle consiste à envoyer des fonctions ou un ensemble de commandes, rassemblées dans un programme, sur l'élément du réseau pour qu'elles soient exécutées. Chaque périphérique du réseau doit supporter un processus élastique [35] qui s'occupe de recevoir les agents délégués (programmes), de débuter et de contrôler leur exécution. Les agents délégués peuvent utiliser des services fournis par le serveur de délégation (processus élastique) pour accéder les ressources du périphérique ou directement interagir avec l'agent du protocole de gestion (e.g., SNMP).

Une telle exécution est asynchrone, donc elle permet à une station de gestion d'effectuer d'autres tâches pendant que l'agent délégué effectue son travail. Comme on peut le constater, cette approche donne plus de flexibilité puisque de nouveaux services peuvent être envoyés au serveur de délégation en tout temps. Le trafic sur le réseau et la largeur de bande utilisée sont réduits puisqu'un long message est transmis plutôt que plusieurs messages courts. De plus, une certaine autonomie peut être atteinte car un agent délégué peut être exécuté pour diagnostiquer un problème et reconfigurer le périphérique pour le rendre dans un état normal, même si la connexion avec la station de gestion est interrompue.

2.4.2 Les agents mobiles

Comme nous l'avons déjà mentionné, les agents mobiles sont des programmes autonomes qui peuvent se déplacer de noeud en noeud et continuer leur exécution. Ils peuvent donc effectuer des tâches qui demandent une distribution de la charge de travail ou qui nécessitent des ressources disponibles sur différents noeuds du réseau. L'utilisation des agents mobiles pour la gestion de réseaux est étudiée par plusieurs groupes de recherche, dont ceux de l'Université de Carleton [17], de l'INRIA-IRISA [62] et de GMD

FOKUS [74].

Les agents mobiles sont intéressants pour la gestion de réseaux car ils peuvent être autonomes, réactifs aux changements dans l'environnement, prendre des initiatives et communiquer avec d'autres agents et ressources. De plus, ils offrent certains avantages considérables dans la gestion des réseaux [17] :

- ils permettent de réaliser des économies d'espace puisqu'un agent réside seulement sur un noeud à la fois, donc les fonctionnalités qu'il offre ne sont pas dupliquées ;
- ils permettent de réduire le trafic du réseau ;
- ils ont une interaction asynchrone autonome avec la station de gestion, donc ils peuvent effectuer des tâches même si la station de gestion n'est pas active ;
- ils peuvent supporter des environnements hétérogènes ;
- ils permettent d'étendre les services offerts et de faire des mises à jour dynamiquement.

Les agents mobiles sont conçus particulièrement pour réaliser des services qui nécessitent des ressources de plusieurs noeuds différents. Ainsi, on peut donner un itinéraire à un agent mobile et ensuite le déployer sur le réseau. L'agent se déplacera sur chacun des noeuds de son itinéraire pour exécuter ses tâches et ensuite il pourra revenir au départ pour délivrer les résultats ou demeurer sur le réseau pour continuer son travail. De ce fait, les agents mobiles peuvent être utilisés pour la collecte d'information de plusieurs éléments du réseau. Par exemple, on peut les utiliser pour la modélisation de réseaux, la gestion des fautes ou la gestion de la performance [17]. De plus, les agents mobiles peuvent être dotés d'une certaine intelligence, ce qui leur permet d'accumuler des connaissances et de réutiliser des fonctionnalités acquises pour diagnostiquer les problèmes et apporter des solutions de façon autonome [74].

L'autonomie et l'ajout dynamique de services sont surtout ce qui rend les agents mobiles avantageux pour la gestion de réseaux. Toutefois, les agents mobiles ne sont pas appropriés pour toutes les situations [62]. Ils sont surtout convenables pour la réalisation

de tâches à long terme. Des facteurs comme le nombre de noeuds impliqués, les protocoles, la nature des tâches ont également une influence sur l'utilisation des agents mobiles versus l'approche client-serveur traditionnelle. Il existe des modèles de calcul de trafic qui permettent d'évaluer qu'elle approche il serait préférable d'utiliser dans les applications de gestion de réseaux [14].

Pour qu'un agent mobile puisse se déplacer et s'exécuter sur un périphérique, ce dernier doit posséder un support logiciel qu'on appelle environnement d'agents mobiles. Cet environnement fournit la capacité d'envoyer, de recevoir et d'héberger des agents mobiles. Il permet également aux agents d'accéder les ressources du périphérique et offre aussi d'autres services. Tout comme CORBA, le manque de ressources matérielles sur les périphériques de réseau est un obstacle majeur pour l'exploitation des agents mobiles pour la gestion de réseaux.

Du fait que les agents mobiles offrent des avantages dans certains cas et que dans d'autres l'approche client-serveur est préférable, et aussi que les environnements d'agents mobiles doivent offrir des services aux agents, on peut constater qu'il y a une intersection entre les agents mobiles et CORBA. En effet, CORBA offre déjà la plupart des services que les environnements d'agents mobiles ont besoin et son architecture est déjà adaptée pour les systèmes répartis. De plus, CORBA est une norme adoptée par l'industrie alors qu'il n'existe aucune norme pour les agents mobiles. Par conséquent, une combinaison des agents mobiles et de CORBA serait vraiment profitable. D'ailleurs, l'OMG effectue des travaux dans le domaine des agents mobiles et une ébauche de norme a été émise pour discussion.

2.4.3 Normalisation

L'IETF et l'ISO avec l'ITU ont formé des groupes de travail, chacun de leur côté, pour établir une norme qui permettrait de déléguer des scripts de gestion dans leur architecture de gestion respective (SNMP et CMIS/CMIP). Aucune norme n'est encore disponible,

ce sont seulement des propositions pour l'instant.

IETF

DISMAN, le groupe de travail de l'IETF, a défini une architecture de gestion répartie qui est utilisée avec l'architecture de gestion de réseaux SNMP. Cette architecture, basée sur la gestion par délégation, étend l'architecture de SNMP en ajoutant des services qui sont nécessaires dans un environnement de gestion distribué. Ces services, définis dans le document de l'architecture de gestion répartie [16], sont :

- le service des systèmes connus,
- le service des domaines de gestion,
- le service des opérations de gestion cibles,
- le service d'identification des délégations,
- le service de contrôle de délégation,
- le service de planification,
- le service de notification fiable,
- le service de destination des notifications.

Pour rendre possible la réception et l'exécution de scripts, une MIB Script à été définie [51]. Cette MIB est utilisée pour déléguer des fonctions de gestion à un agent SNMP. Les fonctions de gestion sont définies comme un script dans un langage de gestion. Le langage utilisé n'a pas d'importance et même du code natif compilé peut être distribué si l'implantation est capable de l'exécuter.

La MIB Script définit une interface standard, pour la délégation des fonctions de gestion, qui fournit les capacités de :

- transférer des scripts de gestion à un gestionnaire réparti (agent SNMP) ;
- initialiser, suspendre, reprendre et terminer l'exécution des scripts de gestion ;
- transférer les arguments pour les scripts de gestion ;

- surveiller et contrôler les scripts de gestion en cours d'exécution ;
- transférer les résultats produit par des scripts de gestion.

Cette interface est composée d'objets qui se retrouvent dans les groupes suivants :

- smLanguageGroup qui fournit les informations sur les langages et les extensions de langages supportées par l'implantation de la MIB Script ;
- SmScriptGroup qui consiste en une table qui contient tous les scripts connus par l'implantation de la MIB Script. Les objets de la table permettent de télécharger un script à partir d'un URL, lire un script, emmagasiner un script, détruire un script, lister les scripts permanents, ainsi que lire et modifier le statut d'un script ;
- SmCodeGroup qui fournit les facilités pour transférer et modifier des scripts via la commande SNMP SET ;
- SmLaunchGroup qui permet d'associer un script avec un propriétaire et de démarrer un script. Un script est associé avec un propriétaire afin de déterminer les droits du script ;
- SmRunGroup qui permet de lister les scripts en exécution ou qui ont été exécutés récemment.

ISO/ITU

L'ISO et l'ITU veulent aussi étendre leur architecture de gestion en permettant la délégation de scripts de gestion. Les extensions requises doivent permettre de satisfaire environ les mêmes besoins que celles de l'IETF. Les besoins sont séparés en deux catégories : les besoins des usagers et les besoins opérationnels [66]. Les besoins des usagers sont satisfaits directement par la gestion par délégation. Les besoins opérationnels sont définis comme étant la possibilité de :

- pouvoir planifier ou reporter l'exécution d'un script ;
- être capable de modifier la requête pour une exécution planifiée ou retardée ;

- être capable d'initialiser, suspendre, reprendre et terminer un script ;
- être capable de fournir une signalisation et un enregistrement ;
- envoyer des notifications lorsqu'un changement d'état survient.

Leur solution consiste à un objet ordonnanceur de commandes, un objet thread, un objet lanceur de script et un objet bloc de lancement [66]. Ces objets coopèrent ensemble pour combler les besoins opérationnels et ils permettent d'exécuter les scripts délégués. Cette approche pour implanter la gestion par délégation dans la norme de gestion OSI utilise les fonctions de gestion existantes. Les scripts délégués doivent être écrits dans le nouveau langage SMSL.

Chapitre 3

Réalisation d'un logiciel de gestion et localisation de services

Ce chapitre présente le système de localisation de services que nous avons réalisé. Pour décrire son architecture et son comportement, nous utilisons la méthodologie orientée objet Unified Modeling Language (UML) [61]. Dans un premier temps, nous introduisons une application typique que nous avons créée pour démontrer la mise en pratique de notre système. Dans un autre temps, nous décrivons le système de localisation de services selon les étapes de conceptualisation, d'analyse, de conception, d'implantation et d'évaluation qui constituent le développement d'un logiciel.

L'application que nous avons créée pour expérimenter notre système est un bloc note à partir duquel il est possible d'imprimer (figure 5). Une fenêtre d'impression est accessible via le menu de l'application (figure 6) pour choisir l'imprimante (figure 7). Les imprimantes présentes dans la liste combinée sont celles obtenues lors de demandes de localisation de services précédentes. Lorsqu'une imprimante est choisie, ses caractéristiques sont affichées. Pour spécifier des caractéristiques d'imprimantes désirées, une fenêtre de configuration est disponible en cliquant sur le bouton *Propriétés*. Il est possible d'obtenir ou de mettre à jour la liste des imprimantes du réseau pour le service d'impression désiré,

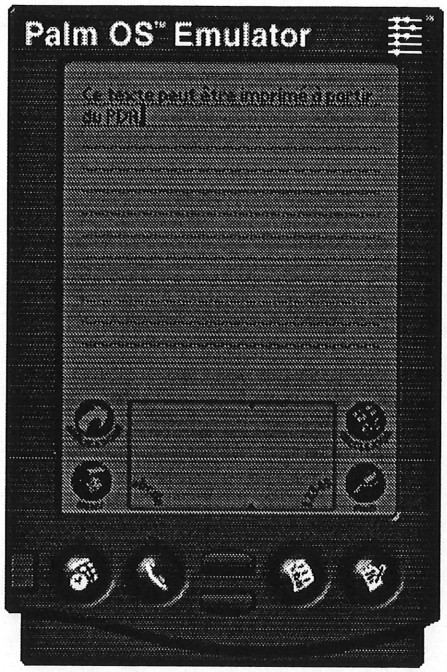

FIG. 5 – *Bloc note*

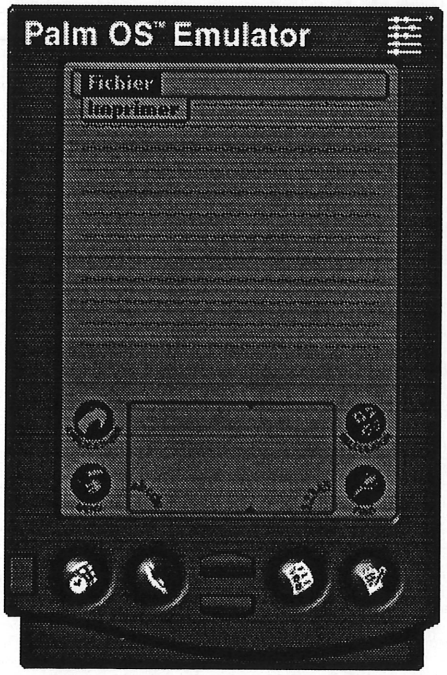

FIG. 6 – *Menu de l'application bloc note*

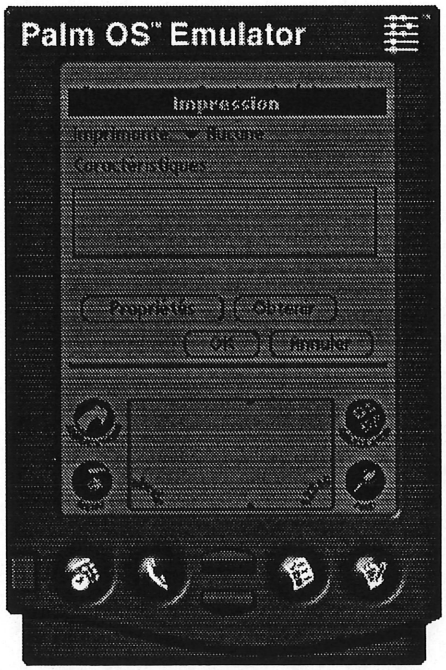

FIG. 7 – *Fenêtre d'impression*

directement à partir de l'application, en cliquant sur le bouton *Obtenir* de la fenêtre d'impression. À ce moment, la localisation du service d'impression est effectuée sur le réseau afin d'obtenir les ressources de type imprimante fournissant le service. Enfin, l'utilisateur peut lancer l'impression du contenu du bloc note en cliquant sur le bouton *OK* ou quitter l'écran d'impression sans imprimer en cliquant sur le bouton *Annuler*.

La fenêtre de spécification des propriétés d'imprimante désirées comporte deux attributs (figure 8) : le langage et la taille du papier. L'utilisateur peut ne pas choisir de valeur spécifique pour l'une ou l'autre des propriétés en sélectionnant *Quelconque* dans les listes de sélection. Le bouton *OK* permet de sauvegarder la spécification et de retourner à la fenêtre précédente, alors que le bouton *Annuler* permet de retourner à la fenêtre précédente sans sauvegarder les informations contenues dans cette fenêtre.

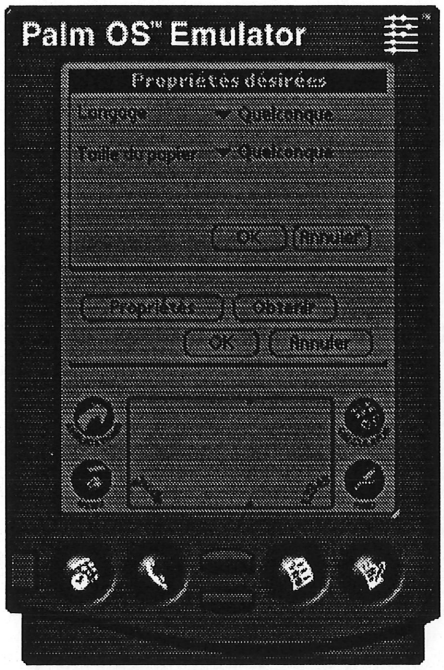

FIG. 8 – *Propriétés d'imprimantes désirées*

3.1 Conceptualisation

La localisation des services d'un réseau sur un ANP implique qu'un certain nombre de fonctionnalités soit effectuées autant du côté de l'administrateur de réseau (figure 9) que du côté de l'utilisateur de l'appareil (figure 10).

D'une part, des services doivent être accessibles aux utilisateurs. Pour ce faire, l'administrateur doit utiliser une application lui permettant de définir des ressources qui offriront certains services. Cette application doit permettre d'associer des propriétés à une ressource afin de décrire les services qu'elle offre. De plus, un serveur de localisation de ressources doit aussi être en fonction sur un nœud du réseau pour recevoir les requêtes de services, rechercher les ressources appropriées et retourner la liste des ressources aux clients.

D'autre part, les applications qui désirent spécifier et localiser des services doivent

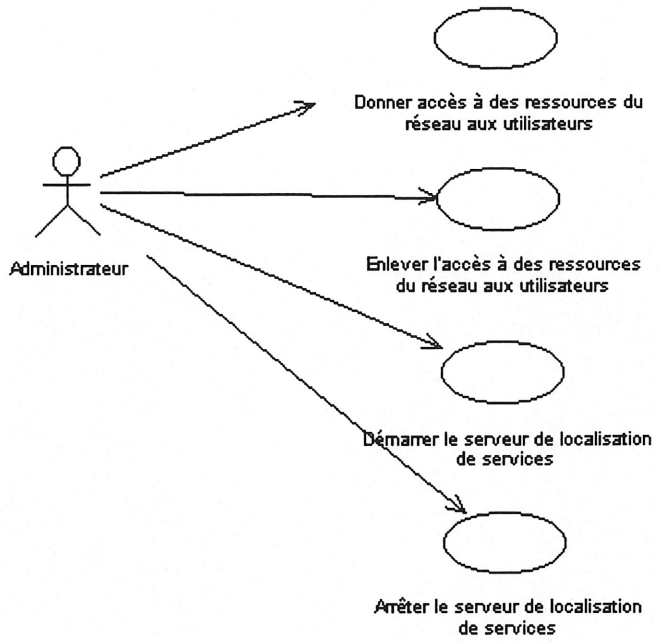

FIG. 9 – *Tâches effectuées par l'administrateur de réseau*

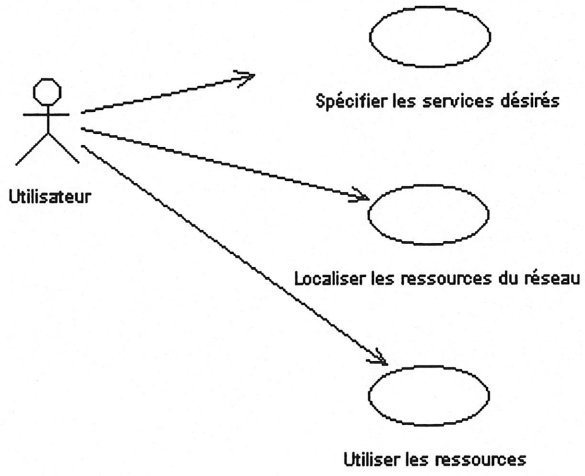

FIG. 10 – *Tâches effectuées par l'utilisateur de l'assistant numérique personnel*

avoir à leur portée une API ou un cadre d'application leur permettant de réaliser ces opérations facilement. Aussi, afin de rendre la localisation de services optimale pour l'utilisateur des applications, un moyen peut lui être fourni pour localiser les services globalement plutôt que de procéder application par application.

Afin de bien représenter les différentes fonctionnalités mentionnées, les deux scénarios suivants décrivent les opérations effectuées par les deux partis. Le premier scénario montre les opérations effectuées par l'administrateur de réseau pour rendre des ressources accessibles aux utilisateurs. L'administrateur démarre d'abord une application qui lui permet de réaliser toutes ses tâches (figure 11).

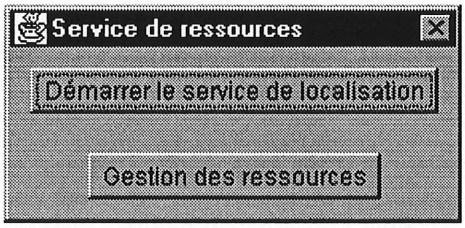

FIG. 11 – *Application de gestion du service de localisation de ressources*

Pour définir les ressources à rendre accessibles aux utilisateurs, l'administrateur utilise la fenêtre de gestion des ressources de l'application. Cette fenêtre peut être ouverte en cliquant sur le bouton *Gestion des ressources*. La fenêtre de gestion des ressources possède une série de boutons qui permettent à l'administrateur d'ajouter, de modifier ou de supprimer des ressources (figure 12). À mesure que des ressources sont définies, elles apparaissent dans la liste de la fenêtre. Il est possible d'avoir un aperçu rapide des ressources définies en visualisant le contenu de la liste. Chaque élément de la liste représente une ressource et les informations affichées sont le nom, le service offert, le URL et les propriétés de la ressource.

Pour ajouter ou modifier des ressources, l'administrateur utilise la fenêtre d'édition de ressources (figure 13). Lors d'un ajout, cette fenêtre apparaît avec seulement les champs correspondant aux informations communes à tous les types de ressource. Le nom de la

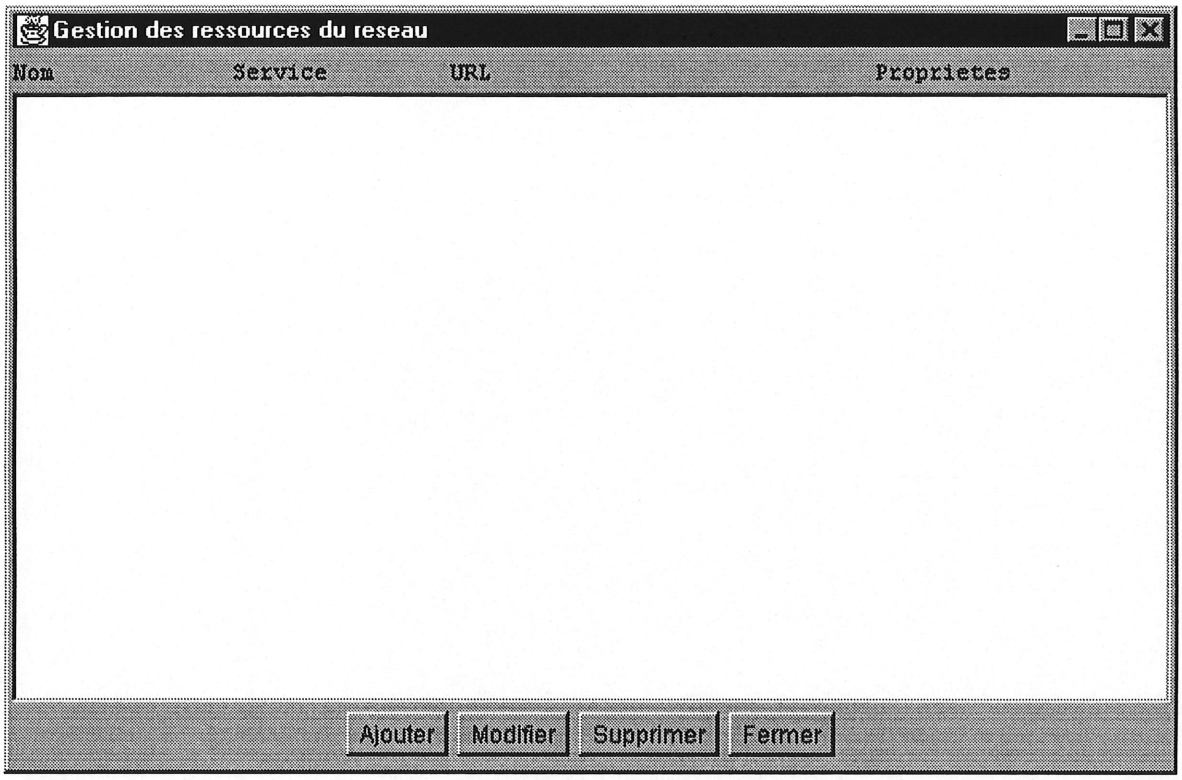

FIG. 12 – *Gestion des ressources*

ressource est celui qui sera employé par les utilisateurs pour obtenir directement un accès à la ressource. Le service fourni par la ressource peut être indiqué en choisissant parmi ceux qui sont présents dans la liste combinée. Il est également possible de préciser un service autre que ceux-ci en choisissant *Autre* dans la liste et en inscrivant le nom du service dans le champ à droite de la liste. Lorsqu'un service prédéfini est sélectionné dans la liste, des champs d'informations propres à ce service apparaissent dans la fenêtre. Par exemple, si on prend le cas du service d'imprimante, une liste combinée apparaît pour permettre à l'administrateur d'associer une imprimante du réseau à la ressource qu'il définit (figure 14). Le champ URL, bien que pouvant être défini pour tous les types de ressources, est optionnel. Enfin, il est possible de spécifier les propriétés de la ressource en choisissant leur nom dans la liste combinée de propriétés ou en l'inscrivant directement

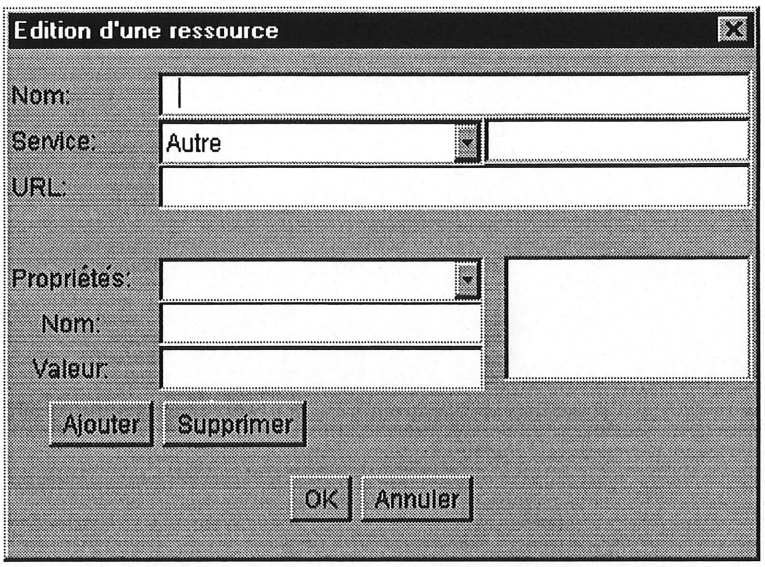

FIG. 13 – *Édition d'une ressource*

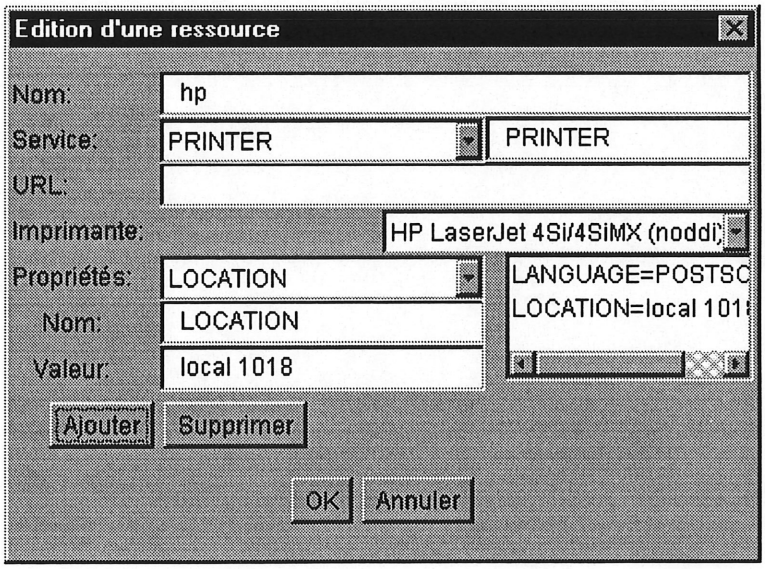

FIG. 14 – *Édition d'une ressource avec le service imprimante*

dans le champ nom de la propriété. Une propriété peut aussi avoir une certaine valeur qui peut être inscrite dans le champ valeur de la propriété. Pour chaque propriété à définir, il suffit de cliquer sur le bouton *Ajouter*, après avoir indiqué son nom et sa valeur, pour qu'elle apparaisse dans la liste de défilement de propriétés. Une propriété peut être détruite en la choisissant dans la liste de défilement et en cliquant sur le bouton *Supprimer*. Les propriétés décrivant les ressources sont utilisées lors de la localisation de services pour retrouver les ressources qui correspondent aux services demandés par les utilisateurs.

Enfin, la suppression d'une ressource s'effectue en choisissant cette dernière dans la liste de la fenêtre de gestion des ressources et en cliquant sur le bouton *Supprimer*. La figure 15 montre un exemple de plusieurs ressources qui ont été définies.

Pour terminer, l'administrateur démarre le service de localisation de services en cliquant sur le bouton *Démarrer le service de localisation* de la fenêtre principale de l'application (figure 11). À ce moment, la nouvelle fonctionnalité de ce bouton est d'arrêter le service si l'administrateur clique sur celui-ci à nouveau (figure 16).

Le deuxième scénario évoque les opérations réalisées par l'utilisateur pour spécifier, découvrir et utiliser les ressources du réseau à partir d'un ANP. Dans un premier temps, l'utilisateur doit spécifier les propriétés pour les services qu'il désire utiliser dans ses applications. Cette étape se fait à partir de chacune des applications utilisant des services du réseau. Par exemple, dans l'application bloc note, l'utilisateur accède à la fenêtre de propriétés d'imprimantes pour spécifier les propriétés désirées pour l'imprimante sur laquelle il veut imprimer ses textes (figure 17). Les propriétés spécifiées limitent l'étendu du domaine sous lequel s'effectue la sélection d'imprimantes. Normalement, l'étape de spécification des propriétés des services est effectuée lors de la première utilisation d'une application. Par la suite, ces mêmes propriétés seront réutilisées pour la découverte de ressources. Cependant, l'utilisateur peut modifier les propriétés pour les services utilisés par les applications aussi souvent qu'il le désire.

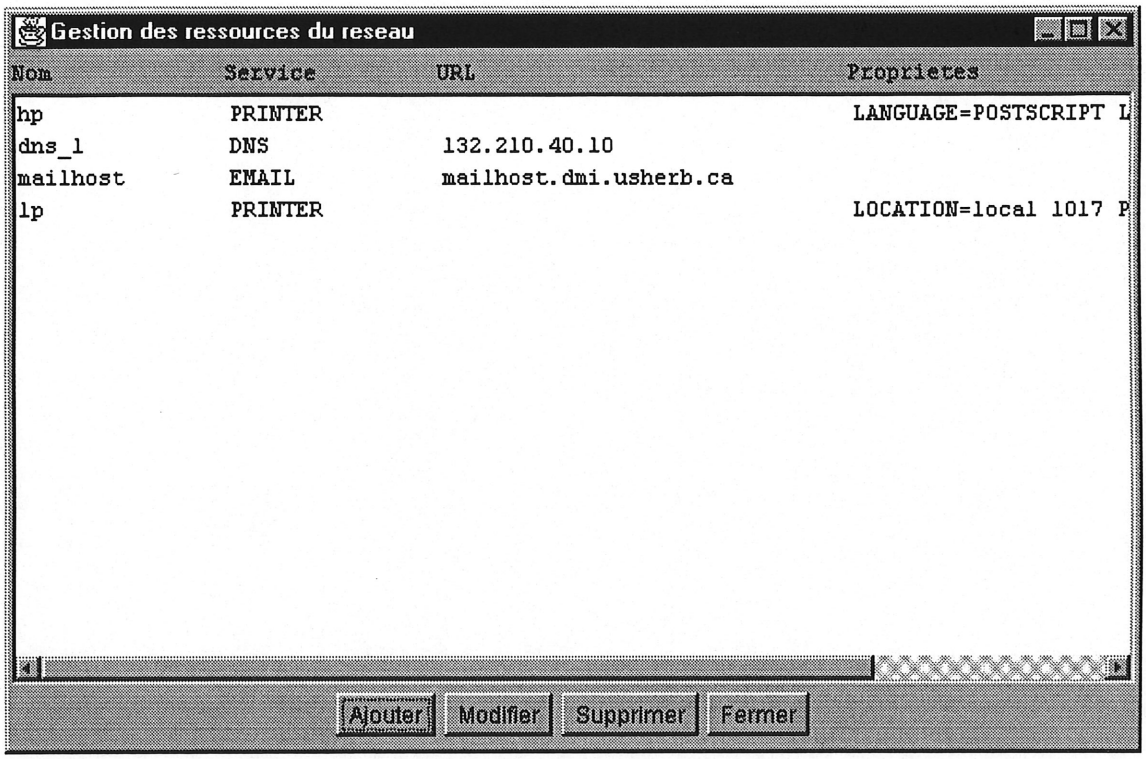

FIG. 15 – *Ressources définies et accessibles aux utilisateurs d'assistants numériques personnels*

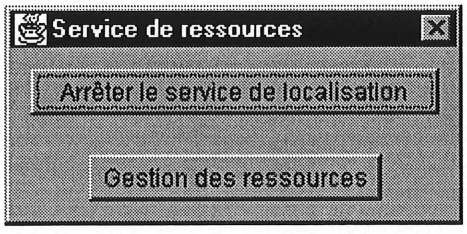

FIG. 16 – *Le service de localisation de services est démarré*

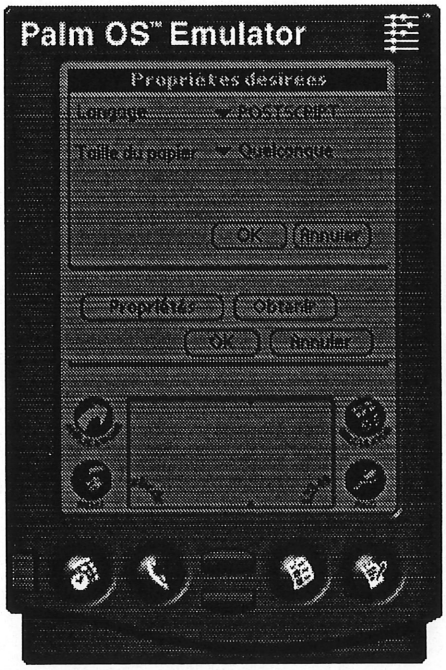

FIG. 17 – *Spécification des propriétés désirées pour une imprimante*

Dans un deuxième temps, l'utilisateur demande à découvrir les ressources réalisant les services qu'il a spécifiés. La localisation des ressources est effectuée seulement dans deux circonstances. Le premier cas est lorsque l'utilisateur se connecte à un réseau pour lequel il n'a jamais découvert les ressources qui accomplissent les services dont il a de besoin. Le deuxième cas se présente lorsque l'utilisateur essai d'utiliser une ressource qu'il a déjà utilisée mais qui n'est plus active.

Afin de localiser ou d'utiliser les ressources, l'utilisateur doit indiquer au service de localisation l'adresse du serveur de localisation de services du réseau. Cette opération est effectuée à l'aide de l'application de localisation des services (figure 18). Un champ est disponible dans la fenêtre de cette application pour indiquer l'adresse du serveur de localisation de services du réseau. En entrant l'adresse IP du serveur et en cliquant sur le bouton *Ajouter*, l'adresse du serveur est ajoutée dans la liste combinée et elle y demeurera présente lors des prochaines utilisations. Ainsi, lorsque l'utilisateur se reconnectera

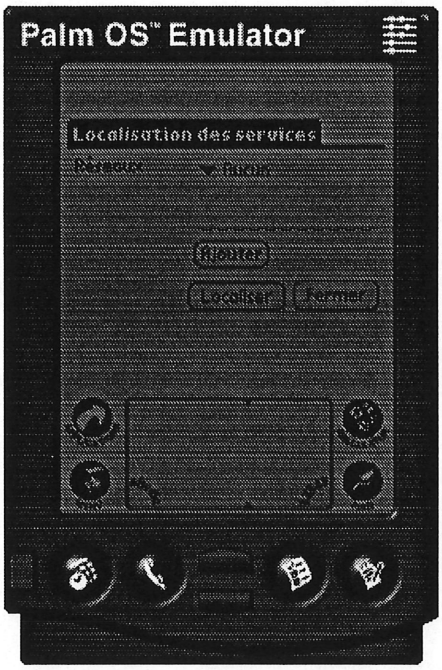

FIG. 18 – *Application de localisation de services*

de nouveau à ce réseau, il pourra tout simplement choisir l'adresse IP dans la liste. Lorsqu'une adresse de serveur de localisation de services est choisie, toutes les applications utilisant des services sont alors configurées pour s'adresser à ce serveur.

Enfin, pour localiser les ressources, l'utilisateur n'a qu'à cliquer sur le bouton *Localiser* pour démarrer la recherche de services. L'utilisateur peut également quitter l'application de localisation des services en cliquant sur le bouton *Annuler* et lancer la localisation de services à partir d'une application si cette dernière offre cette possibilité. Par exemple, dans l'application bloc note, l'utilisateur peut cliquer sur le bouton *Obtenir* de la fenêtre d'impression pour découvrir les imprimantes désirées. L'application de localisation de services découvre les ressources pour toutes les applications qui sont installées sur l'assistant numérique personnel (figures 19 et 20).

Dans un dernier temps, l'utilisateur peut utiliser avec ses applications les services

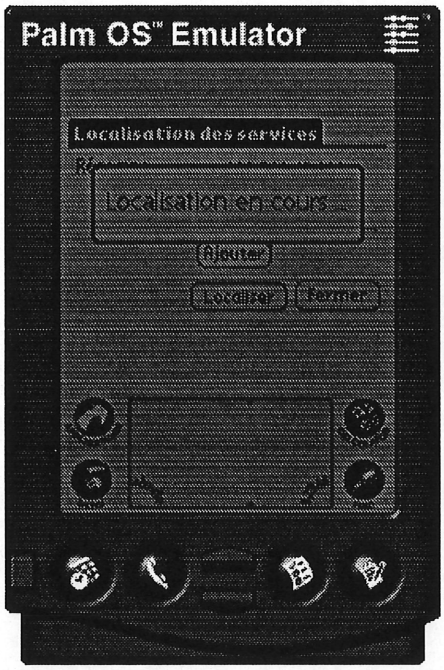

FIG. 19 – *Localisation de services en cours*

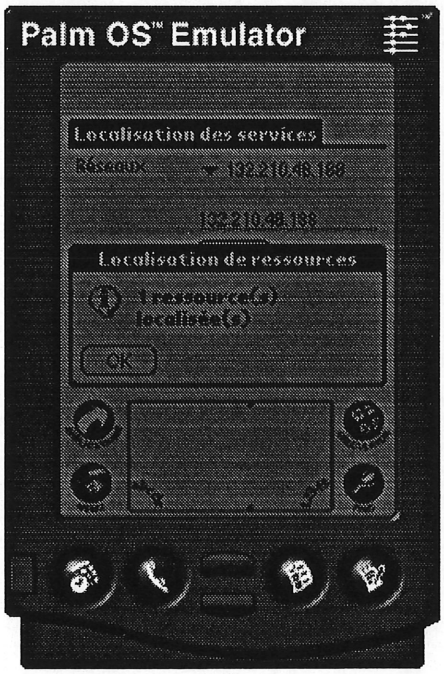

FIG. 20 – *Localisation de services terminée*

des ressources découvertes. Si on revient à l'exemple de l'application bloc note, l'utilisateur n'a qu'à choisir une des imprimantes présentes dans la liste combinée et cliquer sur le bouton *OK* pour imprimer (figure 21). La liste contient toutes les imprimantes découvertes lors de la dernière localisation de services effectuée sur le réseau courant.

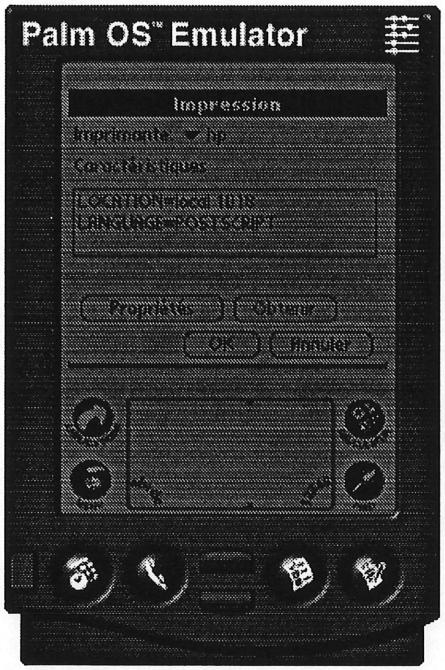

FIG. 21 – *Sélection d'une imprimante pour imprimer*

3.2 Analyse

Notre logiciel de gestion et de localisation de services comprend une partie client et une partie serveur. La figure 22 montre une vue globale simplifiée de la structure de notre logiciel au moyen d'abstractions. La partie client comprend les abstractions *Application client* et *Localisation de services*. L'abstraction *Application client* représente les applications qui désirent utiliser des ressources d'un réseau. Elle spécifie les services qu'elle désire se voir offrir par les ressources et les communique à l'abstraction *Localisation de*

services qui est responsable de localiser les services pour toutes les applications et de leur retourner les ressources correspondantes. L'abstraction *Application client* peut également entreprendre de localiser les ressources elle-même pour les services qu'elle désire. Pour localiser et utiliser les ressources, les abstractions *Application client* et *Localisation de services* utilisent les services du *Service de localisation de ressources* de la partie serveur via le réseau TCP/IP. Les ressources connues du *Service de localisation de ressources* ont dû être définies auparavant à l'aide du module de *Gestion de ressources* de l'application serveur.

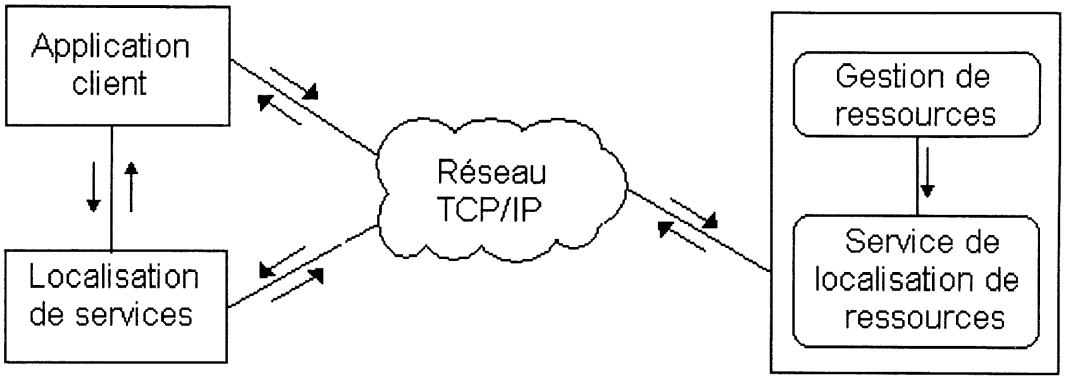

FIG. 22 – *Abstraction du logiciel de gestion et localisation de services*

3.2.1 Spécification des classes

Des vues plus détaillées des parties serveur et client du logiciel sont présentées par les diagrammes de classes modélisant les différentes abstractions. La figure 23 montre le diagramme de classes modélisant la partie serveur du logiciel. La classe *MainWindow* représente la fenêtre principale de l'application d'administration du service de localisation de ressources. Elle permet d'accéder à la fenêtre de gestion de ressources, représentée par la classe *ResourceManagementWindow*, ainsi que de contrôler le serveur de localisation de ressources modélisé par la classe *ResourceLocator*.

La classe *ResourceManagementWindow* est l'interface qui permet à l'administrateur

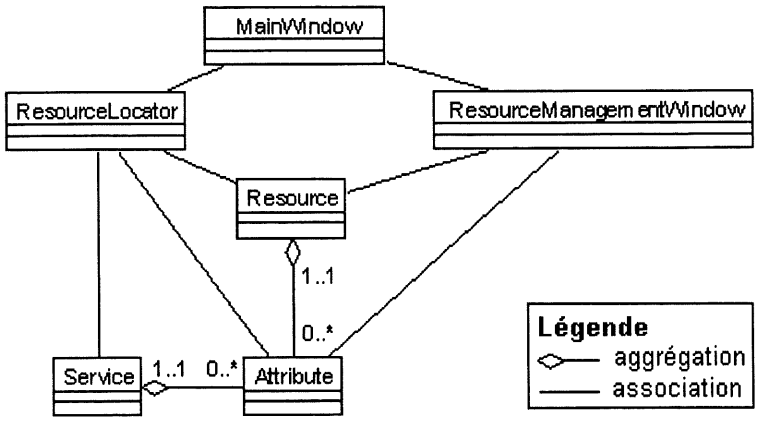

FIG. 23 – *Diagramme de classes de la partie serveur*

du réseau d'ajouter, de modifier et de supprimer des ressources accessibles par les utilisateurs du réseau. La classe *Service* est utilisée par la partie client pour spécifier les services désirés au *ResourceLocator*. La classe *Resource* modélisent les ressources du réseau. La classe *Attribute* sert à définir les propriétés des ressources et des services. Les interactions entre l'administrateur et les différentes classes pour l'ajout d'une ressource sont symbolisées par le diagramme de séquences de la figure 24. Pour ajouter une ressource, l'administrateur clique sur le bouton approprié de la fenêtre *ResourceManagementWindow* (l'événement est pris en charge par l'action add resource). L'administrateur fournit les informations pour la ressource à créer telles que son nom, le service qu'elle accomplit, d'autres informations propres à la ressource et les attributs caractérisant la ressource qui seront utilisés lors de la localisation de ressources. À partir de ces informations, l'instance *ResourceManagementWindow* crée la ressource ainsi que les attributs (les actions create) qu'elle lui associe par la suite (à l'aide de l'action add attribute).

La modification et la suppression d'une ressource sont relativement simples. Pour chacune de ces fonctionnalités, l'administrateur indique le nom de la ressource ainsi que le service qu'elle offre (il peut exister plusieurs ressources avec le même nom mais offrant des services différents). L'instance de *ResourceManagementWindow* retrouve ensuite la

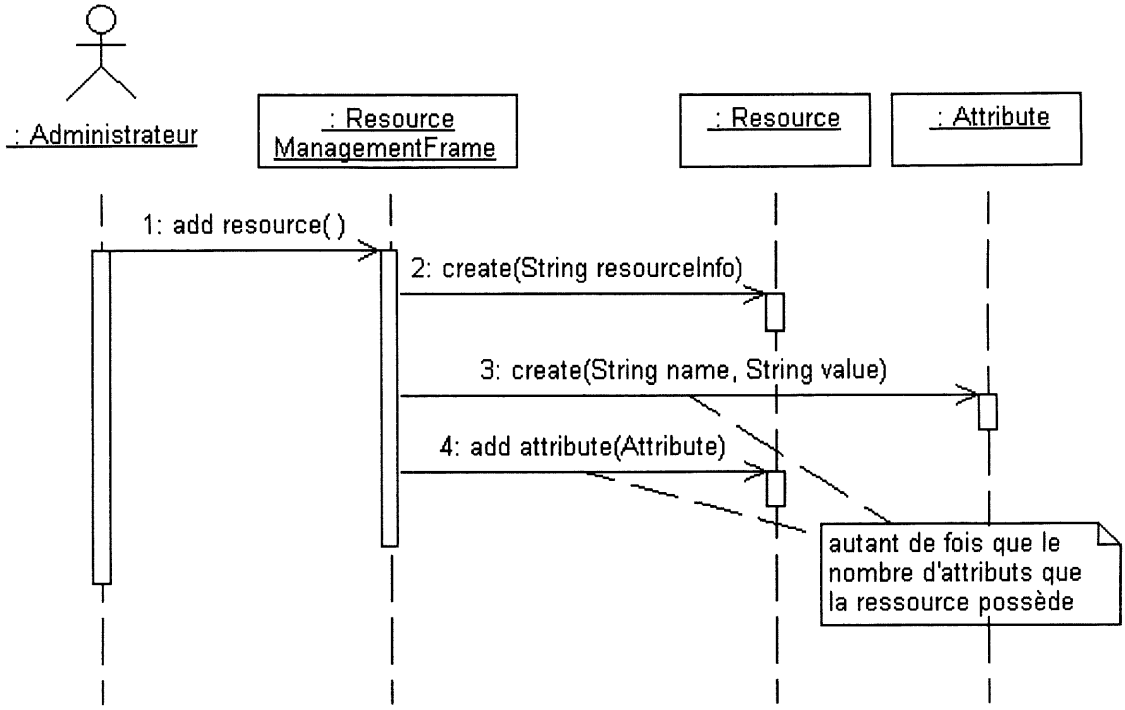

FIG. 24 – *Ajout d'une ressource*

ressource concernée. Dans le cas d'une modification, les informations de la ressource sont présentées à l'administrateur qui peut alors les modifier. Par contre, dans le cas d'une suppression, l'objet de type *Resource* est détruit. L'objet de type *Resource* est responsable de détruire les objets de type *Attribute* qu'il possède.

La classe *ResourceLocator* représente le serveur de localisation de ressources. Pour que les ressources puissent être localisées, elles doivent être enregistrées auprès de l'objet *ResourceLocator* (l'action register dans le diagramme de la figure 25). C'est l'instance de *MainWindow* qui est responsable d'effectuer cette tâche avant de démarrer le *ResourceLocator*. Le démarrage et l'arrêt sont les deux opérations de contrôle sur le *ResourceLocator* accessible par *MainWindow*. Le démarrage du *ResourceLocator* (action start) est représenté à la figure 25. L'arrêt consiste seulement à l'envoi d'un message d'arrêt à l'instance *ResourceLocator* par la fenêtre *MainWindow* lorsque l'administrateur clique

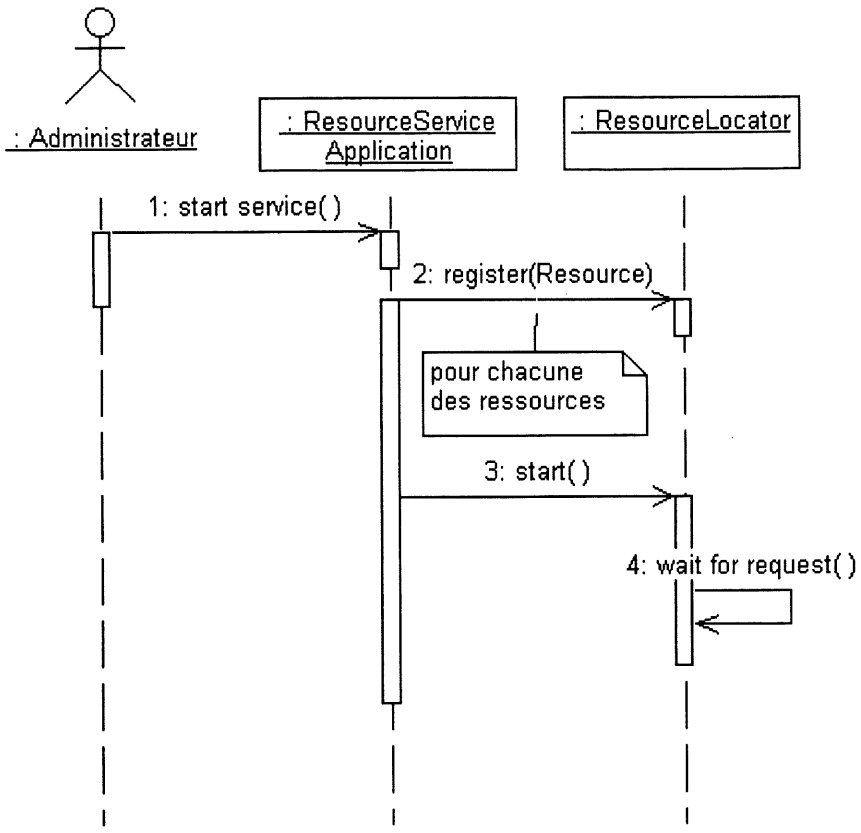

FIG. 25 – *Démarrage du service de localisation de ressources*

sur le bouton correspondant à cette action.

Afin de mieux comprendre la localisation de ressources, cette dernière est expliquée plus loin dans la description de la partie client du logiciel.

La figure 26 présente le diagramme de classes de la partie client du logiciel. La classe *ClientApp* représente les applications qui spécifient et utilisent les services du réseau. Elle peut également permettre la localisation des ressources pour les services qu'elle veut utiliser. La classe *ServiceLocationAppWindow* représente la fenêtre de l'application qui sert à localiser tous les services spécifiés par les *ClientApp*. Les instances de ces deux classes utilisent une instance de la classe *ResourceLocator* pour retrouver les ressources offrant les services désirés. Les classes *Service*, *Resource* et *Attribute* ont les mêmes significations

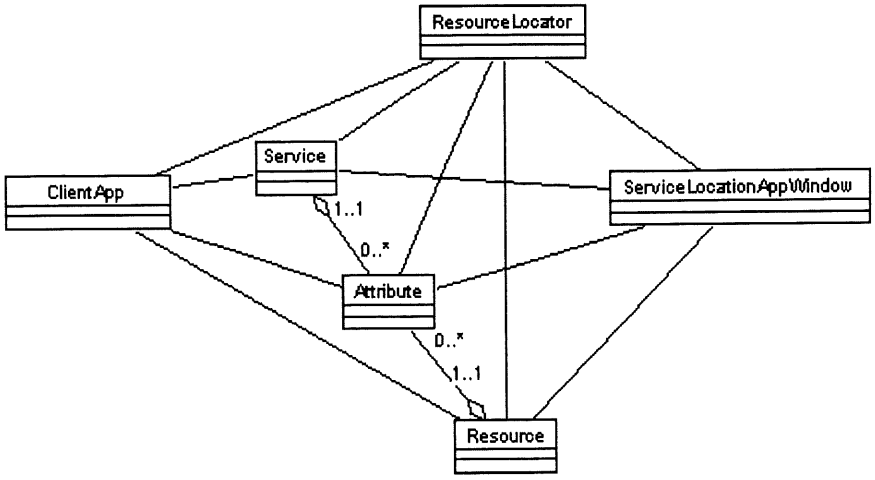

FIG. 26 – *Diagramme de classes de la partie client*

que celles décrites pour la figure 25.

Le déroulement de la spécification de services, de la localisation et de l'utilisation de ressources est représenté par le diagramme de séquences de la figure 27. Pour spécifier les services désirés (*Service*) pour une application, l'utilisateur indique les propriétés (*Attribute*) pour les services à l'aide d'une fenêtre disponible dans *ClientApp*. Une instance de type *ClientApp* construit les instances de *Service* qui seront envoyées dans les requêtes à l'objet de type *ResourceLocator*. L'utilisateur démarre ensuite la localisation de services en cliquant sur un bouton disponible dans l'application. À ce moment, des requêtes sont envoyées au *ResourceLocator*. Une requête est envoyée pour chaque *Service*. Le *ResourceLocator* compare les attributs (*Attribute*) de chaque *Resource* qu'il connaît avec les attributs (*Attribute*) du *Service*. Le *ResourceLocator* retourne la liste des instances de type *Resource* qui réalisent le service demandé s'il en existe. Une application de type *ClientApp* peut par la suite utiliser les objets *Resource* à la demande de l'utilisateur.

En ce qui concerne la classe *ServiceLocationAppWindow*, elle ne fait qu'initier la localisation de services comme la classe *ClientApp* le fait, sauf qu'elle le fait pour les services désirés par toutes les instances de type *ClientApp*.

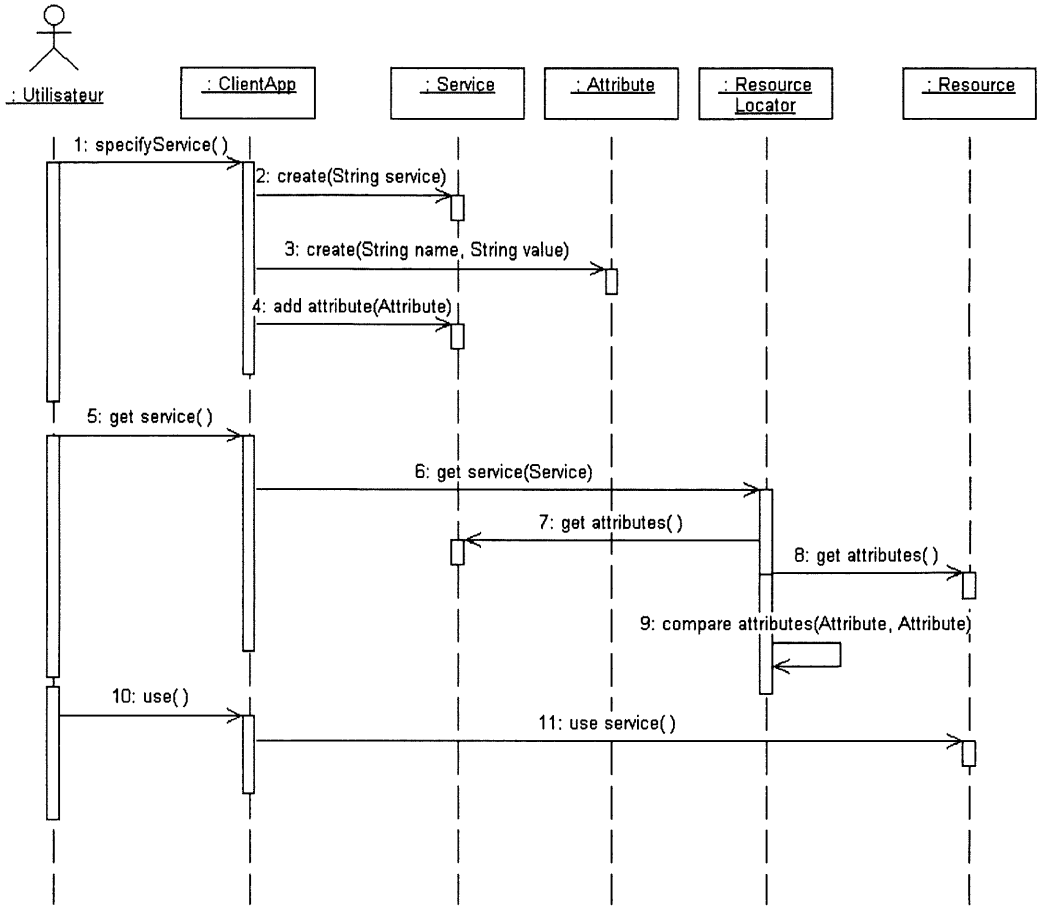

FIG. 27 – *Spécification, localisation et utilisation de services*

3.2.2 Plate-forme physique

Le logiciel nécessite un ANP Palm Pilot avec PalmOS 2.0 ou plus pour la partie client et un PC compatible IBM avec Windows 95/98/NT ainsi qu'une machine virtuelle Java pour la partie serveur. Le ORB Visibroker pour Java doit aussi être installé sur la machine serveur. Une machine d'une autre architecture physique et un autre système d'exploitation peuvent également être utilisés pour le serveur, pourvu que le ORB Visibroker pour Java puisse y être installé. L'ANP peut se connecter à n'importe quel ordinateur du réseau pour accéder à ce dernier, pourvu qu'il exécute un service d'accès distant.

3.3 Conception

3.3.1 Description simplifiée de l'architecture du système

Lorsqu'un utilisateur se connecte à un réseau, il peut désirer retrouver les services pour toutes les applications qu'il utilise. Notre logiciel permet à l'utilisateur de retrouver les services utilisés par toutes les applications en une seule opération à l'aide de l'application de localisation de services. C'est aussi à partir de cette application que l'utilisateur indique l'adresse IP du serveur de localisation de ressources d'un réseau. Une base de données est utilisée pour l'échange d'information entre les applications client et l'application de localisation de services. L'architecture complète de notre système prend également en considération que nous exploitons les composants répartis avec l'architecture de CORBA. Une vue simplifiée de l'architecture du logiciel est présentée à la figure 28.

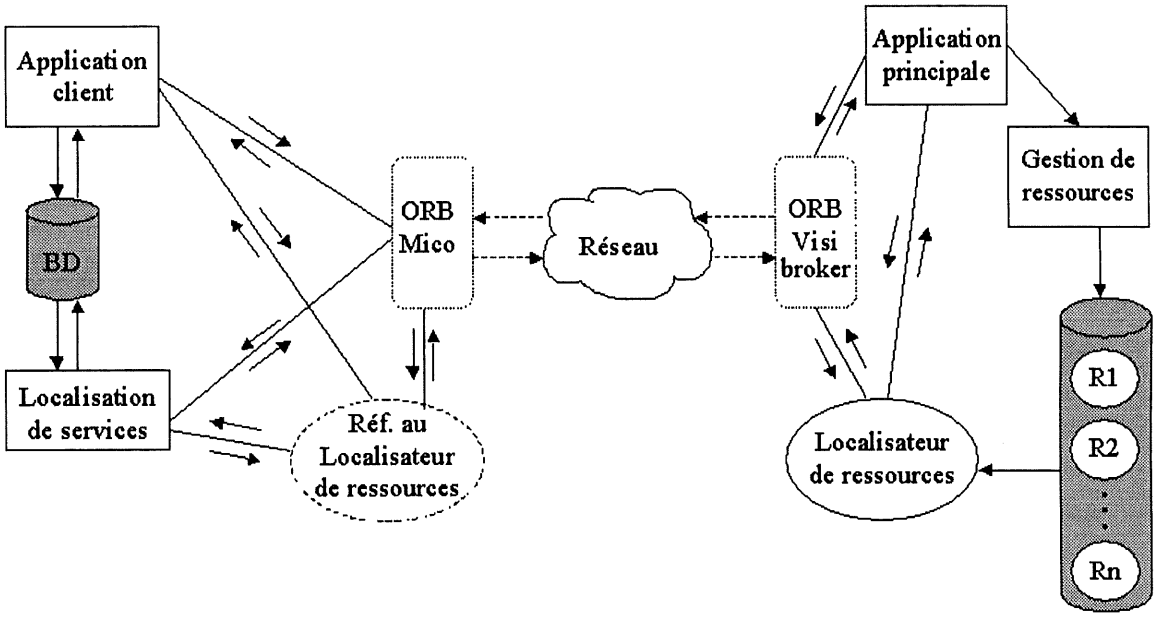

FIG. 28 – *Architecture simplifiée du logiciel*

L'application principale permet à l'administrateur de réseau de gérer des ressources. C'est à partir de cette dernière que l'administrateur peut créer, modifier et supprimer les

ressources accessibles aux utilisateurs. Ce sont des composants CORBA qui représentent les véritables ressources matérielles ou logicielles qui sont créées. Ces composants permettent aux utilisateurs de faire appel aux services des ressources qu'ils représentent. À leur création, les composants sont sérialisés et entreposés dans un répertoire de la machine serveur (R1,R2,...,Rn).

L'application principale permet aussi à l'administrateur de démarrer et d'arrêter le service de localisation de ressources. Au moment du démarrage, l'application crée une instance du localisateur de ressources. Elle lui demande aussi de charger en mémoire et d'inscrire dans son service toutes les ressources qui ont été créées. Ensuite, l'application enregistre le localisateur de ressources auprès d'un serveur de noms CORBA. Le localisateur de ressources devient alors accessible aux clients. L'échange de requêtes et de réponses entre les clients et le localisateur de ressources se fait par l'entremise du ORB présent sur la machine de chacun des partis.

Du côté client, les applications client enregistrent les services spécifiés par l'utilisateur dans une base de données. C'est à partir de cette base de données que l'application de localisation de services peut connaître tous les services à rechercher. C'est aussi dans cette base de données que sont inscrites toutes les ressources retrouvées afin que les applications client puissent en prendre connaissance et les utiliser. L'application client et l'application de localisation de services utilisent le ORB pour obtenir une référence au localisateur de ressources. Par la suite, elles utilisent cette référence pour envoyer des requêtes de localisation de services ou pour obtenir des références à des ressources connues.

3.3.2 Description détaillée de l'architecture

L'architecture détaillée de la partie serveur est modélisée par le diagramme de classes de la figure 29. Certaines classes présentes dans l'analyse ont été éclatées en plusieurs classes. La nomenclature suit des conventions normalement employées en Java. De nouvelles classes sont également présentes pour modéliser la persistance des données et l'usage

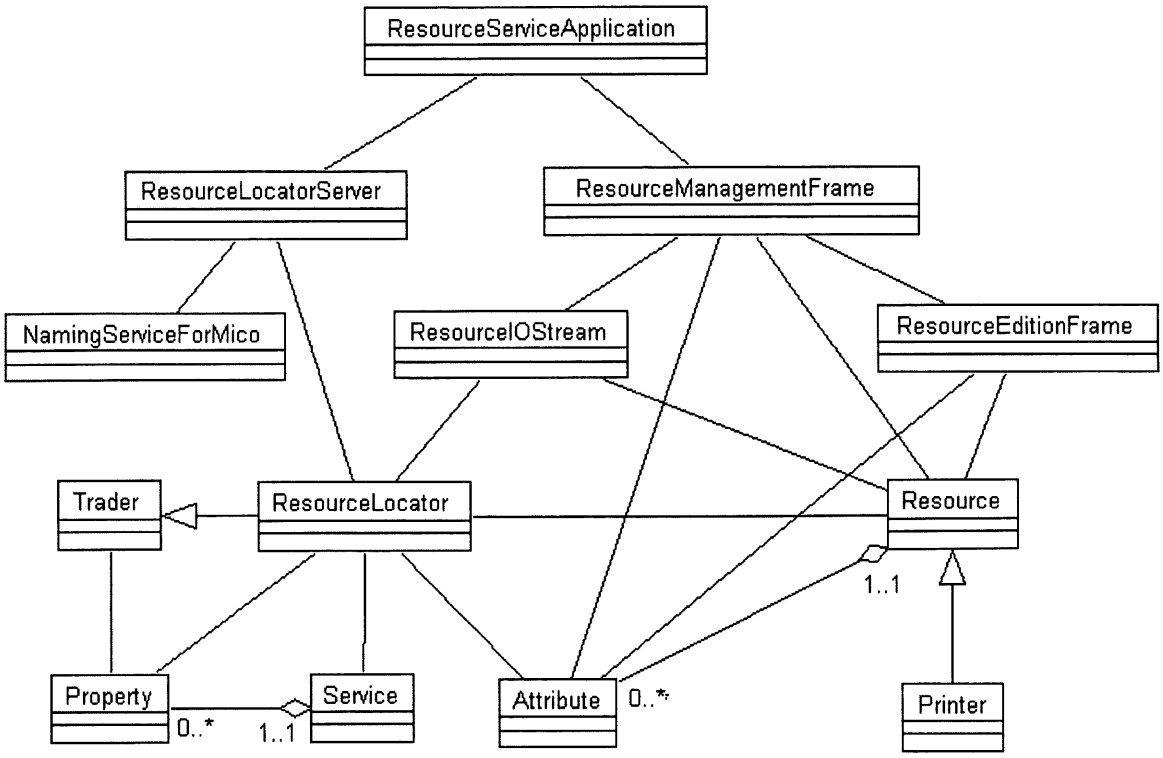

FIG. 29 – *Architecture détaillée de la partie serveur*

de l'architecture de composants répartis. Les classes *ResourceLocator*, *Resource*, *Attribute*, *Service*, *Property* et *Trader* sont des composants répartis CORBA.

La classe *ResourceServiceApplication* représente l'application du service de localisation de ressources qui était représentée par *MainWindow* dans l'analyse. Elle sert à accéder à la fenêtre de gestion de ressources représentée par la classe *ResourceManagementFrame* (*ResourceManagementWindow* dans l'analyse) et à contrôler le serveur de localisation de ressources représenté par la classe *ResourceLocatorServer*.

Le rôle de la classe *ResourceManagementFrame* est de présenter à l'administrateur la liste des ressources définies. Elle lui permet aussi d'ajouter, de modifier et de supprimer des ressources. Lorsque l'administrateur veut ajouter ou modifier une ressource, elle lui affiche une instance de la classe *ResourceEditionFrame* qui lui permet de fournir les informations pour la ressource ainsi que de définir ses caractéristiques (*Attribute*).

Pour entreposer les ressources, le *ResourceManagementFrame* utilise une instance de la classe *ResourceIOStream*. Cette classe encapsule le moyen utilisé pour la persistance des données. Elle permet d'ajouter et de détruire un objet *Resource*, ainsi que d'obtenir la liste des objets *Resource*.

Lorsque le *ResourceLocatorServer* est démarré, il crée une instance de la classe *ResourceLocator*. Cette classe est un composant de l'architecture de composants répartis qui permet aux clients d'utiliser le service de localisation de ressources. La classe *ResourceLocator* est une spécialisation de la classe *Trader* qui représente le service d'annuaire de type page jaune de CORBA. Le *Trader* permet d'enregistrer des objets ainsi que leurs propriétés (*Property*) afin qu'ils puissent être retrouvés par des clients en fonction de leurs propriétés. La classe *ResourceLocator* spécialise ce service en restreignant l'enregistrement d'objets à ceux de type *Resource* seulement.

Après la création de l'instance de la classe *ResourceLocator*, l'instance de la classe *ResourceLocatorServer* demande à cette dernière de créer et d'inscrire dans son service toutes les ressources définies par l'administrateur. Le *ResourceLocator* retrouve ces ressources à l'aide du *ResourceIOStream*. Ensuite, le *ResourceLocatorServer* enregistre le composant *ResourceLocator* auprès de l'objet *NamingServiceForMico*, qu'il instancie lui-même auparavant, afin qu'il soit accessible par tous les clients qui veulent l'utiliser à partir d'un ANP. Une seule instance de *NamingServiceForMico* doit exister sur une machine (sous forme de processus). Le patron de conception *Singleton* [30] est appliqué pour obtenir cette instance. La classe *NamingServiceForMico* est nécessaire puisque Mico est seulement une implantation du protocole IIOP et il ne possède pas de véritable service de noms compatible à celui de CORBA. Le service de type page blanche réalisé par *NamingServiceForMico* est offert sur le port TCP 20000. La figure 30 montre le comportement de la classe *NamingServiceForMico*.

La classe *Printer* présente dans le diagramme de la figure 29 illustre que tous les types de ressources possibles doivent spécialiser la classe *Resource*.

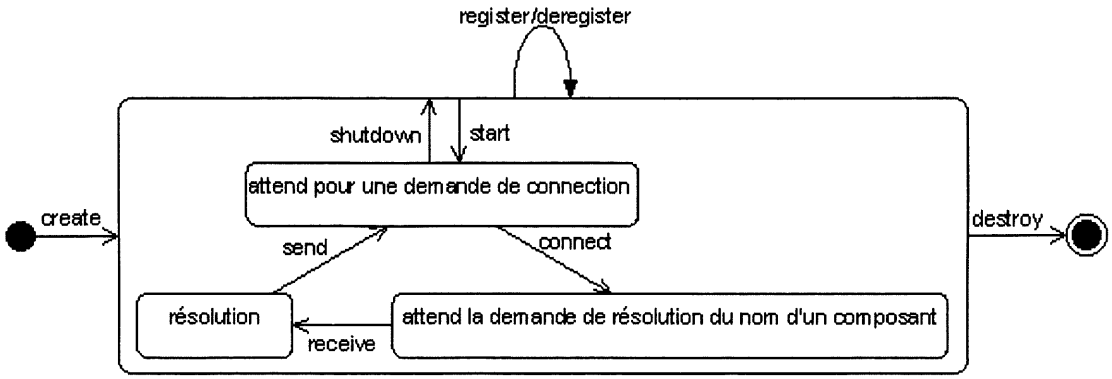

FIG. 30 – *Comportement de la classe NamingServiceForMico*

L'architecture détaillée de la partie client est présentée sur deux figures afin d'améliorer la lisibilité à cause du grand nombre de liens entre les classes. Les classes *_st_ResourceLocator*, *_st_Resource* et *_st_Attribute* sont les souches (stubs) qui permettent de communiquer avec les composants de type *ResourceLocator*, *Resource* et *Attribute* sur un serveur. Le préfixe _st_ est ajouté devant le nom des classes pour indiquer que ce ne sont pas des composants mais seulement leur souche. Parce que le ORB Mico qui est utilisé pour le développement de la partie client n'est pas une implantation complète d'un ORB mais seulement une implantation du protocole IIOP, certaines classes supplémentaires doivent être conçues. Ces classes sont *ResourceSeq*, *AttributeSeq*, *ResourceHelper* et *AttributeHelper*. Les classes *ResourceSeq* et *AttributeSeq* permettent de manipuler des listes d'objets de type *_st_Resource* et *_st_Attribute* respectivement. Les classes *ResourceHelper* et *AttributeHelper* permettent aux classes qui les utilisent de recevoir et de construire une référence ou une liste de références d'objet de type *_st_Resource* et *_st_Attribute* respectivement.

La figure 31 montre le diagramme de classes illustrant la relation entre les classes utilisées par une application client qui se sert des services d'un réseau. La classe *ClientApp* représente une application qui utilise les services des ressources du réseau. Elle utilise la classe *ServiceManager* pour spécifier et conserver ses services dans le système ainsi

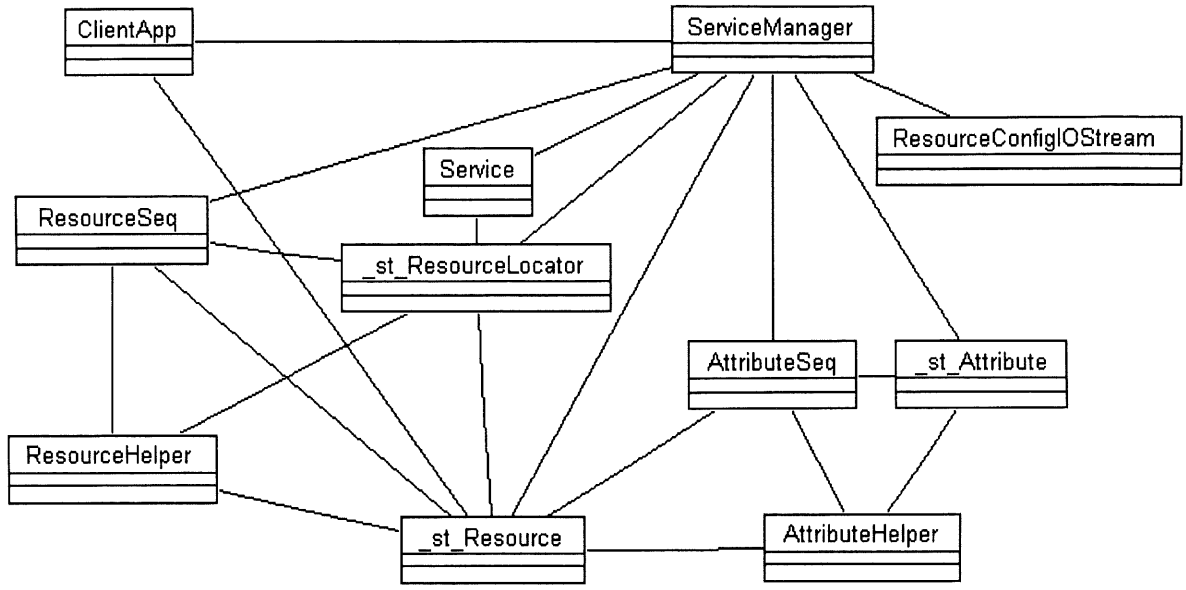

FIG. 31 – *Architecture détaillée de la partie client – application client*

que pour obtenir les ressources du réseau qui réalisent ces services. Chaque application s'identifie auprès du *ServiceManager* pour que ce dernier puisse gérer les informations pour leurs services et leurs ressources. La classe *ServiceManager* utilise la classe *ResourceConfigIOStream* pour enregistrer les services des applications. Lorsqu'une application demande au *ServiceManager* de lui donner la liste des ressources qui satisfont les services qu'elle a spécifiés, celui-ci envoie des requêtes pour des services (*Service*) au localisateur de ressources en utilisant la classe *_st_ResourceLocator*. Cette dernière retourne la liste des ressources retrouvées à l'aide d'une instance de la classe *ResourceSeq*. Le *ServiceManager* parcoure cette liste et enregistre les informations pour chacune des instances de *_st_Resource* qu'elle contient en utilisant une instance de *ResourceConfigIOStream*. Pour utiliser une ressource, le *ClientApp* demande la ressource qu'il désire au *ServiceManager* et celui-ci lui retourne l'instance de *_st_Resource* qui correspond à cette ressource. Le *ClientApp* utilise cette instance pour envoyer des requêtes à la ressource.

L'architecture détaillée de l'application de localisation de services est légèrement

différente. Le diagramme de la figure 32 illustre les relations entre les classes utilisées dans cette application. La classe *ServiceLocationAppWindow* représente l'application de localisation de services. Elle utilise une instance de la classe *ResourceConfigIOStream* pour obtenir la liste des services à localiser. Elle procède de la même façon que la classe *ServiceManager* pour obtenir les ressources du réseau et sauvegarder leurs informations nécessaires. Puisqu'elle n'utilise pas le *ServiceManager*, elle peut retrouver les ressources pour toutes les applications même si elle ne connaît pas ni leur nom ni le type de services qu'elles utilisent.

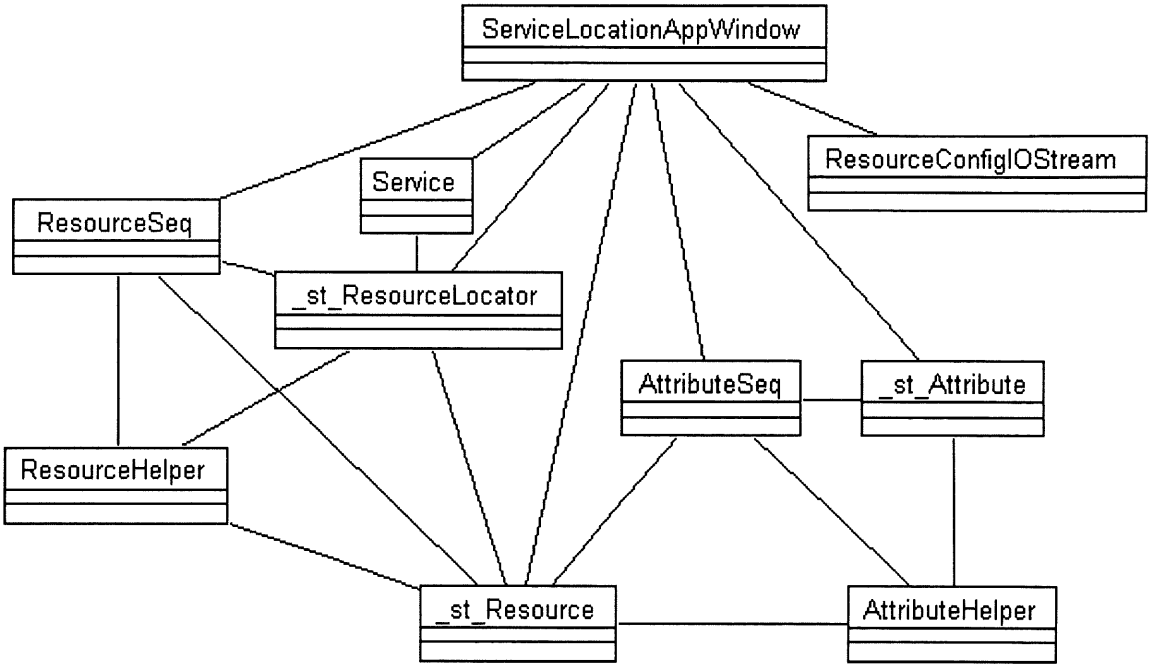

FIG. 32 – *Architecture détaillée de la partie client – application de localisation de services*

3.3.3 Obtention de l'adresse IP du serveur de localisation de ressources

Pour s'adresser au serveur de localisation de ressources en utilisant la souche *_st_ResourceLocator*, l'adresse IP du nœud du réseau qui exécute le serveur doit être connue. Le

ServiceManager ou l'application de localisation de services doit fournir cette adresse au moment de la création de l'instance de la souche. L'utilisateur du système spécifie cette adresse à partir de l'application de localisation de services. Un champ est disponible pour inscrire l'adresse. Chaque adresse est enregistrée pour un usage ultérieur à l'aide de la classe *ResourceConfigIOStream*. Lors de cette opération, cette classe crée une section[1] pour ce réseau dans la base de données afin de regrouper toutes les informations qui seront enregistrées pour les ressources de ce réseau. La classe *ResourceConfigIOStream* permet aussi d'obtenir la liste des adresses (ou sections) enregistrée dans la base de données. Cette liste est présentée à l'utilisateur dans la fenêtre de l'application de localisation de services pour qu'il puisse choisir l'adresse IP du serveur de localisation de ressources du réseau auquel il est connecté.

La dernière adresse qui a été entrée ou choisie à partir de la fenêtre de l'application de localisation de services indique l'adresse du serveur de localisation de ressources courant. Cette adresse est également enregistrée dans une base de données spéciale au moment de sa sélection. Le *ServiceManager* et l'application de sélection de services retrouve l'adresse du serveur de localisation de ressources dans cette base de données lorsqu'ils en ont besoin.

3.3.4 Format des différentes bases de données

Les classes *ResourceIOStream* et *ResourceConfigIOStream* entreposent des données dans des bases de données. La classe *ResourceIOStream* enregistre les composants de type *Resource* définis par l'administrateur de réseau pour rendre accessible les ressources du réseau. La classe *ResourceConfigIOStream* sauvegarde les informations sur les services spécifiés par les applications et les ressources qui réalisent ces services sur chaque réseau. De plus, il existe une troisième base de données qui contient l'adresse IP du serveur de localisation de ressources du réseau courant.

1. La description d'une section est donnée dans la section 3.3.4

Base de données de ressources définies par l'administrateur

Les ressources définies par l'administrateur sont sauvegardées sous forme d'objets sérialisés. Chaque objet est sérialisé dans un fichier distinct. La méthode de sérialisation de Java est utilisée à cette fin.

Le nom du fichier pour une ressource est formé du nom du service fourni par la ressource suivi du caractère de soulignement et du nom de la ressource. L'extension des fichiers est « .dat ». Un exemple de nom de fichier valide est « PRINTER_hp.dat ». Cela indique que la ressource fournit le service d'impression (PRINTER) et se nomme hp.

Ces fichiers sont tous sauvegardés dans le même répertoire. Le nom du répertoire dans lequel les ressources doivent être enregistrées et récupérées est spécifié au moment de la création de l'instance de la classe *ResourceIOStream*.

Base de données de configuration de services et de ressources

La classe *ResourceConfigIOStream* est la seule utilisatrice de cette base de données. Le nom de la base de données est « NetResources ».

Cette base de données est divisée en sections. Une section possède un enregistrement pour son nom et une série d'enregistrements pour les informations qu'elle contient. Chaque application utilise une section distincte pour enregistrer les services qu'elle spécifie. Les applications enregistrent et retrouvent les informations dans cette base de données via l'objet *ServiceManager*, qui à son tour se sert de l'objet *ResourceConfigIOStream* pour arriver à ses fins. Le nom d'une section contenant les spécifications de services d'une application est le nom de l'application. Les autres sections sont utilisées pour enregistrer les informations des ressources qui réalisent les services pour chaque réseau. Le nom de chacune de ces sections est l'adresse IP du serveur de localisation de ressources d'un réseau. Les informations pour chaque service ou chaque ressource sont sauvegardées dans un enregistrement distinct.

Le format d'un enregistrement pour un service ou pour une ressource est comme suit :

NAME=*nom*, SERVICE=*service*, URL=*url*, ATTRIBUTES=*(nom1=valeur1), ..., (nomN=valeurN)*

où *nom* est le nom de la ressource (absent pour un service), *service* est le nom du service qu'elle réalise, *url* est l'URL de la ressource, le cas échéant, *nom1* jusqu'à *nomN* et *valeur1* jusqu'à *valeurN* sont les noms et les valeurs de chacune des propriétés d'un service ou des attributs d'une ressource le cas échéant.

L'exemple de la figure 33 présente une base de données valide. La section *PrintText* contient les spécifications de services pour notre application type. Cette spécification indique que le service désiré par l'application est celui d'une imprimante (PRINTER) et que cette dernière doit être en mesure d'imprimer des documents en langage postscript (LANGUAGE=POSTSCRIPT). La section *WebBrowser* contient la spécification, pour une application de ce nom, pour le service DNS. Les autres sections contiennent les ressources réalisant ces services sur les différents réseaux. Sur le réseau dont l'adresse du serveur de localisation de ressources est 132.210.48.188, le service d'impression correspondant à la spécification est accompli par la ressource *hp*, alors qu'il peut être réalisé par les ressources *hp* et *prof* sur l'autre réseau.

```
{PrintText}
SERVICE=PRINTER ATTRIBUTES=(LANGUAGE=POSTSCRIPT)
{WebBrowser}
 SERVICE=DNS
[132.210.48.188]
NAME=hp SERVICE=PRINTER ATTRIBUTES=(LANGUAGE=POSTSCRIPT),(LOCATION=local 1018)
[132.210.48.1]
NAME=hp SERVICE=PRINTER ATTRIBUTES=(LANGUAGE=POSTSCRIPT),(LOCATION=local 1018)
NAME=prof SERVICE=PRINTER ATTRIBUTES=(LANGUAGE=POSTSCRIPT),(LOCATION=local 1018)
NAME=dns_1 SERVICE=DNS URL=http://132.210.40.10
```

FIG. 33 – *Base de données de configuration de services et de ressources*

Base de données du réseau courant

Cette base de données contient seulement un enregistrement qui contient l'adresse IP du serveur de localisation de ressources du réseau courant. Le réseau courant est le

dernier choisi à partir de l'application de localisation de services. Cette base de données a pour nom « CurrentNetwork ». Elle est mise à jour seulement par l'application de localisation de services et elle est lue par cette même application, ainsi que par la classe *ServiceManager*.

3.4 Implantation

Dans cette section, nous présentons l'implantation physique de notre logiciel de gestion et localisation de services. La réalisation de ce logiciel comprend une partie client et une partie serveur. Afin de pouvoir exécuter le serveur de localisation de ressources et effectuer la gestion des ressources sur plusieurs plates-formes différentes, le langage de programmation Java est utilisé pour implanter cette partie du système. Le ORB Visibroker [2] pour Java de Inprise est utilisé pour le développement des composants répartis.

La partie client devant s'exécuter sur un ANP Palm Pilot de 3Com, l'environnement de développement CodeWarrior version 5 de Metrowerks est utilisé pour la programmation des applications. L'implantation du protocole IIOP de CORBA pour Palm Pilot, Mico, également développé avec cet environnement par l'Université de Frankfurt, est utilisé pour la communication avec les composants répartis résidants sur le serveur.

Par la suite, nous présentons les particularités de notre implantation. Nous terminons cette section en discutant de l'implication de l'utilisation de CORBA sur un réseau.

3.4.1 Architecture de l'implantation de la partie serveur

Le langage de définition d'interfaces (IDL) de CORBA est utilisé pour définir les interfaces des composants répartis. Les composants peuvent être regroupés dans des modules. Les interfaces IDL sont définies dans des fichiers ayant l'extension « .idl ». Lorsque le IDL est traduit en Java par le compilateur IDL, les modules deviennent des packages[2].

2. Un package correspond à un répertoire physique et peut regrouper plusieurs classes.

Les fichiers pour l'implantation de tous les composants faisant partie d'un module sont générés dans le package correspondant au module.

Six fichiers sont générés pour un composant, trois pour le côté client et trois pour le côté serveur. Ces fichiers contiennent tous le nom du composant dans leur nom. Par exemple, pour le composant *Resource*, les fichiers de la partie client sont *_st_Resource.java*, *ResourceHelper.java* et *ResourceHolder.java*, et ceux de la partie serveur sont *_ResourceImplBase.java*, *Resource.java* et *_example_Resource.java*. Le fichier *_st_Resource.java* contient la classe qui implante la souche pour l'objet *Resource*. Le fichier *ResourceHelper.java* contient une classe qui fournit des fonctions utiles pour les clients de *Resource*. La fonction *narrow*, qui permet de convertir un objet en un objet de type *Resource*, est un exemple de fonction fournie par cette classe. Le fichier *ResourceHolder.java* contient une classe possédant une instance publique du type *Resource*. Cette classe est utilisée par les clients et les serveurs pour passer des objets de type *Resource* en paramètres *in* et *inout* dans les appels de méthode. Le fichier *_ResourceImplBase.java* contient la classe qui implante le squelette de l'objet *Resource*. Le fichier *Resource.java* contient l'interface pour les méthodes définies dans l'interface IDL pour le composant. Finalement, le fichier *_example_Resource.java* contient une classe exemple pour l'implantation de l'objet *Resource*. Afin d'éviter d'écraser ce fichier à chaque fois que le fichier IDL est recompilé, nous avons implanté notre composant dans le fichier *ResourceImpl.java*. La classe *ResourceImpl* hérite de *_ResourceImplBase* et implante l'interface *Resource*.

Nous avons implanté un composant *Printer* pour démontrer l'utilisation d'une ressource du réseau à partir de notre application bloc note sur l'ANP. Ce composant implante le service d'impression permettant d'imprimer un texte sur une imprimante du réseau. Le composant utilise l'API Win32 de Windows 95 pour envoyer le texte à l'imprimante. Le composant utilise la bibliothèque de liaisons dynamiques *DynResDisc_PrinterImpl.dll* pour imprimer. Nous avons réalisé l'implantation de cette bibliothèque avec l'environnement Visual C++ 5.0 de Microsoft. L'utilisation de cette bibliothèque est décrite plus

loin dans cette section.

3.4.2 Architecture de l'implantation de la partie client

Les applications de la partie client sont entièrement développées avec l'environnement CodeWarrior. Cet environnement de développement utilise un module de projet pour gérer la conception d'une application. L'architecture de l'implantation de la partie client comporte trois projets : une bibliothèque statique, l'application client type et l'application de localisation de services. Le projet *ResourcesDisc* sert à créer la bibliothèque statique qui contient les objets nécessaires pour utiliser le service de localisation de ressources du serveur. Ces objets sont les souches qui permettent aux applications d'envoyer des messages aux composants répartis. La bibliothèque contient aussi l'objet *ResourceConfigIOStream* qui est responsable de l'enregistrement et la récupération des informations pour les services spécifiés par les applications et les ressources fournissant ces services. Le projet inclut la bibliothèque *IIOP.lib* qui contient l'implantation du protocole IIOP (c'est-à-dire Mico) utilisée par les objets souches.

L'application client type, c'est-à-dire l'application bloc note, est nommée *PrintText*. Le projet contient les fichiers du code source de l'application ainsi que ceux de la classe *ServiceManager*. Il comprend aussi le fichier contenant l'implantation physique des fenêtres de l'application. La librairie *ResourcesDisc* est incluse dans le projet puisque le *ServiceManager* se sert des objets qu'elle contient.

Enfin, l'architecture de l'implantation de l'application de localisation de services est similaire celle du projet *PrintText*, sauf que le projet ne contient pas la classe *ServiceManager*. Le projet pour cette application se nomme *ServLoc*.

3.4.3 Particularités de l'implantation

Notre implantation comporte trois particularités. La première est la création des bases de données sur l'assistant numérique personnel. La deuxième touche l'implantation de Mico, tandis que la troisième concerne l'implantation des services fournis par les composants répartis.

Création dynamique des bases de données

Sur PalmOS, les bases de données sont créés dynamiquement par les applications qui les utilisent à l'aide de fonctions du système d'exploitation. La création d'une base de données doit se faire seulement si elle n'existe pas déjà. Dans notre logiciel, les bases de données sont créées par les objets qui sont responsables de leurs mises à jour. La base de données « NetResources » est créée par la classe *ResourceConfigIOStream* au moment de la création de son instance. La base de données « CurrentNetwork » est créée par l'application de localisation de services.

Il est important de remarquer que les enregistrements d'une base de données n'ont pas de format prédéfini. Donc, si plusieurs applications ou objets doivent se partager une base de données, il est préférable d'encapsuler les accès à la base de données dans un objet. Ainsi, seulement cet objet est au courant des détails concernant la lecture et l'écriture des enregistrements. Les applications et les autres objets n'ont qu'à utiliser cet objet pour obtenir et modifier les données.

Implantation pour l'utilisation de Mico

Mico pour Palm Pilot est une implantation du protocole IIOP de CORBA. Il permet de communiquer avec un composant réparti dont le Interoperable Object Reference (IOR) est connu. Un IOR est une chaîne de caractères, interprétée par le protocole, qui contient les informations pour accéder à la référence d'un objet sur un nœud d'un réseau.

L'implantation disponible de Mico que nous avons utilisée servait à montrer la faisabilité de l'utilisation de composants répartis à partir d'un PDA. Un exemple de programme démontrant son utilisation est fourni avec l'implantation et il semble que certaines fonctionnalités du protocole ont été modélisées pour l'utilisation de cet exemple. Par le fait même, ces fonctionnalités qui se doivent normalement d'être d'usage général ne peuvent pas être utilisées par d'autres applications. Nous avons donc été dans l'obligation de modifier l'implantation de Mico pour rendre ces fonctionnalités utilisables par tous les genres d'applications.

La première modification que nous avons apportée à Mico est au niveau de la classe *GIOPCodec*. Cette classe permet de créer une référence d'objet pour un composant réparti. Dans Mico, le constructeur de cette classe reçoit en paramètre le contexte et le nom du composant pour lequel la référence doit être créée. Cette classe fait appel au service de nommage, c'est-à-dire la classe *Naming* pour obtenir le IOR du composant. La classe *Naming* n'est cependant pas un service de nommage compatible à la spécification CORBA. Elle envoie plutôt une requête HTTP à un serveur Web (celui de l'Université de Frankfurt) pour obtenir le IOR du composant désiré. Cette façon de procéder ne nous permet pas d'obtenir des références d'objets pour des composants autres que ceux connus du serveur Web utilisé dans l'implantation.

Pour contrer ce problème, nous avons ajouté un constructeur à la classe *GIOPCodec* ainsi qu'une méthode *GetObject* à la classe *Naming* qui nous permettent de spécifier une adresse IP, un port et le nom d'un composant. L'adresse IP et le port sont ceux du serveur à qui s'adresser pour obtenir le IOR du composant. Le nouveau constructeur de la classe *GIOPCodec* ressemble à ceci :

```
GIOPCodec::GIOPCodec(CharPtr serverIP, CharPtr port, CharPtr objectName):
          sendbuf((ULong) 100), recvbuf((ULong) 100) {

                .
                .
                .
  Naming *nameResolver = new Naming(pconn);
  CharPtr ior_ptr = nameResolver->GetObject(serverIP, port, objectName);
  delete nameResolver;
                .
                .
                .
}
```

Dans l'implantation de nos objets souche, l'adresse IP est spécifiée par l'utilisateur (via l'application de localisation de services), alors que le port est celui utilisé par l'objet *NamingServiceForMico*. Ce dernier est conçu spécialement pour fournir à Mico les IOR pour les composants demandés. Cet objet ne serait pas nécessaire dans notre implantation si Mico possédait un véritable service de nommage compatible à celui de CORBA. *NamingServiceForMico* permet aux applications sur le PDA d'obtenir le IOR du composant *ResourceLocator*.

La deuxième modification que nous avons apportée à Mico est pour nous permettre de construire une référence d'objet à partir d'un IOR. Cette modification est nécessaire car l'implantation de Mico permet seulement d'envoyer en paramètre et de recevoir en réponse des données de type simple, tels que des nombres, des caractères et des chaînes de caractères. Il n'est pas possible de transmettre ou de recevoir via Mico des tableaux, des objets ou d'autres données de type complexe. Dans notre logiciel, nous demandons au *ResourceLocator* de nous transmettre la liste des ressources qui réalisent certains services. Puisque Mico ne permet pas de recevoir des objets, le *ResourceLocator* nous envoi la liste de IOR de ces objets. De plus, puisque Mico ne permet pas de recevoir un tableau, les IOR sont concaténés les uns à la suite des autres. La construction des références d'objet est donc effectuée à partir des IOR.

Le serveur de localisation de ressources a été implanté pour fournir des services aux clients des autres ORB. Toutefois, comme nous l'avons déjà mentionné, Mico n'implante pas toutes les fonctionnalités d'un ORB CORBA. Nous avons donc dû ajouter certains éléments à l'implantation de notre serveur et de la classe *ResourceLocator* pour des fonctionnalités utilisées exclusivement par Mico. Premièrement, comme nous l'avons mentionné, Mico utilise le service *NamingServiceForMico* pour obtenir la référence du *ResourceLocator*. Le serveur doit donc enregistrer le *ResourceLocator* auprès de l'instance du *NamingServiceForMico*. Deuxièmement, toutes les méthodes du *ResourceLocator* qui retourne des objets ou une liste d'objets ne peuvent pas être utilisées avec Mico. Nous avons donc dupliqué l'implantation des services réalisés par ces méthodes pour qu'elles retournent les références d'objet sous la forme de IOR, qui est une chaîne de caractères. Le nom de ces nouvelles méthodes, à l'usage exclusif de Mico, se termine par *ForMico*. Ces méthodes sont appelées à disparaître si l'implantation de Mico en venait à permettre le transfert de références d'objet. L'exemple suivant montre l'implantation d'une méthode utilisée par tous les ORB CORBA et d'une méthode utilisée par Mico :

```
// Utilise par tous les ORB
public DynResDisc.Resource[] getAllResourcesByService(String service) {
  DynResDisc.Resource resource[] = null;

  try {
    CosTrading.Offer offer[] = mySuper.get(service, new CosTrading.Property[0]);

    if(offer != null) {
      resource = new DynResDisc.Resource[offer.length];
      for(int i=0; i<offer.length; i++)
        resource[i] = DynResDisc.ResourceHelper.narrow(offer[i].reference);
    }
  }
  catch(org.omg.CORBA.UNKNOWN e) {
    throw e;
  }
  return resource;
}

// Utilise par Mico
public String getAllResourcesByServiceForMico(String service) {
  String resources = null;

  try {
    CosTrading.Offer offer[] = mySuper.get(service, new CosTrading.Property[0]);

    if(offer != null) {
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(new String[0],null);

      resources = String.valueOf(offer.length);
      for(int i=0; i<offer.length; i++)
        resources = resources + ";" +
        // obtention du IOR de l'objet
        orb.object_to_string(DynResDisc.ResourceHelper.narrow offer[i].reference));
    }
  }
  catch(org.omg.CORBA.UNKNOWN e) {
    throw e;
  }
  return resources;
}
```

La méthode utilisée par tous les ORB retourne un tableau d'objets *Resource*. Par contre, la méthode utilisée par Mico retourne une chaîne de caractères contenant les IOR des objets concaténés et séparés par des points-virgules. La méthode *object_to_string* de l'objet *ORB* permet d'obtenir le IOR d'un objet.

Il est à remarquer que puisque Mico ne permet pas toutes les fonctionnalités d'un véritable ORB, beaucoup de transformations sur les données doivent être effectuées dans l'implantation, autant dans la partie client que dans la partie serveur du logiciel, avant l'envoi des requêtes et après la réception des réponses.

Implantation des services des composants de type Resource

Les implantations des services offerts par les composants de type *Resource* peuvent être très variées. Les services offerts peuvent être réalisés en utilisant seulement les API de Java, mais parfois ils peuvent nécessiter des fonctionnalités qui ne sont pas réalisables directement avec ces API.

Le composant *Printer* que nous avons développé permet de rendre les imprimantes d'un réseau accessibles aux applications sur un assistant numérique personnel. Pour implanter l'impression dans la méthode *print* du composant *Printer*, nous avons envisagé utiliser l'objet Java *PrinterJob*. Toutefois, cet objet affiche une fenêtre de dialogue pour permettre à l'utilisateur de choisir l'imprimante qui imprimera le document. Cette approche ne convient pas. En effet, lorsqu'un utilisateur transmet un document à imprimer, une personne doit se rendre à l'ordinateur du réseau qui exécute le composant afin de choisir l'imprimante dans la liste de sélection et cliquer sur le bouton *OK*. De plus, l'objet *PrinterJob* ne permet pas l'association d'une imprimante précise au composant lors de sa création.

L'approche que nous avons utilisée consiste à faire appel directement aux fonctions du système d'exploitation pour l'implantation des services du composant *Printer*. Dans notre cas, nous avons utilisé l'API Win32 pour obtenir la liste des imprimantes du réseau

et pour implanter l'impression d'un document. Le composant *Printer* peut faire appel aux fonctions que nous avons créées en utilisant les méthodes natives Java [45]. Pour utiliser les méthodes natives, celles-ci doivent être dans une bibliothèque de liaisons dynamiques. Nous avons donc créé la bibliothèque *DynResDisc_PrinterImpl.dll.* L'exemple suivant illustre en partie l'utilisation d'une méthode native.

Premièrement, le chargement de la bibliothèque dynamique contenant l'implantation de la méthode se fait avec la méthode *loadLibrary* de l'objet *System.*

```
static
{
  System.loadLibrary("DynResDisc_PrinterImpl");
}
```

Les clients utilisent la méthode *print* de l'objet *Printer* pour imprimer. Cette méthode se sert de la méthode native *print* pour imprimer. L'appel de la fonction *print* qu'elle contient, ainsi que la déclaration de la méthode native en Java et en C++.

```
// methode print du composant
public void print(String text) {
              .
              .
              .
    //appel de la methode native
    print(realPrinterName, text);
              .
              .
              .
}

private native void print(String printerName, String textToPrint)
throws Exception;
```

Lorsque la méthode native est appelée, c'est la fonction associée à cette méthode dans la bibliothèque dynamique qui est exécutée. Le nom de la fonction appelée est composé du mot Java, du nom de la hiérarchie de packages où se trouve l'objet, du nom de l'objet et du nom de la méthode. Chaque nom est précédé du caractère de soulignement. Dans

notre cas, la fonction *Java_DynResDisc_PrinterImpl_print* est appelée. Sa déclaration en C++ ressemble à ceci :

```
// declaration C++ de la mthode native
#include <jni.h>

JNIEXPORT void JNICALL Java_DynResDisc_PrinterImpl_print
  (JNIEnv *, jobject, jstring, jstring);
```

Il est important de mentionner que notre composant *Printer* fonctionne seulement sur Windows 95. Il s'avère qu'avec Windows NT il faille utiliser certaines options différentes dans les appels de l'API Win32. Nous avons essayé les options indiquées dans la littérature, ainsi que toutes sortes de combinaisons, mais cela n'a pas fonctionné.

3.4.4 Discussion sur l'utilisation de CORBA sur un réseau

CORBA facilite beaucoup le développement des systèmes répartis. L'indépendance qu'il permet face à la localisation des composants permet au programmeur de ne pas se soucier des détails associés aux communications distantes. En général, lorsqu'une facilité quelconque pour une fonctionnalité plus complexe est fournie au développeur, un certain coût y est associé. Ainsi, la transparence de localisation qu'offre CORBA peut nous amener à nous demander si la prise en charge de cette fonctionnalité par le ORB occasionne des coûts supplémentaires, entre autres en ce qui concerne le trafic généré sur le réseau. Afin de nous éclairer sur ce point, nous avons inspecté le trafic engendré par l'utilisation de notre logiciel.

Nous avons utilisé le logiciel CNApro Network Analyzer de Chevin Software [8] pour regarder les paquets échangés pendant l'utilisation de notre système. Ce logiciel permet d'obtenir des informations concernant les paquets, entre autres leur taille, le protocole de routage, la longueur des datagrammes, le protocole de transport, l'adresse source, l'adresse destination, le port source, le port destination ainsi que les données contenues

dans les paquets.

Puisque nous utilisons le protocole IIOP pour la communication entre les ORB, les messages échangés sont transportés dans des segments du protocole TCP. L'analyse du trafic capturé pendant l'utilisation de notre logiciel révèle que le nombre de segments échangés par les ORB et leurs composants est comparable au nombre de segments qui seraient échangés en utilisant un système client-serveur.

La figure 34 illustre les segments échangés lors de la création d'une référence d'objet, l'appel d'une méthode de cette référence et la destruction de cette référence. À la création d'une instance, les messages *Synchronise*, *Ack Synchronise* et *Ack* sont échangés entre la référence d'objet et le BOA de l'objet désiré. Ces trois messages servent à établir une connexion TCP. Lors de l'appel d'une méthode, le message *Ack Push* est envoyé au BOA de l'objet. Ce message contient le nom de l'objet, le nom de la méthode ainsi que les valeurs des paramètres de la méthode. Si la méthode retourne une donnée, le message *Ack Push* est envoyé par le BOA, alors que le message *Ack* est envoyé pour accusé réception du message précédent si la méthode ne retourne pas une donnée. La référence d'objet envoie ensuite le message *Ack* pour accuser réception de la réponse du BOA le cas échéant. Enfin, lorsque la référence d'objet est détruite, quatre messages sont échangés. Ces messages servent à effectuer la procédure normale de fermeture d'une connexion TCP.

En plus de la quantité de segments échangés, la taille de ceux-ci peut affecter l'utilisation de la largeur de bande d'un réseau. En effet, plus la taille des messages transmis par une application est grande, plus la largeur de bande est monopolisée par cette application. Nous avons donc examiné les frais fixes au niveau des messages du protocole IIOP. Les messages IIOP sont échangés seulement lors des appels de méthodes et des réponses à ces appels. Lors de l'appel d'une méthode, l'en-tête du message IIOP contient toujours 44 octets. Le reste du message est de longueur variable. Il contient le type et le nom unique de l'objet sur lequel la méthode doit être appelée, le nom de la méthode et

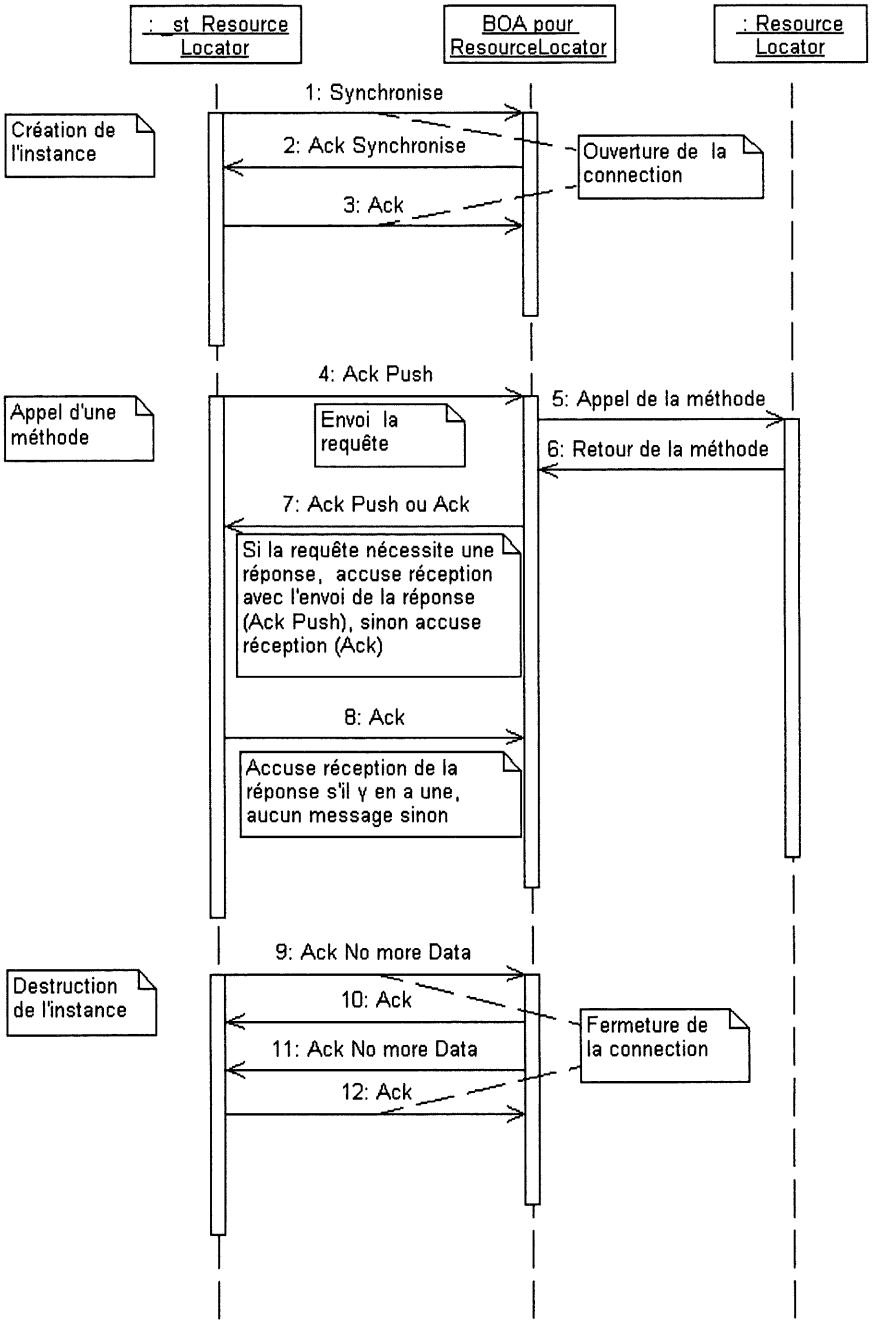

FIG. 34 – *Segments TCP échangés pour la création d'une référence d'objet, l'appel d'une méthode et la destruction de la référence*

les paramètres de la méthode. Pour une réponse, l'en-tête du message est de 28 octets. Le reste du message contient les données retournées par la méthode. Comme on est en mesure de le constater, les frais fixes du protocole IIOP pour la taille des messages sont minimes et n'impliquent pas un usage important de la largeur de bande.

Après cette vérification, nous pouvons affirmer que l'utilisation de CORBA, en ce qui concerne le protocole IIOP, a un impact très négligeable sur le trafic du réseau comparativement à l'avantage qu'il apporte. En ce qui concerne l'usage des composants répartis, il est certain que si un appel individuel de méthodes est effectué sur un composant pour obtenir chaque élément de données, le nombre de messages échangés sur le réseau sera grand. Il est de la responsabilité du développeur de prévoir le transfert en lot de données afin de diminuer le trafic sur le réseau.

3.5 Évaluation du logiciel

Le logiciel que nous avons développé est à l'état de prototype. Certaines améliorations peuvent être apportées afin de le rendre plus convivial et augmenter le degré d'automatisation. Dans cette section, nous présentons les éléments améliorables de notre logiciel ainsi que les possibilités de travaux futurs.

3.5.1 Éléments améliorables

Notre logiciel a été développé dans le but de montrer la faisabilité de retrouver et d'utiliser les services d'un réseau à partir d'un assistant numérique personnel. De ce fait, une plus grande importance a été accordée à la convivialité et au côté pratico-pratique de la partie client que ceux de la partie serveur. Des améliorations de la partie client sont toutefois possibles.

Tout d'abord, l'obtention de l'adresse IP du serveur de localisation de ressources d'un réseau peut se faire automatiquement. En effet, cette adresse peut être obtenue par la

diffusion d'un message sur le réseau plutôt que d'exiger la saisie de cette donnée par l'utilisateur au travers de l'application de localisation de services. Il suffit que le serveur de localisation de ressources écoute sur un port précis et retourne son adresse à l'émetteur du message lors de la réception de ce dernier.

Deuxièmement, la performance des applications client peut être améliorée. Même si la performance au point de vue du temps d'exécution des diverses fonctionnalités est acceptable, il n'en demeure pas moins qu'aucune optimisation n'a été faite dans le code. De plus, la récupération de la mémoire est un autre aspect où une amélioration de la performance peut être obtenue. Dans notre code, la mémoire est récupérée lorsqu'elle est allouée par une méthode ou une fonction et qu'elle est utilisée seulement par celle-ci. Par contre, si l'espace mémoire allouée est utilisée par d'autres méthodes ou fonctions, il n'est pas récupéré dans la plupart des cas. Ainsi, ces zones de mémoire sont récupérées seulement lorsque l'application se termine.

Actuellement, les applications client sont de taille assez grande (entre 59 et 66 kilo octets). Ce qui rend les applications volumineuses, c'est l'ensemble du code concernant les souches des composants répartis ainsi que la classe *ResourceConfigIOStream*. Toutes les applications qui localisent et utilisent des services ont le code objet de ces fonctionnalités intégré dans leur programme exécutable. Le moyen de diminuer la taille des applications est de mettre le code partagé par toutes les applications dans une bibliothèque de liens dynamiques. Malheureusement, ce type d'unité logicielle n'existe pas sur le système d'exploitation PalmOS. S'il le devenait, cette solution serait à envisager afin de réduire la taille des applications.

Enfin, en ce qui concerne la structure de l'implantation de la partie client, la classe *ServiceManager* devrait être mise dans une bibliothèque statique puisqu'elle doit être utilisée par toutes les applications qui spécifient et utilisent des services d'un réseau. Actuellement, cette classe est incluse dans le projet de notre application type puisque seulement celle-ci l'utilise.

Du côté de la partie serveur, des améliorations peuvent être apportées à l'interface de la fenêtre d'édition des ressources afin de la rendre plus conviviale. Dans son état actuel, la fenêtre possède des champs généraux pour la définition d'un service et seulement des champs spécifiques pour la définition du service *PRINTER*. Des champs spécifiques pour la définition d'autres services devraient aussi être implantés. De plus, les champs pour la définition d'un service spécifique devraient faire partie d'une fenêtre distincte. Dans la fenêtre de gestion des ressources, la possibilité de pouvoir distinguer les ressources actives de celles qui ne le sont pas serait intéressante. De plus, la possibilité de pouvoir activer ou désactiver une ressource dynamiquement pendant que le serveur de localisation de ressources est en fonction serait une fonctionnalité très utile. En ce qui concerne l'implantation, la classe *ResourceLocator* devrait être modifiée si une version du ORB Visibroker avec le service de type page jaune devenait disponible. Ce service devrait être utilisé plutôt que notre implantation du service de vendeur.

3.5.2 Travaux futurs

Au niveau de l'application de gestion des ressources, un certain nombre de propriétés pourraient être définies automatiquement lorsque l'administrateur définit des ressources. Pour la plupart des ressources d'un réseau, il existe des MIBs SNMP qui contiennent les caractéristiques de la ressource et leurs valeurs instantanées. L'application pourrait utiliser le protocole SNMP pour définir une série de propriétés pour les ressources plutôt qu'elles soient définies manuellement par l'administrateur. Dans un autre ordre d'idée, des mécanismes d'authentification devraient être implantés pour déterminer les droits d'accès au serveur de localisation de ressources ainsi que les droits d'utilisation de chacune des ressources. Une fois la sécurité implantée au niveau du serveur de localisation de ressources, l'application de gestion de ressources pourrait être utilisée sur un nœud autre que celui qui exécute le serveur pour enregistrer des ressources dans le système. Les fonctionnalités pour l'enregistrement des ressources à distance sont déjà disponibles

mais elles ne sont pas utilisables à distance présentement. Il suffit d'ajouter les méthodes correspondantes dans le IDL du composant *ResourceLocator* pour qu'elles soient disponibles.

Conclusion

Le nombre d'utilisateurs d'ordinateurs et d'appareils de communication mobiles est en croissance constante. Ces appareils sont de plus en plus petits afin de faciliter leur transport et leur usage dans les activités des utilisateurs. La petite taille de ces dispositifs implique une diminution des capacités des ressources matérielles (e.g., mémoire, processeur) qui les composent. Il n'en demeure pas moins qu'un appareil, tel un assistant numérique personnel, puisse faire partie d'un réseau informatique. Cela implique que tous les services du réseau soient mis à sa disposition de la même façon que pour les ordinateurs de taille plus grande. De plus, puisqu'il est mobile, il est probable qu'il doive utiliser les services de plusieurs réseaux différents. Cette mobilité oblige les utilisateurs à configurer leurs applications pour utiliser les services d'un réseau donné à chaque fois qu'ils changent d'endroit géographique. Pour ce faire, ils doivent découvrir les ressources du réseau qui offrent les services dont ils ont besoin. Plusieurs protocoles existent présentement pour la localisation de services sur un réseau.

La contribution apportée par la recherche présentée dans ce mémoire est un prototype d'un système de gestion et localisation de services sur un assistant numérique personnel. La plupart protocoles qui existent présentement permettent seulement de localiser les services. L'utilisation des services dans les applications nécessite une connaissance des protocoles utilisés par les services et parfois même de l'implantation des services. Les protocoles qui permettent un accès direct aux services ne sont pratiquement pas utilisables présentement sur les assistants numériques personnels à cause des technologies

qu'ils utilisent. Notre logiciel, en plus de permettre la localisation de services, permet également l'utilisation des services directement à partir des références sur les ressources offrant ces services. L'utilisation de CORBA et des composants répartis pour notre implantation apporte une indépendance aux applications face à l'implantation des services, de même qu'une facilité de leur utilisation.

De plus en plus, des entreprises de différents secteurs reçoivent pour de courtes périodes des travailleurs externes qui viennent leurs apporter expertises ou services. Beaucoup de ces travailleurs transportent avec eux leur petit ordinateur mobile afin d'effectuer le travail. Il en est de même pour les personnes qui sont amenées à voyager souvent pour le travail. Ces personnes peuvent avoir besoin d'utiliser des ressources du réseau où ils se trouvent pour obtenir des services tels l'impression d'un document, l'envoi d'un fax, Internet, et tout cela à partir de leur ordinateur mobile. La présence d'un logiciel de gestion et de localisation de services sur le réseau devient donc presque une nécessité pour faciliter la découverte et l'utilisation des services.

Annexe A

Acronymes

Nom		Signification
ACE	:	Adaptative Communication Environment
ANP	:	Assistant numérique personnel
API	:	Application Programming Interface
ASN.1	:	Abstract Syntax Notation One
ATM	:	Asynchronous Transfer Mode
CMI	:	Common Management Information
CMIP	:	Common Management Information Protocol
CMIS	:	Common Management Information Services
CMISE	:	Common Management Information Service Element
CORBA	:	Common Object Request Broker Architecture
COM	:	Component Object Model
DCE	:	Distributed Computing Environment
DCOM	:	Distributed Component Object Model
DHCP	:	Dynamic Host Configuration Protocol
DII	:	Dynamic Invocation Interface
DMI	:	Desktop Management Interface

DMTF	:	Desktop Management Task Force
DSI	:	Dynamic Squeleton Interface
EDF	:	Earliest Deadline First
FIFO	:	First In First Out
GDMO	:	Guidelines for the Description of Managed Objects
GIOP	:	General Inter-ORB Protocol
IANA	:	Internet Assigned Number Authority
IDL	:	Interface Definition Language
IETF	:	Internet Engineering Task Force
IIOP	:	Internet Inter-ORB Protocol
ILP	:	Integrated Layer Processing
IOR	:	Interoperable Object Reference
IP	:	Internet Protocol
IrDA	:	Infra Red Dat Association
ISO	:	International Standards Organization
ITU	:	International Telecommunication Union
JIDM	:	Joint Inter-Domain Management
LDAP	:	Lightweight Directory Access Protocol
MIB	:	Management Information Base
MIDL	:	Microsoft IDL
MFL	:	Maximum Laxity First
MIF	:	Management Interface Format
MMTF	:	Mobile Management Task Force
MUF	:	Maximum Urgency First
NTP	:	Network Time Protocol
OMA	:	Object Management Architecture
OMG	:	Object Management Group

ORB	:	Object Request Broker
ORPC	:	Object RPC
OSI	:	Open Systems Interconnection
POA	:	Portable Object Adapter
RDP	:	Resource Discovery Protocol
RFP	:	Request For Proposal
RIOP	:	Real-time Inter-ORB Protocol
RMI	:	Remote Method Invocation
RMON	:	Remote Network Monitoring
RMS	:	Rate Monotonic Scheduling
RPC	:	Remote Procedure Call
RTU	:	Real-time Upcall
SAP	:	Service Advertising Protocol
SCM	:	Service Control Manager
SII	:	Static Invocation Interface
SLP	:	Service Location Protocol
SMF	:	System Management Functions
SMFA	:	System Management Functionnal Areas
SMI	:	Structure of Management Information
SNMP	:	Simple Network Management Protocol
TAO	:	The ACE ORB
TCP	:	Transport Control Protocol
TDMI	:	Timed Distributed Method Invocation
UDP	:	User Datagram Protocol
UML	:	Unified Modeling Language
URL	:	Uniforme Resource Locator

Bibliographie

[1] http://www.vsb.informatik.uni-frankfurt.de/ mico/pilot/index/html.

[2] http://www.borland.com/visibroker/.

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

[4] http://ligwww.epfl.ch/software/ilu/manual_toc.html.

[5] http://java.sun.com/products/jdk/rmi/index.html.

[6] http://www.orbycom.fr/OpenSnmp.html.

[7] http://www.outbackinc.com/products/jsnmp/.

[8] http://www.chevin.com/.

[9] Networking Guide. http://www.steview.com:457NetAdminG/ipxC.service_advert_-protocol.html.Z.

[10] SNMP : MMTF releases four proposed mobile MIBs for SNMP. *EDGE*, 11(391):24, 1996.

[11] Alcatel, Hewlett-Packard Company, Lucent Technologies Inc., Object-Oriented Concepts Inc., Sun Microsystems Inc., and Tri-Pacific. Real-time CORBA. Draft, 1998.

[12] TechniCom Australia. RMON & HSRMON. http://www.techni.com.au/html/farmon.html.

[13] M. Baldi, S. Gai, and G. P. Picco. Exploiting Code Mobility in Decentralized and Flexible Network Management. In *Mobile Agents. First International Workshop, MA'97 Proceedings, Berlin, Germany*, pages 13–26. Springer-Verlag, 1997.

[14] M. Baldi and G. P. Picco. Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications. In *Proceedings of the 1998 International Conference on Software Engineering. Forgong New Links, Kyoto, Japan*, pages 146–155. IEEE Computer Society, 1998.

[15] B. Ban. Extending CORBA for Multi-Domain Management. IBM Zurich Research Laboratory, 1996.

[16] A. Bierman, G. Greene, B. Stewart, and S. Waldbusser. Distributed Management Framework. Internet Draft, 1998.

[17] A. Bieszczad, B. Pagurek, and T. White. Mobile Agents for Network Management. *IEEE Communications Surveys*, (quatrième quart de 1998), 1998. http://www.comsoc.org/pubs/surveys/.

[18] J.-M. Chauvet. *Corba, ActiveX et Java Beans.* Eyrolles, 1997.

[19] Z. Choukair and A. Beugnard. Real-time Object-oriented Distributed Processing with COREMO. In S. Mitchell J. Bosch, editor, *Object-Oriented Technology ECOOP'97 Workshop Reader - Lecture Notes in Computer Sciences 1357, Jyvaskyla, Finland*, pages 45–50. Springer-Verlag, 1998.

[20] Highlander Communications and Visigenic Software Inc. Real-time CORBA – join initial submission. Draft, 1998.

[21] UH Communications. The CORBA-to-CMIP Gateway. http://www.uhc.dk/q3c2cgtw/q3c2cgtw.html.

[22] 3Com Corporation. RMON Methodology. http://www.3com.com/nsc/500251.html.

[23] Intel Corporation. Intel Desktop Management Interface (DMI) and Management Information Format (MIF) Technical Overview.

http://www.intel.com/support/desktopmgmt/9617.htm.

[24] Microsoft Corporation. DCOM Architecture white paper. White Paper, 1998.

[25] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design.* Addison-Wesley, 1994.

[26] M. Daniele, B. Wijnen, and D. Francisco. Agent Extensibility (AgentX). RFC2257, 1998.

[27] L. Deri. Network Management for the 90s. In *ECOOP'96 Workshop on OO Technology for Service and Network Management, Linz, Austria*, pages –, 1996.

[28] L. Deri and B. Ban. Static vs. Dynamic CMIP/SNMP Network Management Using CORBA. In *Lecture Notes in Computer Sciences 1238*, pages –, 1997.

[29] W. Keith Edwards. *Core Jini.* Prentice Hall PTR, 1999.

[30] E. Gamma, R. Helm, R. Johnson, J. Vlissides, and G. Booch. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[31] C. D. Gill, D. L. Levine, and D. C. Schmidt. The Design and Performance of a Real-time CORBA Scheduling Service. *International Journal of Time-Critical Computing Systems*, pages –, 1999.

[32] R. H. Glitho. Contrasting OSI Management to SNMP and TMN. *Journal of Network and Systems Management*, 6(2):113–133, 1998.

[33] A. S. Gokhale and D. C. Schmidt. Measuring and Optimising CORBA Latency and Scability Over High-speed Networks. Washington University, Department of Computer Science, 1998.

[34] A. S. Gokhale and D. C. Schmidt. Principles for Optimising CORBA Internet Inter-ORB Protocol Performance. Washington University, Department of Computer Science, 1998.

[35] G. Goldszmidt and Y. Yemini. Delegated Agents for Network Management. *IEEE Communication Magazine*, 36(3):66–70, 1998.

[36] Object Managament Group. A Discussion of the Object Management Architecture. White Paper, 1997.

[37] Object Management Group. Realtime CORBA 1.0 Request For Proposal. Request For Proposal, 1997.

[38] Object Management Group. CORBAservices: Common Object Services Specification. Specifications, OMG, 1998.

[39] The Open Group. Inter-Domain Management: Specification Translation. http://www.opengroup.org/onlinepubs/8349099/chap01.htm.

[40] P. Haggerty and K. Seetharaman. The Benefits of CORBA-Based Network Management. *Communications of the ACM*, 41(10):73–79, 1998.

[41] D. Harrington, R. Presuhn, and B. Wijnen. An Architecture for Describing SNMP Management Frameworks. RFC2271, 1998.

[42] T. H. Harrison, C. O'Ryan, D. L. Levine, and D. C. Schmidt. The Design and Performance of a Real-time CORBA Event Service. Soumis au IEEE Journal on Selected Areas in Communications, 1997.

[43] Lockheed Martin Federal Systems Inc. Real-time CORBA. Draft, 1998.

[44] Objective Interface Systems Inc. Real-time CORBA – initial submission. Draft, 1998.

[45] Sun Microsystems Inc. Java Native Interface. http://java.sun.com/products//jdk/1.2/docs/guide/jni/index.html.

[46] Sun Microsystems Inc. Jini Architectural Overview. Technical White Paper, 1999.

[47] SNMP Research International. Introduction to SNMPv3. http://www.snmp.com/white.html.

[48] K. Iseda, T. Chujo, and T. Suzuki. CORBA-Based Network Operation System Architecture. In *1998 IEEE Networks Operations and Management Symposiumi, New Orleans, United States*, pages 639–648. IEEE, 1998.

[49] H. Johner, L. Brown, F.-S. Hinner, W. Reis, and J. Westman. Understanding LDAP. Book, IBM Corporation, 1998.

[50] A. Keller. Systems Management with Distributed Objects: Porting SNMP Agents to a CORBA environment. In *Proceedings of the 4th Workshop of the OpenView University Association: OVUA'97, Madrid, Espagne*, pages –, 1997.

[51] D. Levi and J. Schönwälder. Definitions of Managed Objects for the Delegation of Management Scripts. Internet Draft, 1999.

[52] S. Mazumdar. Inter-Domain Management between CORBA and SNMP: WEB-based Management - CORBA/SNMP Gateway Approach. Présenté au septième IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, L'Aquila, Italie, 1996.

[53] R. Orfali, D. Harkey, and J. Edwards. *Client/Serveur Guide de survie.* International Thomson Publishing France, 1997.

[54] J. Park, K.-C. Sohn, and J.-W. Hong. Web-based Customer Network Management. In *First IEEE Enterprise Networking Mini-Conference (ENM-97) in conjunction eith the ICC-97, Montreal, Canada*, pages 160–169. IEEE, 1997.

[55] C. Perkins. SLP white paper topic. White Paper, 1997.

[56] C. Perkins and H. Harjono. Resource Discovery Protocol for Mobile Computing. In *Mobile Comunications. Technology, Tools, Applications, Authentication and Security. IFIP World Conference on Mobile Communications, Canberra, Australia*, pages 219–236. Chapman & Hall, London, UK, 1996.

[57] C. Perkins and K. Luo. Using DHCP with computers that move. *Wireless Networks*, 1(3):341–353, 1995.

[58] D. Perkins and E. McGinnis. *Understanding SNMP MIBs.* Prentice Hall PTR, 1997.

[59] R. Raud. OSI and CORBA, Alternatives or Complementary. http://www.magicnet.net/~rraud/OSICorba.html.

[60] M. T. Rose. *The Simple Book: An Introduction to Networking Management.* Prentice Hall PTR, 1996.

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

[62] A. Sahai and C. Morin. The Mobile Agent Enhanced Thin Client Approach to Network Management. In *Proceedings of the IEEE Singapore International Conference on Networks (SICON'98), Singapore*, pages 433–445. World Scientific Publishing, 1998.

[63] D. C. Schmidt. Evaluating Architectures for Multi-threaded CORBA Object Request Brokers. Washington University, Department of Computer Science, 1998.

[64] D. C. Schmidt, D. L. Levine, and S. Mungee. The Design of the TAO Real-time Object Request Broker. Washington University, Department of Computer Science, 1997.

[65] D. C. Schmidt, S. Mungee, S. Flores-Gaitan, and A. Gokhale. Alleviating Priority Inversion and Non-determinism in Real-time CORBA ORB Core Architectures. In *Proceedings of the Fourth IEEE Real-Time Technology and Applications Symposium (RTAS), Denver, Colorado*, pages 92–101. IEEE COmputer Society, 1998.

[66] J. Schönwälder. Network Management By Delegation From Research Prototypes Toward Standards. *Computer Networks and ISDN Systems*, 29(15) :931–1–931–9, 1997.

[67] M. Sharrott, G. Hall, S. Fukui, W. Shibata, and A. Enjou. Multi-Operator Access to a Network Management System. In *1998 IEEE Network Operations and Management Symposium, New Orleans, United States*, pages 334–341. IEEE, 1998.

[68] C.-C. Shen and J. Y. Wei. The Network As Distributed Object Database. In *1998 IEEE Network Operations and Management Symposium, New Orleans, United States*, pages 540–548. IEEE, 1998.

[69] O. Tallman and J. Bradford Kain. COM versus CORBA : A Decision Framework. *Distributed Computing*, pages –, 1998.

[70] Asanté Technologies. RMON Remote Monitoring. http://www.asante.com/FAQ/faqrmon.html.

[71] Iona Technologies and Northern Telecom. Real-time CORBA. Draft, 1998.

[72] J. Veizedes, E. Guttman, C. Perkins, and S. Kaplan. Service Location Protocol. IETF Request for Comments 2165, 1997.

[73] V. F. Wolf, L. C. DiPippo, R. Ginis, M. Squadrito, S. Wohlever, and I. Zykh et R. Johnston. Real-time CORBA. In *Proceedings of the Real-Time Technology and Applications Symposium, Montreal, Canada*, pages 148–157. IEEE Computer Society, 1997.

[74] T. Zhang and S. Covaci. Java-based Mobile Intelligent Agents as Network Management Solutions. In *Proceedings of the JENC8. 8th Joint European Networking Conference (JENC8). Diversity and Integration : The New European Networking Landscape, Edinburgh, UK*, pages 933–1–933–10. TERENA, 1997.

[75] S. Znaty, M. Lion, and J.-P. Hubaux. DEAL : A Delegated Agent Language for Developping Network Management Functions. In *PAAM 96 Proceedings First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, London, UK*, pages 923–933. Practical Application Company, 1996.