Thèse

présentée pour obtenir le grade de docteur
de l'Ecole nationale supérieure des télécommunications

Spécialité: Informatique et Réseaux

Pierre Olivier Conti

Agents Intelligents: Emergence d'une nouvelle technologie pour la Gestion de Réseaux

Soutenance prévue le 17 juillet 2000 devant le jury composé de:

André Schaff	Rapporteur
Omar Cherkaoui	Rapporteur
Guy Pujolle	Examinateur
Samir Tohmé	Examinateur
Raùl Teixeira de Oliveira	Examinateur
Jacques Labetoulle	Directeur de thèse

à l'Institut Eurécom Sophia Antipolis

Ecole nationale supérieure des télécommunications

Table des matières

Liste des figures

I Introduction

I.1 Présentation de la thèse

I.1.1 Cadre de la recherche

En moins de deux décennies, les réseaux informatiques sont devenus des éléments indispensables de notre vie tant professionnelle que privée. Ce phénomène qui amène certains à considérer que nous sommes rentrés dans "l'ère du réseau" [Slo94] s'est accéléré ces dernières années autour de quatre axes principaux:

Les Télécommunications

Passés de l'ère des réseaux analogiques à l'ère des réseaux numériques pour transporter de la voix, les réseaux de télécommunications s'orientent vers le transport de données. L'ouverture des marchés de télécommunications à la concurrence a entraîné la multiplication des réseaux de télécommunications proportionnellement au nombre de nouveaux opérateurs. Il est de plus en plus courant que particuliers ou entreprises utilisent plusieurs opérateurs et par conséquent plusieurs réseaux pour satisfaire leurs besoins en communications.

Les Entreprises

Les réseaux comme support à l'utilisation des systèmes d'information des grosses entreprises se sont maintenant aussi imposés dans les petites entreprises où ils permettent un partage facile des données et un accès immédiat aux informations.
Avec la mondialisation de l'économie, ces réseaux doivent s'adapter au découpage ou regroupement des entreprises, liés aux opérations de ventes et de rachats.

L'Internet

L'Internet appelé quelquefois le réseau des réseaux continue à se développer aussi bien en terme de nombre d'abonnés que de capacités. La prolifération d'un côté

des FAI (fournisseurs d'accès à internet), et de l'autre, des services comme la vidéo à la demande, la téléphonie ou le commerce électronique, nécessitent une remise en question régulière des équipements et protocoles utilisés par les différents acteurs d'Internet.

La Domotique

Encore en phase d'émergence, les réseaux domotiques permettront à terme de contrôler tous les systèmes aussi bien multimédias, en permettant par exemple l'écoute de musiques ou la visualisation de vidéos (dans n'importe quelle pièce d'un immeuble), que les systèmes de sécurité ou de confort en permettant l'activation des fermetures ou la régularisation de l'éclairage et de la climatisation.

Les réseaux étant devenus indispensables, il est important de pouvoir les gérer facilement et efficacement afin de prévenir et de réparer toute défaillance. D'un autre côté, la diversité et l'évolution continuelle des réseaux imposent que les moyens de gestion utilisés puissent offrir une grande flexibilité et une grande capacité d'adaptation sans atteindre pour autant des coûts disproportionnés par rapport au domaine d'utilisation.

Face à ces nouveaux besoins, les systèmes de gestion traditionnels apparaissent trop lourds et trop complexes. En effet, bien que de nombreuses plates-formes disponibles soient développées en utilisant des architectures ouvertes ou distribuées, dans la pratique, leurs limitations en termes d'extensibilité opérationnelle ou fonctionnelle sont fortement ressenties par les utilisateurs [Der97]. En terme opérationnel, il est reconnu que l'accroissement de la taille d'un réseau géré impose au mieux l'augmentation des capacités de traitement des systèmes sur lesquels sont installées les plates-formes de gestion ainsi que l'augmentation des capacités de transfert des données, et au pire le changement des systèmes existants pour des systèmes plus puissants. Fonctionnellement parlant, les faibles niveaux d'abstraction utilisés par les systèmes d'information sous-jacents ont pour conséquence d'entraîner des délais et des coûts importants lorsqu'il s'agit de développer de nouveaux services, difficiles à admettre par les clients.
De nouvelles technologies, comme CORBA, ont été introduites pour supporter la distribution des applications de gestion à travers le réseau, des méthodologies comme ODP ont été proposées pour faciliter la conception des applications distribuées en offrant des niveaux d'abstraction bien définis (view points), mais les résultats sont encore limités et il est toujours problématique de développer des applications de gestion flexibles, économiques, et insensibles au facteur d'échelle.

I.1.2 Motivation pour une nouvelle approche

Ces résultats mitigés sont-ils pour autant suffisants pour motiver la recherche d'une nouvelle approche? Il faut noter qu'un frein important à la recherche de nouvelles approches est lié aux lourds investissements consentis par les industriels pour le développement, la standardisation, et la mise en place des approches objet, des technologies objet, et des méthodologies objet, ce qui les incitent avant tout à poursuivre leur recherche dans ce même domaine.

C'est pourquoi tout en notant les progrès que représente l'apport des technologies objet notamment pour le développement du TMN (Telecommunications Management Network), [HD98] identifient trois raisons principales pour justifier leurs recherches d'une nouvelle approche:

- la trop grande longueur des cycles de développement
- le besoin important de personnel hautement qualifié
- des architectures monolithiques faiblement extensibles

D'autres recherches menées pour répondre aux problèmes rencontrés par le développement de systèmes de gestion de réseaux, problèmes qui ne sont que partiellement et difficilement résolus par l'évolution des technologies objet, ont montré que les solutions devraient intégrer les concepts de:

- délégation de tâches de gestion, pour éviter les goulots d'étranglements qui se forment autour des systèmes centralisés [GY95][Ane99]
- distribution de l'intelligence, afin de diminuer la complexité des applications centralisées, et d'augmenter la réactivité du système de gestion [Goy94]
- modularité des composants opérationnels afin de s'adapter facilement à l'évolution des équipements et des services [Deri97]

[Woo99] récapitule les paradigmes apparus depuis le début des années 1980 et note que si le développement orienté objet a été le seul à s'imposer, un autre paradigme semble émerger depuis quelques années: celui des agents intelligents et systèmes multi-agents.

Si nous définirons plus précisément au cours de cette thèse ce que sont les agents intelligents, l'intérêt que nous leur portons ici est dû avant tout à leurs propriétés que [Wei99] et [BG99] présentent comme étant les suivantes:

- Vitesse et efficacité – Les agents peuvent opérer d'une manière asynchrone, en parallèle, et directement sur les équipements, ceci ayant pour effet un accroissement de la vitesse de l'ensemble.
- Robustesse et fiabilité – La défaillance d'un ou de plusieurs agents ne rend pas nécessairement l'ensemble du système inutilisable, parce que d'autres agents disponibles dans le système peuvent prendre le relais.
- Extensibilité et flexibilité – Le système peut accepter des problèmes de taille croissante en ajoutant de nouveaux agents et ceci sans affecter les capacités opérationnelles des autres agents.
- Faible coût – Le système multi–agent est plus économique qu'un système centralisé, puisqu'il est composé de sous-systèmes d'un faible coût unitaire.
- Développement et réutilisabilité – Les agents peuvent être développés individuellement par des spécialistes, l'ensemble du système pouvant être facilement testé et maintenu sans une longue phase d'intégration, et il est possible de configurer et réutiliser les agents dans différents scénarios d'applications.

Au vu de cette liste de propriétés, les agents intelligents et systèmes multi-agents[1] semblent résoudre non seulement les problèmes des systèmes de gestion de réseaux que nous avons évoqués mais aussi apporter plus.

I.1.3 Objectif de la thèse

S'il existe de plus en plus de publications concernant des recherches menées autour du paradigme agent dans le cadre de la gestion de réseaux, celles-ci réfèrent principalement le développement d'agents pourvus de propriétés restreintes. Ces agents sont ensuite utilisés comme solution ad hoc, et comme nous le verrons par la suite, il n'existe actuellement aucune technologie agent disponible permettant de concevoir et donc d'évaluer ce qu'est réellement un système multi-agents. En effet, contrairement au paradigme objet qui a été clairement et précisément identifié par des concepts comme celui de classe, d'héritage, de polymorphisme, le paradigme agent n'a pas encore dégagé les concepts qui peuvent faire de lui une technologie indépendante et lui ouvrir les portes du monde industriel.

Cette thèse a un objectif double: rechercher ce que pourra apporter la technologie agent à la gestion de réseaux, et déterminer les bases nécessaires au développement de cette nouvelle technologie.

Pour mener à bien ces objectifs nous avons adopté la démarche suivante:

3. Recherche des véritables obstacles rencontrés par les systèmes de gestion actuels
4. Analyse de l'évolution des technologies couramment utilisées et de leur aptitude à résoudre les problèmes
5. Etude des principales origines de l'approche agent et construction d'une liste de propriétés expérimentées correspondant aux besoins identifiés dans le cadre du développement d'applications de gestion de réseaux.
6. Développement d'une architecture à même de supporter ces propriétés
7. Validation de notre approche au travers de deux études se rapportant à des applications de gestion de réseaux
8. Analyse des résultats et mise en relief des avantages essentiels qui permettraient d'imposer cette nouvelle technologie.

I.2 Contributions

Ces dernières années, de nombreuses publications ont été faites sur l'intérêt de la technologie agent. Si parmi ces publications beaucoup correspondent à des études

[1] Dans un souci de simplification nous utiliserons lorsqu'il n'y a pas d'ambiguïté le terme agent à la place de celui d'agent intelligent.

théoriques sur le comportement individuel et social[2] des agents, que d'autres correspondent à des approches expérimentales du paradigme agent, d'autres sont par contre présentées comme issues de technologies agent existantes. La conséquence de ces différentes perception est qu'il existe maintenant une confusion lorsque l'on parle de technologies agent, certains étant convaincus qu'il est dès maintenant possible de développer des applications multi-agents sur la base de ces technologies. Notre première contribution dans ce mémoire est de montrer qu'il n'existe encore que des ébauches d'une technologie agent par opposition à la technologie objet, et donc en supprimant toute ambiguïté à ce propos, de laisser ouvert la possibilité d'identifier la technologie agent comme une nouvelle technologie à part entière.

Notre deuxième contribution est ici de montrer que les concepts utilisés dans le paradigme agent se situent au carrefour des problèmes soulevés par la gestion de réseaux, et de l'évolution des technologies utilisées dans cette même gestion de réseaux.

Une troisième contribution est apportée par la spécification d'une architecture générique d'agent qui ouvre la possibilité de concevoir un système multi-agents comprenant l'ensemble des propriétés annoncées et attendues. Cette architecture nous a permis de mettre en valeur les concepts importants d'une future technologie "agents intelligents" pour la gestion de réseaux en sélectionnant ce qui était du domaine de l’envisageable du domaine de la spéculation.

Enfin nous avons contribué à démontrer qu'un système multi-agents pouvait être plus facilement fiabilisé et plus flexible qu'un système de conception traditionnelle, en étant capable de s'adapter automatiquement aux modifications de son environnement.

Notre dernière contribution consiste à avoir éclairé les prochaines recherches qui devraient permettre au paradigme agent de s'imposer comme une technologie mûre et indépendante.

I.3 Plan

Le chapitre II est consacré à l'étude de l'évolution de la gestion de réseaux. Il s'agit de rechercher les orientations qui ont guidé les recherches dans ce domaine, en étudiant les principaux standards qui ont été développés, et les technologies utilisées. Nous y constatons que ces standards et ces technologies ne sont plus adaptés à la révolution que connaît depuis quelques années le monde des réseaux. Puis au travers des recherches menées pour diminuer l'écart entre les besoins et les possibilités techniques, nous identifions les principaux problèmes reconnus dans ce domaine.
La fin de ce chapitre est une projection en avant dans l'évolution des technologies utilisées ou développées dans la gestion de réseaux, pour constater que celle encore balbutiante des agents intelligents devrait représenter l'aboutissement logique des recherches en cours.

[2] Voir le chapitre III pour la présentation du concept social des agents

Le deuxième chapitre est dévolu à la présentation du paradigme "agent intelligent". Dans celui-ci, nous nous intéressons principalement aux recherches ayant été menées dans le cadre de la gestion de réseaux et de services. Pour mieux cerner cette nouvelle technologie nous y exposons rapidement ses origines, puis les principales propriétés qui sont généralement présentées comme intégrées dans le paradigme des agents intelligents. Ensuite nous décrivons les limites des approches proposées dans le cadre de la gestion de réseaux, et nous posons la question de savoir si les travaux mis en avant reflètent réellement le potentiel de cette nouvelle technologie.
La fin de ce chapitre est un constat sur le fait qu'actuellement, le paradigme agent intelligent est avant tout issu d'une somme d'expérimentations, et que l'intérêt qu'il présente ne peut être que difficilement cerné, tant qu'il n'existe pas une technologie agent à part entière.

Il s'agit alors de se demander ce que doit comporter une véritable technologie agent, et c'est le but du quatrième chapitre de ce mémoire. Nous y présentons une architecture originale que nous avons créée dans le but de développer des systèmes multi-agents comme applications de gestion de réseaux. Dans un premier temps, nous exposons les principes que nous devrons intégrer dans cette architecture, puis nous présentons l'architecture proprement dite.
Nous détaillons ensuite les deux principales parties de cette architecture qui sont la partie délibérative et la partie réactive. L'approche que nous soutenons dans cette thèse étant une approche prospective, c'est donc avec une vision "macroscopique" que sont exposées ces deux couches de l'architecture. Les principaux composants de l'architecture sont présentés ainsi que les principes de fonctionnement.

Dans le chapitre suivant nous abordons le problème de la méthodologie utilisable pour développer un système multi-agents. Nous y développons les notions de bases applicables à la conception d'un tel système, et nous y étudions les quelques propositions ayant été formulées dans ce domaine. Nous expliquons comment l'architecture que nous proposons influence l'interprétation de ces notions, et nous donnons les définitions adaptés à notre architecture ainsi que quelques relations utilisées par l'agent intelligent générique pour son fonctionnement.
Nous expliquons dans ce chapitre V l'importance de la perception organisationnelle d'une l'application, où rôles et missions présentent des abstractions de plus haut niveau que ce que peut capturer et donc décrire l'approche objet.

Puis nous décrivons dans la dernière partie les études de cas que nous avons conduits pour valider l'architecture et l'approche agent intelligent pour la gestion de réseaux. Deux études de cas sont détaillées, chacune ayant pour objectifs de démontrer à la fois certaines propriétés propre au paradigme agent intelligent, et à la fois l'intérêt de ces propriétés dans le cadre de la gestion de réseaux.
La première étude concerne la fiabilisation d'un système à base d'agents intelligents, et ses capacités d'auto adaptation. Les propriétés "agent" de délégation et de coopération sont traitées, couvrant implicitement les propriétés recherchées dans la gestion de réseaux qui sont la flexibilité et l'extensibilité des applications.

La deuxième étude de cas traite plus particulièrement les problèmes de l'hétérogénéité des équipements et démontre l'efficacité d'un système multi-agents pour les opérations complexes de configurations de PVC dans un réseau ATM notamment par rapport à l'approche agent mobile.
A la fin de ce chapitre VI, une analyse synthétique des résultats est effectuée où nous défendons l'idée que des propriétés comme la modification dynamique de comportements d'une application multi-agents représentent des atouts majeurs qui peuvent permettre la création d'une véritable technologie agent.

II Evolution de la gestion de réseaux

En un peu moins de vingt ans, l'informatisation et les systèmes informatiques ont envahi notre quotidien. Des premières calculettes à la domotisation, en passant par internet et le commerce électronique, les informations que nous pouvons traiter, et auxquelles nous avons maintenant accès, sont considérables, et ceci principalement grâce au réseaux. Mais de nos jours même si petites et moyennes entreprises ont besoin d'un réseau non seulement pour survivre mais aussi pour se développer, elle n'ont pas forcément les ressources humaines, techniques et financières pour gérer celui-ci.
Les solutions centralisées que nous trouvons encore actuellement sont comme nous le verrons plus tard beaucoup trop onéreuses et complexes pour un grand nombre d'entreprises, et pas assez flexibles pour des entreprises plus grosses. C'est pourquoi l'importance de la distribution dans le domaine de la gestion de réseaux va en s'amplifiant jusqu'à devenir le thème central des plus importantes conférences sur la gestion de réseaux, comme Integrated Network Management dont le thème était pour l'année 1999 «Distributed Management for the Networked Millennium».
Des chercheurs se penchent depuis le milieu des années 1980 sur l'intérêt de la gestion de réseaux distribuée [YGY91], et en 1994 [Slo94] explique que l'on a enfin pris conscience qu'il fallait automatiser et distribuer la gestion de réseaux pour l'adapter à la réalité de la distribution des systèmes à gérer. La gestion centralisée n'est pas pour autant complètement rejetée: par exemple [Pra95] discute des avantages et inconvénients de la distribution par rapport à la centralisation pour les systèmes de gestion de réseaux, sans pour autant se prononcer sur l'avenir de l'un et de l'autre. Néanmoins la tendance est inéluctable, et [MZH99] justifie l'avancée de la distribution comme étant liée au moins à deux facteurs. Le premier est que la gestion distribuée est moins sensible aux problèmes d'intéropérabilité, d'extensibilité, de flexibilité et de robustesse. Le deuxième facteur est

que les technologies de distribution qui étaient balbutiantes il y a quelques années, sont maintenant matures et de plus continuent à évoluer.

Nous proposons d'étudier dans ce chapitre comment se déroule de cette évolution, afin de dégager les concepts qui seront les acteurs essentiels de la gestion de réseaux de demain.

Pour cela, nous avons divisé le chapitre en cinq parties. Tout d'abord une courte revue des principaux standards utilisés pour le développement des plates-formes de gestion, en incluant un point de vue sur la réalité de l'utilisation de ces plates-formes. Nous détaillons ensuite les principaux problèmes que l'on rencontre aussi bien au niveau de la conception des systèmes de gestion qu'au niveau opérationnel. Dans la troisième partie nous étudions l'évolution des technologies de développement, pour expliquer l'importance de ces technologies dans l'évolution des systèmes de gestion. Enfin nous analysons l'évolution des protocoles et outils de gestion de réseaux avant de terminer par une discussion sur l'importance des nouveaux paradigmes.

II.1 Problématique de la gestion de réseaux

Dans l'introduction de cette thèse nous avons souligné l'importance grandissante des réseaux dans notre vie quotidienne. Pour [HAN99], la fiabilité des services de communication offerts par les réseaux est devenue non seulement vitale pour la science mais aussi pour la société. [Slo94] nous donne quelques exemples de la paralysie économique qui surviendrait si les réseaux d'entreprises venaient à défaillir, comme dans le secteur bancaire où les transactions ne pourraient plus avoir lieu. Certains ont déjà pu vivre le stress de la panne du réseau informatique dans un supermarché, ou même dans un petit commerce équipé de quelques terminaux reliés à un serveur, et qui se trouvent l'un comme l'autre incapables d'émettre une facture pendant la durée de la panne. La gestion de réseaux est donc cruciale pour assurer la continuité des services.

II.1.1 Définition des besoins

Mais devant la diversité des réseaux, la question que l'on doit se poser maintenant est: les besoins en termes de gestion sont-ils les mêmes quel que soit le ou les réseaux à gérer.
En effet le réseau utilisé par un particulier à son domicile, où sont connectés ordinateurs, chauffage, éclairage, systèmes de sécurité, grille-pain ou lave-linge, n'a rien a voir avec celui d'un opérateur télécom qui doit contrôler en temps réel des centaines voir des milliers d'équipements. De plus la perception du réseau et de son système de gestion va varier suivant les personnes. Un responsable financier dans une entreprise voudra que le réseau soit peu coûteux aussi bien à l'achat qu'à la maintenance, tandis qu'un responsable de la gestion du réseau voudra une interface simple, un environnement de configuration homogène et évolutif en dépit de la diversité des équipements.
[TER92] nous propose une liste de besoins exprimés par les entreprises:

- contrôler les atouts stratégiques des entreprises

- contrôler la complexité
- améliorer les services
- répartir l'utilisation des ressources
- réduire le temps d'indisponibilité
- contrôler les coûts
- contrôler la sécurité.

La liste correspondante pour un opérateur s'accroît considérablement [TER98] et comprend entre autre:

- capacité à surveiller en temps réel les composants des réseaux
- capacité à traiter un très grand nombre d'événements, d'alarmes, de messages dans un court intervalle de temps
- possibilité de distribuer les fonctions de gestion et les bases de données
- accès en temps réel aux informations de comptabilité
- accès en temps réel des informations sur la performance
- capacité de corrélation d'alarmes
- possibilité d'utiliser des politiques de gestion
- intégration de technologie de gestion web
- support de structure client serveur avec une grande flexibilité de distribution de tâche
- très grande flexibilité dans les capacités de configuration
- personnalisation des applications et des interfaces des OSS (Operations Support Systems)
- etc

Néanmoins, et pour simplifier, l'utilisateur d'un réseau quel qu'il soit va vouloir exprimer ses besoins en terme de qualité de service comme par exemple:

- temps de réponse court
- disponibilité permanente (pratiquement 99,9%)
- fiabilité totale pour garantir la protection des données contre les erreurs de transfert,
- transferts sécurisés
- grande facilité d'utilisation

L'expression de la plupart de ces besoins a été depuis longtemps l'objet d'analyses et de formalisations par des organisations internationales comme le CCITT et l'ISO pour déboucher sur des standards permettant le développement de systèmes de gestion (confer section suivante). De son côté, [Slo94] mentionne que les premiers travaux sur la gestion ont porté sur les trois domaines suivants:

- gestion des télécommunications
- gestion des systèmes
- gestion de réseaux

Ces découpages ont souvent été revus notamment pour répondre efficacement à la demande d'intégration émanant de grandes entreprises, dont l'objectif de management correspond d'après [HAN99] à la mise en place de "toutes les mesures assurant l'utilisation effective et efficace d'un système et de ses ressources, en accord avec les objectifs de l'entreprise".

Malgré tout, si le résultat de ces travaux a été utilisé par les opérateurs télécom c'est un autre standard approuvé en 1988 par l'IAB (Internet Activities Board) pour gérer les équipements du réseau Internet qui s'est rapidement imposé pour la gestion des réseaux locaux ou LAN (Local Area Network) et même des réseaux WAN (Wide Area Network). Il s'agit de SNMP (Simple Network Management Protocol), un protocole de gestion initialement développé pour le protocole IP utilisé sur Internet. Une des raisons de ce succès est la simplicité de l'approche proposée, simplicité qui correspond à la première préoccupation des utilisateurs et des fournisseurs d'équipements [Sta96].

II.1.2 Modèle d'une architecture d'un système de gestion de réseaux

L'approche retenue dans la plupart des standards, que nous détaillerons plus loin, consiste à définir au minimum un système d'information permettant de décrire les objets du réseau à gérer et des applications client-serveur (manager-agent) servies par un protocole de communication.

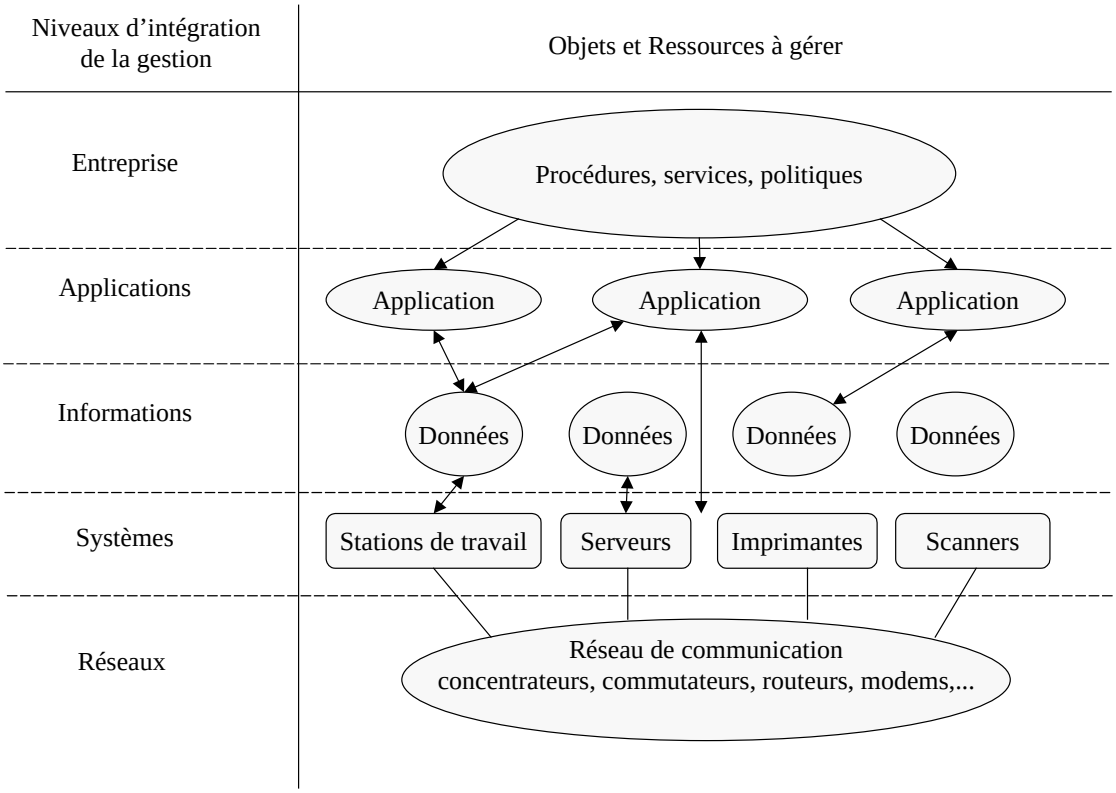


figure 1 Modèle d'architecture d'un système de gestion de réseaux

Les informations de gestion décrites par le système d'information et manipulées par les interactions entre manager et clients sont ensuite présentées à l'utilisateur grâce à des applications de présentation ou interfaces homme-machine (voir figure 1).

Les systèmes d'information sont spécifiés par un SMI (Structure of Management Information) qui précise le cadre général de la définition des informations de gestion comme le langage de description des objets, ou l'organisation des informations (modèle d'information). Bien qu'il existe de grandes différences entre les standards sur la définition de ce que représentent les objets gérés, on pourra retenir que ceux-ci ont été conçus pour reporter des valeurs de données de bas niveau.

Ces informations sont accessibles par le manager au travers de requêtes qu'il transmet à l'agent au moyen d'un protocole (exemple: CMIP). Localement, l'agent dispose d'une base de données appelée MIB (Management Information Base) qui contient l'ensemble des informations du système géré qu'il va pouvoir utiliser.

II.1.3 Indépendance des plates-formes

La recherche de la simplicité est une préoccupation toujours présente chez les utilisateurs des systèmes de gestion, et pour cette raison de nombreux outils et applications de gestion "indépendants" sont utilisés en complément ou en parallèle des plates-formes de gestion. Ces deux dernières années, c'est la simplicité des technologies et les interfaces web qui semblent avoir conquis les gestionnaires de réseaux. En effet les technologies web ont pour caractéristique d'être indépendantes des systèmes, et il est possible de proposer grâce à cela une présentation identique des informations de gestion (cartes, images, textes, etc.) aussi bien sur un PC que sur une station de travail. Cette indépendance est essentiellement le fait des navigateurs (browsers en anglais) qui permettent d'afficher des pages utilisant des langages comme HTML (HyperText Markup Language) ou XML (Extensible Markup Language), et qui possédent des machines virtuelles permettant l'exécution de programmes écrits en Java. La force de ce langage est avant tout de rendre le code généré par son compilateur indépendant des systèmes d'exploitation. Les enquêtes publiées par les journaux spécialisés indiquent que ces technologies sont considérées comme immatures par les administrateurs de réseaux, mais ces derniers apprécient la possibilité qu'elles offrent de faire de l'administration à distance.
L'importance de l'indépendance des plates-formes et des implémentations a été bien comprise par le DMTF (Desktop Management Task Force) qui a retenu et poursuivi les travaux de WBEM sur CIM (Common Information Model) pour décrire son modèle d'information. L'objectif poursuivi est de proposer avec CIM l'intégration de tous les modèles existants

II.2 Les standards des systèmes de gestion

L'objectif de cette section est de faire une revue critique des principaux standards de gestion de réseaux, afin de comprendre quels sont les besoins qui ne sont pas encore couverts par ceux-ci. Pour cela nous commençons par rappeler brièvement ce que l'on attend de la standardisation des systèmes de gestion de réseaux, puis après avoir revu les avantages et les inconvénients des principaux standards, nous évoquons la réalité des implémentations vécue par les utilisateurs.

Les standards ont avant tout l'avantage de permettre un développement séparé et admis par tous des composants d'une application de gestion, en offrant une décomposition explicite des services, architectures et protocoles.
Dans les années 1980, l'Organisation Internationale des Standards (OSI) a proposé une norme pour l'architecture de gestion de réseaux connue sous le nom d'Interconnexion de Systèmes Ouverts (ISO). Cette initiative a été suivie par l'Union Internationale des Télécommunications (ex CCITT en 1985), avec le début des travaux sur les standards Réseaux de Gestion des Télécommunications (RGT) appelés Telecommunications Management Network (TMN) en anglais. L'objectif de ces derniers étant de définir une architecture fonctionnelle pour un système de gestion de réseaux de télécommunications souple, complet et évolutif [Maz97].
Les standards décrits dans le chapitre suivant représentent l'aboutissement de travaux menés par de nombreuses équipes pour résoudre ou répondre à la difficulté de mettre au point ces systèmes et leur permettre d'interopérer. Cette interopérabilité est importante non seulement pour relier les systèmes de gestion entre eux, mais aussi pour dégager les opérateurs de leur dépendance vis à vis de leurs fournisseurs de systèmes de gestion, en leur permettant de s'équiper de plates-formes différentes.

Nous nous attachons dans un premier temps à démontrer que le long processus de création de standards n'est plus adapté au dynamisme du monde des réseaux, et que l'avancé des technologies influence et bouscule même notre vision de la gestion de réseaux. Néanmoins les standards sont indispensables car on ne peut pas communiquer si l'on ne se met pas d'accord sur les protocoles et les structures d'information à échanger.
C'est pourquoi les regroupements d'efforts pour mettre au point les techniques d'échanges d'informations sont indispensables car plus économiques, en évitant d'avoir à créer des interfaces inutiles et complexes. C'est le constat que l'on peut faire en observant la percée des consortiums comme l'OMG qui avec CORBA impose la modification des architectures des plates-formes de gestion, et fait évoluer les concepts objets en intégrant la notion de transparence de localisation.
Nous tâcherons aussi de démontrer que cette évolution conduit inéluctablement vers l'autonomie des objets, au travers des composants (souvent de simples assemblages d'objets) auxquels on rajoute certaines propriétés.

II.2.1 ISO

Cette section décrit l'approche de la gestion de réseaux préconisée et normalisée par l'ISO. Il s'agit ici de mettre en avant l'architecture du système de gestion pour déterminer les limites de celle ci.
Le premier standard qui décrit la gestion OSI est le modèle de référence OSI (OSI Reference Model) dérivé du Cadre de Gestion OSI (OSI Management Framework X700).

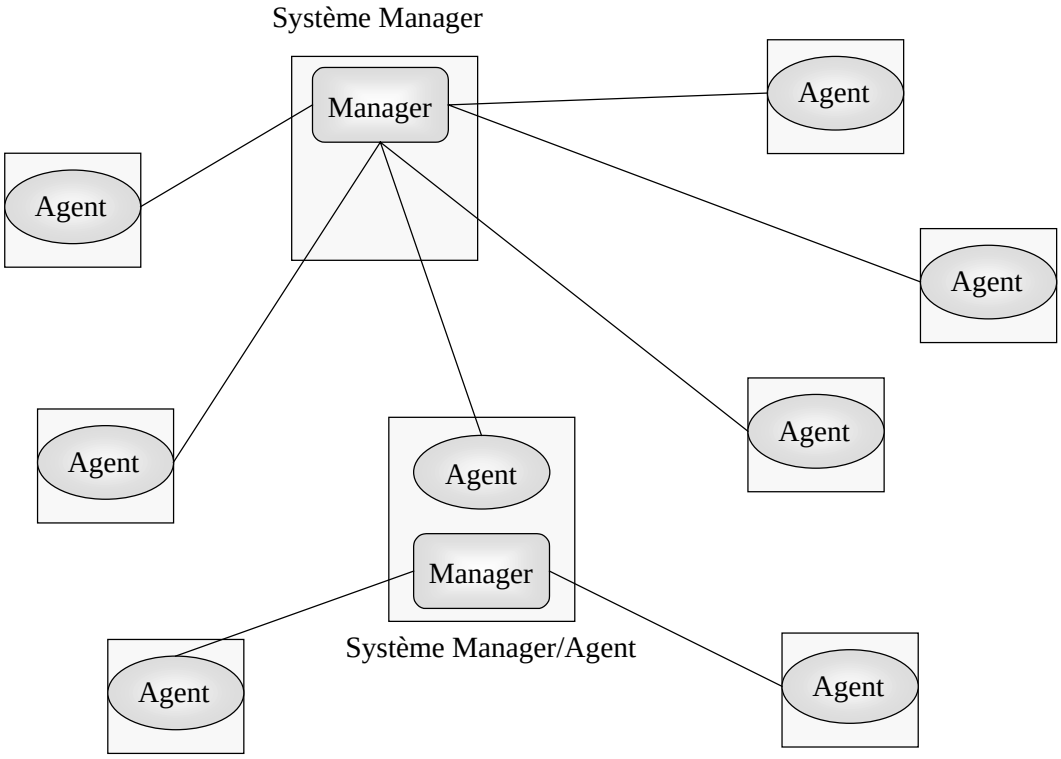


figure 2 OSI modèle hiérarchique

Cette norme ne faisant pas l'unanimité, un autre document a été produit « OSI Systems Management Overview (X701) » contenant plus d'informations et distinguant les aspects suivants :

- Information
- Organisation
- Fonction
- Communication

L'aspect *information* décrit les objets gérés, représentant la perception qu'a le système de gestion des ressources, mais aussi les objets de support de gestion qui sont des objets participant aux activités de gestion, un exemple étant l'enregistrement d'un historique.

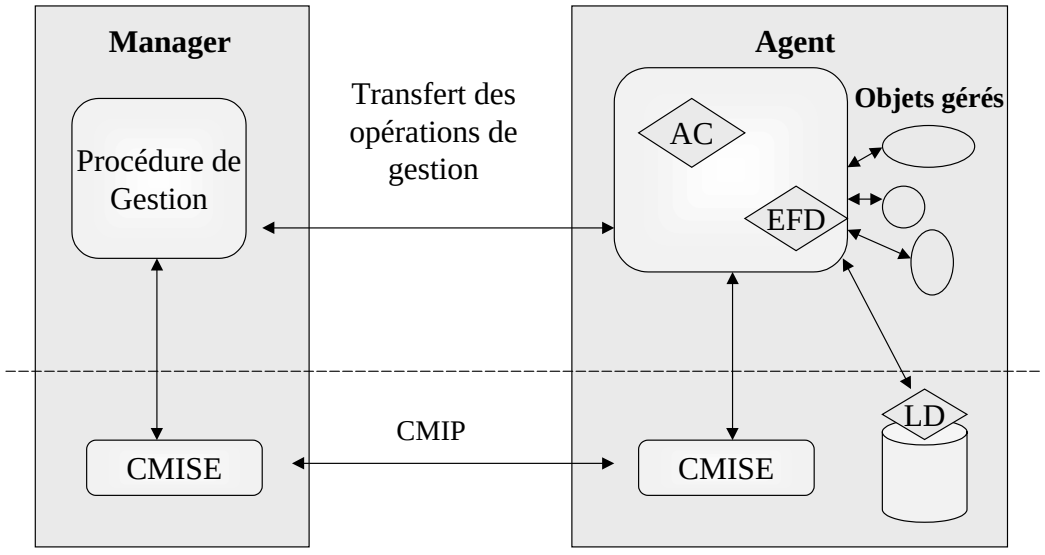
Cette description est complétée par une série de standards complémentaires dont le tout constitue le SMI (Structure of Management Information)

L'aspect *organisation* décrit le concept manager-agent correspondant à l'approche centralisée adoptée par l'ISO , avec la possibilité de découper l'environnement de gestion en domaines de gestion. L'idée étant de faciliter la gestion en offrant des vues fonctionnelles, géographiques, ou technologiques des réseaux. [Pras95] note que les travaux sur ces domaines de gestion sont toujours en cours.
Mais un système pouvant à la fois jouer le rôle d'un manager ou d'un managé, il est alors possible de concevoir des organisations hiérarchiques (figure 2), ou en chaînes.

L'aspect *fonction* comprend les cinq aires fonctionnelles de gestion arrêtées en décembre 1988 qui sont :

- Gestion de fautes
- Gestion de configurations
- Gestion de comptes
- Gestion de performances
- Gestion de sécurité

Ces aires fonctionnelles ont été complétées après 1988 par des standards définissant les fonctions élémentaires de gestion, dont l'ensemble constitue le SMF (Systems Management Functions).



AC: Acces Control
EFD: Event Forward Descriminator
LD: Log Discriminator

figure 3 Modèle de gestion OSI

Enfin l'aspect *communication* décrit les services d'échange d'informations dont la décomposition comprend les SMAEs (Systems Management Application Entities) et les ASEs (Application Service Entities). Ces services utilisent le protocole CMIP (Common Management Information Protocol).

Conclusion

De ce survol rapide de la structure de gestion de réseaux proposée par l'OSI, nous pouvons déjà relever quelques problèmes importants.

Parmi ces problèmes [Pras95] soulève des incohérences entre les différents standards OSI. Notamment celle qui existe entre le modèle de référence OSI qui spécifie que chaque couche possède son propre service de gestion, alors que le modèle de gestion OSI au travers des SMAES autorise la manipulation des objets de n'importe quelle couche.
Ce problème est plus inquiétant lorsque l'on remarque qu'une couche de protocole qui est gérée est aussi utilisée par l'application qui la gère. Ainsi, si une couche N est défaillante, il est impossible à l'application de gestion (SMAE) d'envoyer un rapport d'alarme. Ces problèmes bien qu'anecdotiques, traduisent la difficulté qu'il y a à normaliser d'une manière cohérente la gestion de réseaux, chaque groupe de normalisation devant connaître l'ensemble, et en détail, des travaux produits par les autres groupes.

Mais en dehors de ces problèmes de cohérence, d'autres problèmes plus importants ressortent actuellement du fait de la libéralisation des télécoms et de l'explosion des réseaux d'entreprises. Ces problèmes sont :

- **Complexité** de l'architecture OSI visible au nombre et au volume des standards. Cette complexité était supportable par les opérateurs de télécoms grâce aux revenus que leur conférait leur position de monopole, et à la lente évolution des réseaux. Cette complexité est maintenant rejetée par les nouveaux opérateurs et les entreprises de taille moyenne. Lors du dernier salon Télécom 99 à Genève, M. Yoshio Utsumi Secrétaire Général de l'ITU (ex CCITT) a clairement annoncé le risque de voir disparaître les organismes de standardisation, ceux-ci ne ralliant plus qu'une partie minoritaire des opérateurs.
- **Coût,** corollaire de la complexité que nous venons d'évoquer, et qui est un facteur déterminant pour les entreprises qui préfèrent opter pour des solutions plus économiques. Certaines entreprises affirment même que des solutions ad hoc sont plus économiques pour une fonctionnalité équivalente que les plates-formes traditionnelles utilisant les standards OSI.
- **Délais** nécessaires à la mise au point des standards OSI, et qui sont beaucoup trop longs. Ce sont ces délais qui ont permis la percée des solutions SNMP, et ceux-ci apparaissent encore plus importants du fait de l'évolution rapide des technologies réseaux.

- ▹ **Rigidité** de l'architecture qui rend impossible l'adaptation des plates-formes aux changements de structure des entreprises, et donc de leurs réseaux. [Framework Fraud. By DataCom]
- ▹ **Manque de garantie** du comportement global des applications de gestion. En effet si le modèle d'information est bien établi par une syntaxe et une sémantique statique, le comportement dynamique des objets est quant à lui décrit en langage naturel non formel et donc par nature non vérifiable [Maz97].
- ▹ **Approche centralisée**. L'architecture OSI par nature non distribuée ne peut satisfaire les demandes d'extensibilité et répond mal aux demandes d'accroissement des performances [CNK96].

Parmi les conséquences des problèmes que nous venons d'évoquer, il faut noter le changement d'orientation de l'ITU-T, branche de standardisation des télécommunications de l'ITU (International Telecommunication Union), et dont le groupe SG4[3] envisage de ne conserver CMIP que pour les communications avec les éléments de réseaux, tandis que CORBA serait utilisé pour les communications dans les couches plus hautes [Sid99] . Plus anecdotique, CMIP disparaît du titre de certaines rééditions des protocoles traitant de gestion de réseaux [Sta93] [Sta96].

II.2.2 TMN

Les RGT (Réseaux de Gestion des Télécommunications) ou TMN en anglais (Telecommunications Management Network) ont été développés pour gérer les équipements, les réseaux et les services de télécommunications offerts par les opérateurs.

Le TMN établit une séparation logique entre la partie opérationnelle de la télécommunication et la partie gestion (figure 4), ces deux parties étant servies par deux réseaux.

Un abonné quelconque d'un réseau téléphonique doit pouvoir joindre n'importe quel autre abonné, qu'il soit sur le même réseau ou sur un autre. Donc par nature les systèmes de gestion TMN sont voués à s'échanger des informations. Aussi l'architecture TMN prévoit le support et l'interconnexion d'OSS (Operation Support Systems) provenant de différents vendeurs, utilisés par différents opérateurs.

Ces opérateurs, contrairement aux administrateurs de réseaux d'entreprise ou universitaires, tirent directement leurs revenus de l'utilisation des services liés aux réseaux télécom. Pour cela, les notions de gestion de services et de commerce ont été dès le début intégrées dans cette architecture de gestion. L'UIT (Union Internationale des Télécommunications) qui est le principal promoteur de cette architecture a émis une série de normes définissant celle-ci.

[3] Le groupe SG4 s'occupe entre autre de la définition de l'architecture TMN (M.3010) et des protocoles de communications de ce même TMN.

C'est la série des normes M. (M.3000, M.3010, M.60, ...) définies par le groupe d'étude SG4.

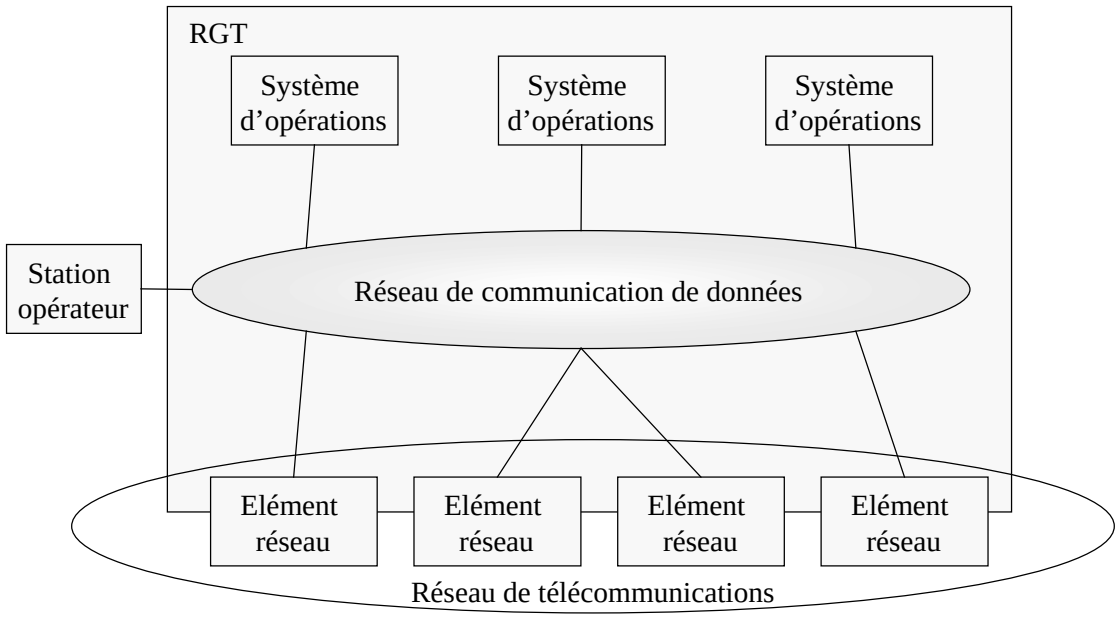


figure 4 Relation entre le TMN et le réseau Télécom

L'architecture générale TMN (M3100) définit des composants de construction de base (ex : basic building blocks) ainsi que leurs mécanismes d'interactions. Ces composants sont regroupés sous trois architectures:

- Fonctionnelle : qui définit cinq blocs fonctionnels ainsi que cinq points de référence
- Physique : qui définit six blocs de construction et leurs interfaces
- Informationnelle : qui est calquée sur le système d'information OSI

Ces architectures servent un modèle de gestion divisé en couches hiérarchiques (Logical Layered Architecture), chaque couche correspondant à une responsabilité différente (figure 5):

- Couche gestion commerciale: comprend la gestion de la planification, des budgets, des objectifs de l'entreprise
- Couche gestion de services: s'occupe de tout ce qui touche à la gestion des services utilisateurs, comme la qualité de service, l'identification et le suivi des problèmes
- Couche gestion de réseaux: s'occupe des équipements en terme de configuration des réseaux, comme le routage, le traitement des erreurs, etc

- ▹ Couche gestion des éléments: s'occupe de toute la gestion des éléments ou groupes d'éléments comme leur charge, leur état, mais aussi de la gestion logique en terme de configuration, par exemple en traitant les demandes contradictoires
- ▹ Couche éléments de réseaux: contient les éléments de réseaux eux-mêmes.

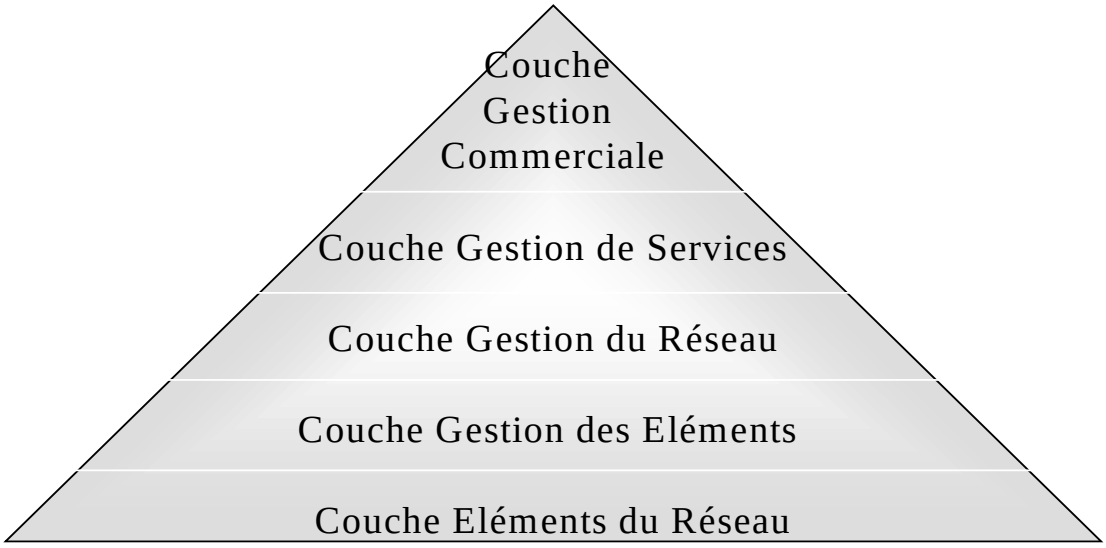


figure 5 Couches de gestion du TMN

Ces différentes couches contiennent des blocs fonctionnels qui sont reliées entre eux grâce aux points de référence, appelés aussi interfaces lorsque les blocs fonctionnels connectés sont situés sur des systèmes distincts (voir figure 6).
Les points de référence permettent de définir les informations qui doivent être échangées entre les blocs fonctionnels ainsi que les protocoles à utiliser.
Ces blocs fonctionnels sont:

- ▹ Operations System. Ce bloc doit offrir l'ensemble des activités de gestion comme le monitoring, la coordination et le contrôle des entités du TMN. Parmi les activités nous retrouvons celles définies dans le cadre de la gestion OSI comme la gestion des fautes, de la configuration, de la comptabilité, des performances et de la sécurité. Mais nous devons trouver la possibilité de gérer la planification des opérations, l'administration, la maintenance et l'approvisionnement des réseaux de communications.
- ▹ Mediation Device. Ce bloc s'occupe du transfert et du prétraitement des informations entre l'OS (Operation System) et les NE (Network

Element). Il a donc non seulement un rôle de conversion de protocoles mais aussi de stockage, de filtrage et de synthèse des informations.

- ▷ Workstation. Ce bloc est chargé de présenter les informations provenant des différents composants actifs du système de gestion TMN.
- ▷ Data Communications Networks. Ces composants TMN sont en charge du transport des informations entre les entités TMN.
- ▷ Network Elements. Ce groupe fonctionnel est implémenté dans les équipements de réseaux eux-mêmes, et présente des interfaces conventionnelles ou non

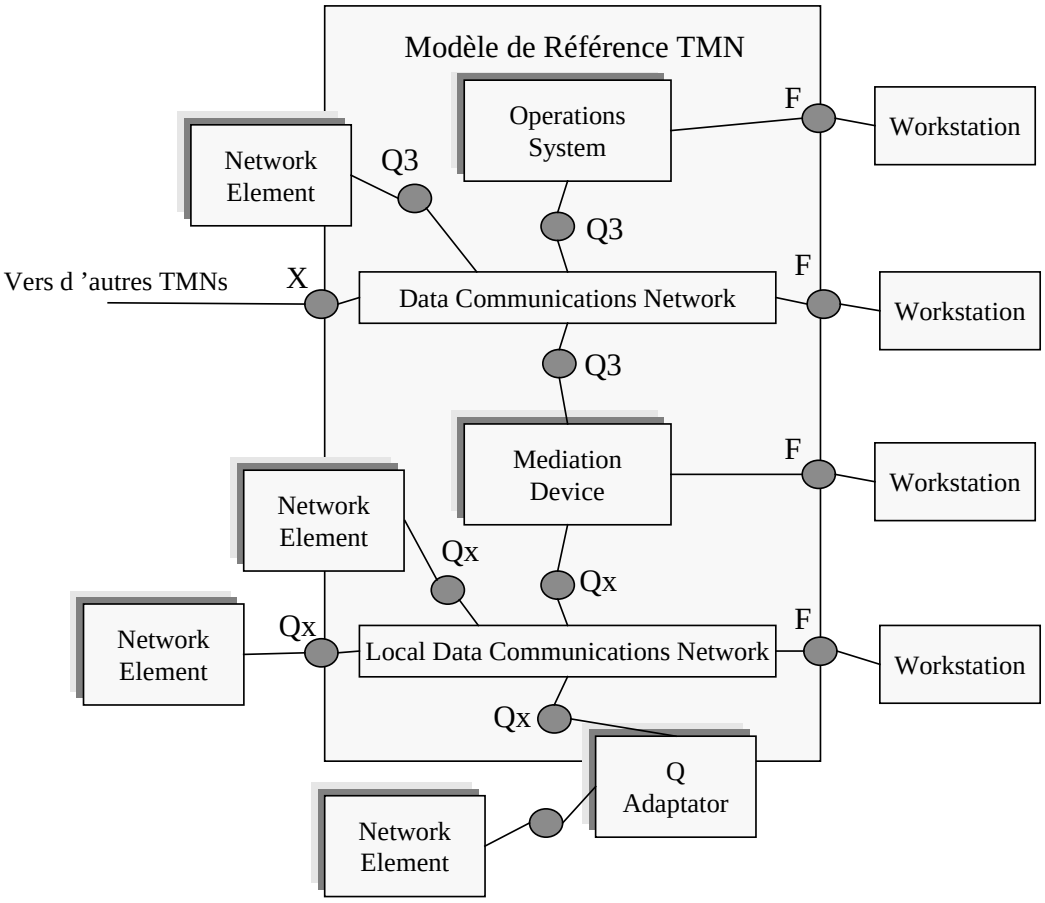


figure 6 Modèle de référence TMN

L'architecture TMN a adopté de l'OSI l'approche orientée objet pour définir le modèle d'information du TMN, et les normes CMISE et FTAM (File Transfert Access and Management) pour échanger les informations de gestion. Plusieurs interfaces de gestion sont définies par la norme M.3300 pour relier entre eux les blocs fonctionnels définis par la norme M.3400 (figure). Actuellement nous pouvons observer une forte volonté de la part des groupes de travail pour dégager l'architecture TMN de l'approche manager-agent héritée de l'OSI vers une approche plus distribuée soutenue par l'OMG. Par exemple,

comme nous l'avons déjà reporté, le groupe SG4 a récemment approuvé l'utilisation de l'architecture CORBA (CORBA) pour l'interface X de la couche de gestion de service, autrefois réservée à CMIP. Et plus généralement un consensus est en train de se former sur l'utilisation de CORBA pour toutes les couches hautes du TMN, ne laissant plus à CMIP que les interactions du système de gestion avec les éléments de réseaux.

Mais si actuellement on commence à trouver quelques plates-formes de Gestion de Réseaux se prétendant conformes à l'architecture TMN, aucune n'implémente complètement celle-ci.

Malgré tout, l'intérêt des standards TMN est triple :

1. Le TMN représente une architecture de gestion de réseaux, intégrant la notion d'hétérogénéité des équipements
2. L'architecture intègre la notion de distribution du fait même de l'obligation qu'ont les opérateurs d'interconnecter leurs réseaux
3. L'architecture propose des niveaux d'abstraction qui sont susceptibles de faciliter le développement et l'utilisation des framework TMN

Conclusion

Il est intéressant de noter que l'ITU a retenu plusieurs concepts utilisés par l'OSI. Parmi ces concepts nous trouvons [Pras95] :

- La relation agent-manager
- L'approche objet « ... la méthodologie TMN utilise les principes des systèmes de gestion OSI et est basée sur le paradigme orienté objet. »
- La notion de domaines de gestion

D'un autre côté, l'architecture TMN introduit des niveaux d'abstraction comme le service et le commerce qui n'existent pas dans le monde OSI, et qui permettent de définir des niveaux de responsabilité.
L'approche par composants, la définition précise des interfaces ou points de références, sont des éléments essentiels pour assurer une certaine flexibilité et indépendance vis à vis des constructeurs. Pourtant tout n'a pas été défini et tout ne peut-être normalisé. La conséquence de cette constatation est que les recommandations établissant la conformité au TMN ne suffisent pas à garantir un remplacement facile des composants, ni une interconnexion immédiate entre eux.
Le TMN est aussi connu pour décrire un système d'information plus complexe que nécessaire. Toujours au sujet du système d'information, si les modèles d'information définissent bien les structures d'information échangées entre les TMN ainsi que le comportement microscopique des objets correspondants, ils ne définissent pas le comportement macroscopique du réseau et donc du service comme un tout [Ter98].

De plus, la complexité de la modélisation des informations échangées par les interfaces X et Q, rend longue et délicate la conception de nouveaux modèles. C'est pour cela qu'aujourd'hui, entre les systèmes de gestion de réseaux des opérateurs, seules sont échangées les informations relatives aux connections de ligne (ex: configurations), la plupart des interactions entre opérateurs impliquant encore des interventions manuelles [TMF99]. Les changements concernant les interfaces que nous avons reportés précédemment, s'ils vont permettre d'améliorer les performances et apporter une certaine souplesse grâce à CORBA, ne changeront rien au problème de complexité du système d'information. Pour diminuer cette complexité, certains proposent l'utilisation d'outils chargés de masquer les détails des implémentations [Ter98].

II.2.3 IETF

A l'opposé des standards OSI et TMN, l'IETF n'avait pas défini d'architecture standard pour la gestion d'internet. L'origine de cet «oubli » peut être trouvé en étudiant l'historique de SNMP, car ce standard avait été proposé en attendant que les travaux menés par l'ISO aboutissent. A cette époque déjà, l'importance croissante du réseau Internet rendait impossible le traitement manuel des configurations et des pannes, et l'utilisation de SNMP est vite devenue indispensable.
Il fallait rendre possible la configuration à distance des équipements de réseaux ainsi que le monitoring de ceux-ci. Pour cela le standard devait décrire à la fois la représentation des informations de gestion de ces équipements, mais aussi un protocole d'accès à ces informations.
L'IAB (Internet Activities Board) a donc adopté une approche pragmatique en adoptant un protocole de gestion simple SNMP (Simple Network Management Protocol), et une représentation des informations MIB (Management Information Base), et plus récemment RMON (Remote MONitoring), outils que nous étudierons un peu plus en détails dans le chapitre suivant.

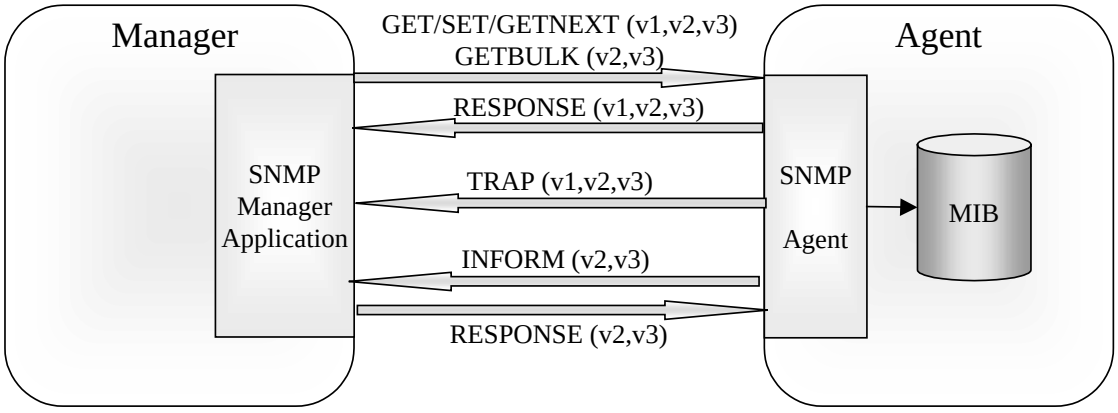


figure 7 Architecture du système de gestion SNMP

Cette architecture définit le système de gestion SNMP comme un système devant contenir:

- ▹ plusieurs noeuds chacun ayant une entité SNMP comprenant des applications servant à répondre aux commandes, et à générer des notifications d'événements, celles-ci ayant accès à l'instrumentation de gestion (traditionnellement appelés agents). Associée à chacun de ces agents, une base de données contient les informations de gestion, c'est la MIB (Management Information Base).
- ▹ au moins une entité SNMP comprenant des applications permettant de générer des commandes et de recevoir des notifications d'événements (traditionnellement appelé manager)
- ▹ un protocole de gestion utilisé pour transporter l'information de gestion entre les entités SNMP

Les principaux reproches qui ont été faits à l'encontre de la première version de SNMP sont:

- la faiblesse du mécanisme de sécurité
- l'utilisation quasi exclusive du mécanisme de polling
- le manque d'efficacité lorsqu'il s'agit de transférer une grande quantité d'informations
- le manque de fiabilité des échanges (basés sur UDP)
- l'absence de relation Manager-Manager

Pour combler ces lacunes, plusieurs versions V2 ont été proposées sans jamais entraîner une adhésion claire de l'ensemble de la communauté. Aussi a-t-il été décidé d'utiliser ces différentes propositions autour d'une vraie architecture "compatible" et plus ouverte avec la version V3 de SNMP [RFC2271]. Cette nouvelle version utilise des concepts objet comme l'encapsulation, et offre aussi la possibilité d'ajouter ou de retirer dynamiquement des modules de gestion.

Si la version V3 de SNMP permet une plus grande souplesse d'utilisation et améliore la sécurité des accès aux données de gestion, elle se base par contre toujours sur un modèle d'information non objet.

C'est-à-dire que des notions comme l'héritage ne peuvent pas être utilisées pour définir de nouveaux objets de gestion, qui restent des données typées. Ces objets restent de même rattachés à l'arbre de nomage dont ils constituent les feuilles. Un des inconvénients du principe utilisé par ce SMI est que les objets ne sont pas identifiés d'une manière unique au niveau du système de gestion, mais uniquement au niveau de l'agent. De ce fait un manager SNMP ne peut pas adresser une requête concernant une opération sur la MIB d'un agent à un autre manager SNMP. SNMP est de ce point de vue dans l'état actuel tout à fait inadapté à la gestion de réseaux distribuée.

Actuellement, la majorité des équipements de réseaux implémentent ces standards (principalement SNMP V1) et sont gérables grâce à eux, ce qui tend à démontrer que les solutions les plus simples sont les plus demandées. Toutefois cette simplicité présente plusieurs revers, un des plus importants étant sa sensibilité au facteur d'échelle. Ce problème, comme nous l'avons noté précédemment, provient du SMI, mais aussi du

modèle client serveur, ou plutôt manager agent, adopté qui oblige à faire un compromis entre la précision de la gestion et les capacités des systèmes de gestion. Ce phénomène est très connu par les gestionnaires de réseaux qui doivent souvent revoir la périodicité des opérations de polling sur les équipements en fonction de la taille et de la charge du réseau.

Pour tenter de résoudre ces problèmes liés à la centralisation du système de gestion, un autre groupe de travail "The Distributed Management Working Group" a été chargé de définir un ensemble d'objets de gestion permettant de distribuer les applications de gestion de réseaux. Les membres de l'IETF ont en effet reconnu que la gestion distribuée est incontournable si l'on veut faire face à la croissance d'internet. Si les premiers travaux [RFC2592] préconisent SNMP comme mécanisme de communication pour distribuer les applications de gestion, d'autres techniques comme HTTP et CORBA sont à l'étude.

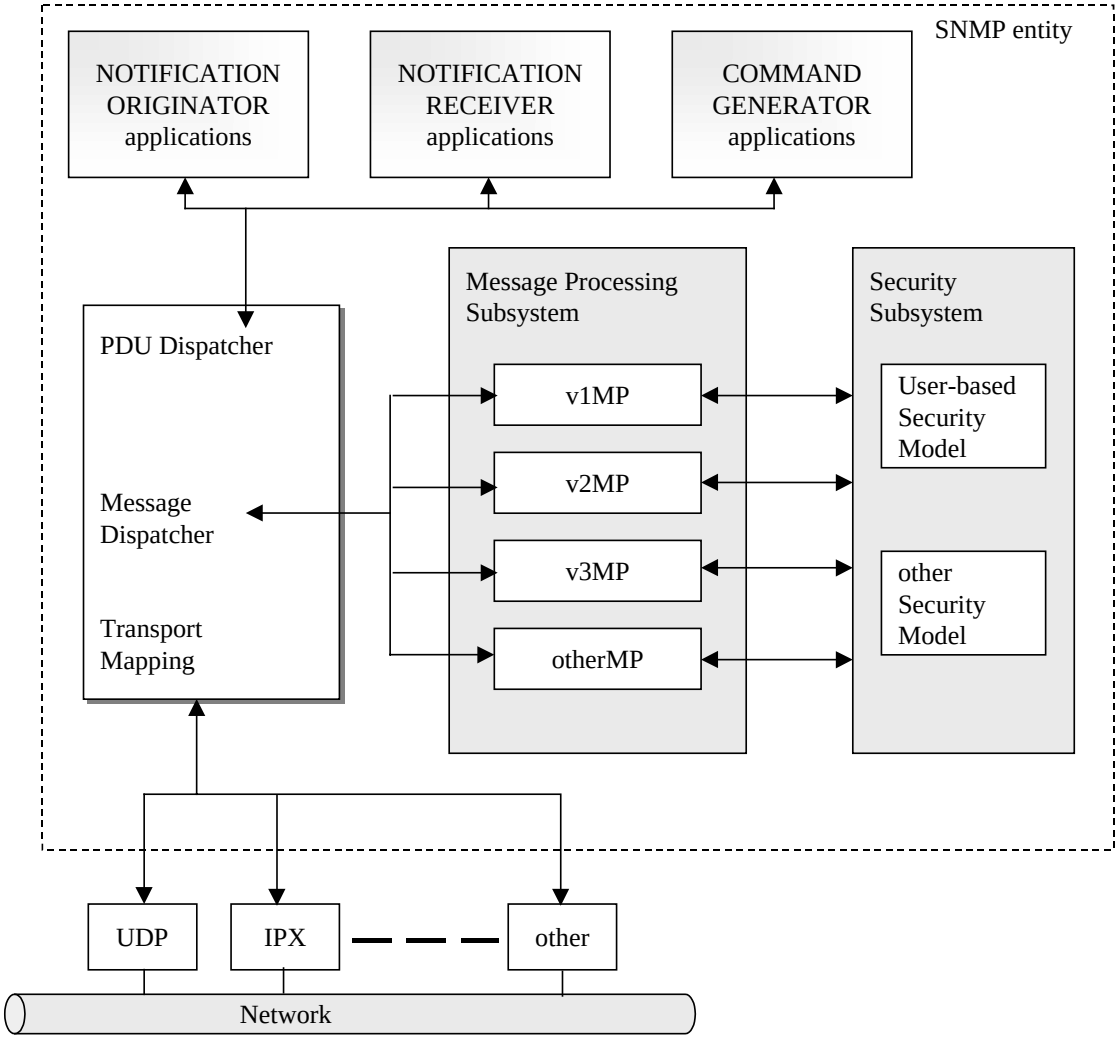


figure 8 SNMP Manager V3

Conclusion

Nous pouvons faire ici plusieurs remarques importantes. La première est que l'importance de la distribution dans le cadre de la gestion de réseau est reconnue et admise au sein même de l'IETF. La deuxième est que SNMP n'est pas encore préparé à supporter cette distribution notamment en raison de la structure du système d'information utilisée. Toujours à propos du système d'information, si nous examinons de près les RFC produites par l'IETF nous pouvons remarquer que les quelques 3000 MIBs propriétaires qui sont utilisées, représentent une entrave indéniable à l'intéropérabilité des systèmes de gestion. Et dernière remarque, si l'IETF était jusque là resté en dehors du monde objet, l'utilisation de propriétés comme l'encapsulation, la réutilisabilité des composants dans l'architecture de SNMPV3 (voir figure 8), ou la prise en compte de technologies comme CORBA marquent une évolution vers celui-ci.
Nous pouvons nous demander si cette nouvelle version de l'architecture de gestion de l'IETF est vraiment mature. Les versions V2 de SNMP n'ont pas réussi à s'imposer et de nouveaux concepts et des applications comme les serveurs web embarqués sur les équipements sont apparus pour compléter les outils de gestion. Par rapport à ces évolutions, et en considérant les délais qui existent entre la reconnaissance des standards et leur présence réelle et utilisable dans les équipements et les systèmes de gestion, il est probable que ce nouveau standard soit sorti tardivement.

II.2.4 Réalité des plates-formes de gestion

Comme nous venons de le voir, de nombreux efforts ont été faits pour étudier et standardiser les systèmes de gestion. Développées au départ pour les très grosses entreprises, les architectures des systèmes de gestion de réseaux sont restées complexes malgré les nombreuses recherches pour simplifier ou faciliter leur conception (OSF/DME, ISO/ODMA, ITU/TMN), et malgré les avancées technologiques (ISO/ODP). Nous avons vu aussi que cette complexité se traduisait par le coût élevé du développement de ces plates-formes mais aussi par le coût d'exploitation de celles-ci. La question que nous devons impérativement nous poser maintenant est si cette complexité n'est pas évitable. Plus précisément, il faut savoir si les standards qui ont été produits sont suffisants, et dans le cas contraire s'il y a un intérêt à en produire d'autres.
En prenant en compte le fait qu'il n'existe pas deux réseaux identiques, que leur taille varie de quelques machines connectées à quelques millions (internet), et que tous les jours de nouveaux équipements et services sont mis en place, il est raisonnable de se demander s'il sera possible d'avoir un jour un seul modèle de gestion.

Maintenant, si nous voulons savoir si les solutions disponibles sont acceptables, ce n'est pas la communauté des chercheurs qui peut statuer là-dessus mais les utilisateurs. A ce sujet, une enquête récente de Data Communication parue le 21 septembre 1999 [www.data.com/issue/990921] met en relief le mécontentement des utilisateurs de plates-formes, qui dénoncent:

- la complexité des API

- le manque de flexibilité
- les faibles performances
- l'inadéquation entre leurs demandes et les capacités des frameworks
- la nécessité de reconstruire leur serveur à chaque correction d'erreurs (patchs).

Il est à souligner ici que ce type d'étude est rare et que certains fournisseurs de solutions de gestion imposent leurs propres équipes pour la mise en place et les configurations des systèmes, cachant ainsi aux utilisateurs une partie de cette complexité. De même, il est pratiquement impossible de faire une comparaison objective de deux plates-formes, ne serait-ce que parce qu'il n'existe pas deux réseaux identiques, et que les avantages que présentent une plate-forme pour gérer un type de réseau, peuvent être des inconvénients pour un autre type de réseau.

II.2.5 Conclusion sur les approches standardisées

Dans la section précédente nous avons fait remarquer que les standards actuels, bien qu'indispensables à la gestion des équipements et à l'interconnexion des réseaux, n'apportent pas encore la solution idéale attendue par les utilisateurs. Les procédures de normalisation elles-mêmes sont remises en cause, car étant beaucoup trop longues elles amènent les constructeurs à proposer des solutions de gestion propriétaire, lorsqu'il s'agit de gérer de nouveaux types d'équipements [PM99]. Ces solutions posent ensuite d'importants problèmes pour leur intégration dans les plates-formes de gestion.

Un autre problème est lié à la conception manager / agent décrit par les standards et plus particulièrement au rôle que joue l'agent. Toutes les approches, même celle de l'OSI, présupposent que l'agent n'est là que pour renvoyer des informations de gestion au système qui lui, au contraire, est chargé de gérer le réseau et par conséquent présupposé avoir toute la "connaissance de la gestion" [Pic98]. L'utilisation d'une telle conception interdit à l'agent la possibilité de prendre des initiatives pour corriger la cause d'une alarme qu'il a détectée.

Plus généralement, les standards de gestion de réseaux proposent des solutions:

- qui sont très sensibles au facteur d'échelle. La principale raison étant leur approche manager/agents
- qui ne permettent pas de traitement par domaine. Une vue globale d'un domaine de gestion se traduit par des relations individualisées.
- qui n'autorisent pas la construction d'un système d'information hiérarchique. Il est très difficile de créer des niveaux d'abstraction, permettant de faire abstraction des objets gérés contenus
- qui présentent des systèmes d'informations trop complexes (OSI, TMN) ou trop spécifiques aux équipements et constructeurs (MIBs SNMP).

- ▷ en trop grand décalage avec la réalité du marché. Les délais de normalisation sont incompatibles avec la demande du marché, et de plus ces standards ne sont pas toujours suivis (SNMP V2)
- ▷ qui ne sont que partiellement implémentés. Les frameworks n'implémentent qu'une partie des standards et diminuent de ce fait leur interopérabilité, et donc l'intérêt des standards.

On constate aussi que dans l'environnement hétérogène que constitue un réseau, le manque d'abstraction complique singulièrement la tâche du gestionnaire de réseaux. Il existe de ce fait de nombreuses études proposant des méta-modèles, des traducteurs [IIMC] (ISO/CCITT Internet Management Coexistence) [RC97], des proxys ou passerelles [RPS96] ou encore d'autres solutions pour permettre aux développeurs d'applications de gestion de faire autant que possible abstraction des modèles ou protocoles utilisés par les plates-formes ou équipements. Ces outils bien que souvent indispensables, ne sont pas encore la panacée car ils augmentent la complexité du système de gestion et entraînent des pertes d'informations [HAN99].

II.3 Identification des principaux problèmes

Depuis le début de ce deuxième chapitre nous avons pu constater qu'il existe deux niveaux de problèmes liés à la gestion de réseaux. Le premier est le niveau conceptuel auquel sont confrontés les développeurs d'applications de gestion. Le deuxième est le niveau opérationnel auquel sont confrontés quotidiennement les administrateurs réseaux. Si les problèmes relatifs au niveau conceptuel ont été abordés lors de la revue des principaux standards, les problèmes du niveau opérationnel sont plus difficiles à cerner. En effet en dehors des enquêtes menées par les journaux spécialisés comme celle de Data Communications que nous avons déjà citée, il est impossible de trouver des études détaillées sur les problèmes relatifs à l'utilisation des plates-formes de gestion de réseaux.
Il existe plusieurs explications à cette constatation. Du côté des fournisseurs de solutions de gestion, les renseignements de ce type sont d'une importance commerciale stratégique, et sont donc gardés confidentiels. Du côté clients, peu d'opérateurs ou d'entreprises sont prêts à partager les informations sur les problèmes rencontrés et les dépassements de budgets inhérents à l'installation de leur NMS.
Néanmoins les justifications des recherches menées dans le cadre de la gestion de réseaux et les publications qui en découlent nous permettent d'avoir un aperçu de la plupart des problèmes importants identifiés et reconnus. Dans cette section nous proposons une classification des principaux problèmes.

II.3.1 Hétérogénéité

Il y a une vingtaine d'années les fabriquants d'équipements de réseaux étaient peu nombreux, et pour éviter les problèmes d'incompatibilité, il était préférable de construire son réseau en utilisant les équipements d'un même constructeur. Les solutions de gestion qui étaient proposées par ces constructeurs étaient fiables et suffisantes, chaque nouvel

équipement étant testé dans un environnement connu et précis. La contrepartie était une dépendance inacceptable des entreprises envers les constructeurs. Pour éviter cette dépendance, et faire jouer la concurrence, les entreprises se sont équipées de différents réseaux qu'il était très difficile d'interconnecter.
Depuis, grâce à la normalisation des protocoles, de nombreuses sociétés se sont développées en proposant des équipements de réseaux censés être compatibles entre eux. Des passerelles ont été proposées pour faire communiquer les différents protocoles, et partager les services d'un type de réseau à un autre. L'offre s'est beaucoup diversifiée et il est courant de trouver de petits réseaux employant simultanément des protocoles comme Ethernet, ATM, ou Token Ring, pour distribuer des services de partage de fichiers (NFS), de courrier, ou même d'impression.
Cette hétérogénéité des équipements composant un réseau impose une compétence très importante de la part des administrateurs. Ceux-ci doivent par exemple maîtriser les différentes interfaces de gestion qui sont fournies par les constructeurs pour un même type d'équipement. [Mas97] de Cisco Systems Inc, nous explique qu'un des problèmes les plus délicats que doivent traiter les vendeurs d'équipements de réseaux est la réalisation d'un système de gestion fiable et extensible. Une des raisons évoquées est le manque de coordination entre les différents constructeurs. Le résultat est un foisonnement d'interfaces et d'outils de gestion spécifiques à chaque constructeur. La solution qui devrait être l'utilisation des standards est souvent rejetée ou biaisée du fait de la complexité de ceux-ci.
Nous pouvons distinguer quatre niveaux d'hétérogénéité:

- l'hétérogénéité des protocoles
- l'hétérogénéité des équipements (fournisseurs et types)
- l'hétérogénéité des applications
- l'hétérogénéité des systèmes de gestion

Le contrôle des protocoles dépend des interfaces de gestion fournies par les fournisseurs d'équipements qui proposent le plus souvent des applications spécifiques. La modification de la configuration d'un réseau privé virtuel (VPN) sur ATM par exemple peut nécessiter, l'utilisation d'autant d'applications que de types d'équipements ATM composant le réseau. Si des propositions ont été formulées pour uniformiser ces interfaces en utilisant des protocoles de gestion classiques comme SNMP [AC96], l'introduction des technologies internet avec HTTP, ou les applets JAVA semble rencontrer plus de succès auprès des constructeurs. Ceux-ci proposent de plus en plus souvent la possibilité d'interagir avec leurs équipements directement à partir d'un browser internet qui fait alors office de station de contrôle, celle-ci se connectant sur un serveur HTTP "embarqué" sur le système à gérer. Bien qu'intéressante, cette approche ne permet pas l'automatisation des procédures de configuration.
Les trois premiers niveaux d'hétérogénéité (protocoles, équipements, applications) ont bien sûr un impact sur le dernier niveau, celui des systèmes de gestion de réseaux. En effet les applications, équipements et protocoles sont souvent associés à des applications de gestion spécifiques ou partiellement intégrées dans les plates-formes de gestion.
Au niveau des plates-formes de gestion, c'est l'hétérogénéité des différents standards de gestion utilisés qui pose un problème. Car si les petits réseaux n'utilisent généralement

qu'un seul système de gestion, les moyens et gros réseaux utilisent plusieurs systèmes de gestion différents et il se pose alors le problème de l'intéropérabilité de ceux-ci.
Des groupes comme le JIDM (Joint Inter Domain Management) supportés par des organisations internationales comme le TMF (TeleManagement Forum) ou l'OMG (The Open Group, OMG) ont été chargés de définir les moyens d'interconnecter les domaines de gestion en utilisant comme support la technologie Corba (voir figure 9).

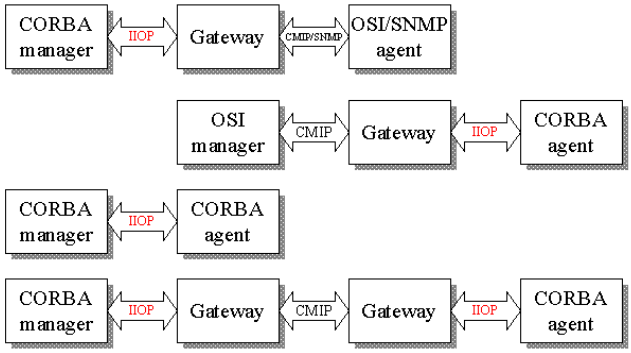


figure 9 Scénarios d'interactions JIDM

D'autres se penchent sur le problème du système d'information universel comme le DMTF (Distributed Management Task Force) qui propose CIM (Common Information Model).

II.3.2 Complexité

Comme nous venons de le voir, la complexité des systèmes de gestion est en partie liée à l'hétérogénéité des réseaux. Cette complexité se répercute au niveau de la conception des systèmes de gestion [Maz97], mais aussi au niveau opérationnel. Néanmoins si nous considérons un réseau à peu près homogène en terme de protocoles et équipé par un même constructeur, nous remarquerons que sa gestion n'est pas pour autant simple. Actuellement il est pratiquement impossible au gestionnaire de systèmes et de réseaux de connaître les répercutions que peuvent avoir les défaillances d'un élément du réseau sur les services de ce même réseau. Ce problème est essentiellement lié à la conception des systèmes d'information des plates-formes de gestion, qui utilisent des données de bas niveau, sans pouvoir offrir facilement des abstractions. Par exemple la disponibilité des ressources, le coût d'utilisation, ou la priorité pour la mise en place d'un service, sont des abstractions qui seront difficilement traduisibles et gérées par les plates-formes de gestion conventionnelles. L'utilisation d'abstractions comme le *domaine*, ou le *rôle* sont l'objet de recherches depuis plusieurs années, notamment sous l'influence des travaux de Sloman [Lup97] [Slo96] [SR96]. Mais celles-ci n'ont pas encore dépassé le stade expérimental.
Un autre paramètre intervenant sur la complexité des plates-formes est lié à la ou aux technologies utilisées pour leur développement. Si l'utilisation des méthodologies et des langages objet a permis de mieux maîtriser une partie de cette complexité, la gestion de l'évolution des architectures reste toujours problématique.

II.3.3 Coût

La gestion de réseau si elle est indispensable, est une source de coût important. Le prix des plates-formes est élevé au départ, puisqu'il faut débourser au minimum plusieurs centaines de milliers de francs pour une plate-forme, et à ce prix de base se rajoutent les services. L'IDC (International Data Corporation) estime que pour 1$ investi dans l'achat d'un framework de gestion de réseaux, 3$ sont dépensés en service après vente. Ces estimations ne prennent pas en compte le coût d'exploitation très important lui aussi, et ces chiffres élevés sont souvent la source de mécontentement des utilisateurs qui considèrent ne pas en avoir pour leur argent. Ce coût est bien entendu en grande partie dû à la complexité des plates-formes de gestion que nous avons évoquée précédemment.
Un autre paramètre ayant une influence sur le coût des plates-formes est l'architecture centralisée des systèmes de gestion. Il y a quelques années cette approche était légitimée par le coût des équipements, la puissance de calcul valait cher et il était préférable de regrouper toutes les données collectées près d'un système unique de contrôle. Ce n'est plus le cas maintenant car le coût des systèmes de communication et des équipements informatiques a suffisamment baissé pour qu'il soit tout à fait envisageable d'avoir un système de contrôle et une interface de communication sur des éléments aussi simples que des ampoules. Si l'on en n'est pas encore là, il faut quand même s'intéresser aux moyens de faire baisser le coût des systèmes de gestion. Sur ce sujet [HAN99] nous fait remarquer que les concepts de systèmes terminaux intelligents sont une approche intéressante pour tenter de réduire les coûts inhérents à la gestion des systèmes "bêtes". En effet la distribution de l'intelligence et donc du traitement des opérations de gestion semble être "la" voie pour réduire la complexité et le coût de la gestion de réseaux comme nous le verrons plus loin.

II.3.4 Délai

Une autre conséquence de la complexité des plates-formes de gestion est le délai de développement et de mise en place d'un système de gestion. D'après une enquête de l'IDC il faut entre plusieurs mois et 2 ans pour rendre opérationnel un NMS.
Une fois opérationnel, le NMS doit pouvoir s'adapter aux modifications de l'architecture du réseau. Ces modifications nécessitent une étude et un travail de mise à jour aussi long. Les délais de mise en place de nouveaux services sont d'autant plus dramatiques pour les entreprises qu'ils réduisent leur possibilité de s'adapter à des marchés à fort rendement en raison de leur extrême dynamicité. Ces délais caractérisent la faiblesse de l'infrastructure des systèmes de gestion et des concepts utilisés, mais sont aussi dus à la lenteur des processus de normalisation [HAN99]. Certains marchés comme la téléphonie sur internet stagnent actuellement en raison de l'impossibilité de prendre en compte à un coût et dans un délai raisonnables la qualité de service attendue.
Les méthodologies et les technologies utilisées pour développer les futures plates-formes de gestion de réseaux, doivent donc permettre la modification dynamique d'un côté de l'architecture, et de l'autre des fonctionnalités.

II.3.5 Flexibilité

La notion de flexibilité désigne ici la facilité d'intégration à la fois de nouveaux types d'équipement, de nouvelles applications, mais aussi la facilité que peuvent avoir les administrateurs de réseaux à modifier ou à réorganiser les activités de gestion. Nous considérons aussi la facilité de développer une nouvelle application, ou à créer un nouveau service, qui peut se mesurer en terme de réactivité à la demande de création. Plus la réalisation est rapide, plus la réactivité est grande.
S'il n'est pas dans notre objectif ici de chercher à améliorer les délais de normalisation, nous pouvons par contre nous pencher sur les problèmes de "réactivité" des applications de gestion. Ce manque de réactivité est attribué par [HAN99] à la faiblesse des concepts couramment utilisés pour le développement des NMS. [BGP97] considère le manque de réactivité pour le développement des applications des gestion comme étant dû à un manque de flexibilité des architectures des systèmes de gestion. Le résultat étant la difficulté de la mise en place et de l'intégration de nouvelles applications. Certains systèmes de gestion dit "ouverts" offrent une certaine flexibilité en autorisant le chargement dynamique d'applications. Mais ce type de flexibilité ne répond pas aux besoins décrits précédemment car il n'y a pas d'intégration réelle au système de gestion. Pour assurer cette véritable flexibilité, les architectures de gestion de réseaux doivent donc autoriser l'insertion de nouvelles fonctionnalités sans remettre en cause l'existant.

II.3.6 Fiabilité

Traditionnellement la gestion de réseaux amène à considérer deux types de fiabilité. La fiabilité du système de gestion et la fiabilité du réseau lui-même. La défaillance temporaire du NMS n'entraîne généralement pas de défaillance du réseau, celui-ci pouvant continuer à fonctionner s'il n'y pas d'opération de changement de configuration majeur en cours. Dans l'autre sens il arrive plus souvent qu'une défaillance du réseau rende le système de gestion inopérationnel, principalement quand la défaillance remet en cause la connexion du système de gestion au reste du réseau. C'est un problème connu des systèmes de gestion centralisés. Pour pallier à ce problème de fiabilité très important dans beaucoup de domaines, comme le domaine bancaire ou la navigation aérienne, la solution utilisée par les systèmes traditionnels consiste à dupliquer voir tripler les centres et les systèmes de gestion, et en cas de défaillance d'un des systèmes à commuter la gestion du réseau sur les systèmes valides. Cette solution ne peut être envisagée que pour des grosses entreprises en raison du coût et de la difficulté technique qu'elle représente. Les utilisateurs de réseaux moins critiques sont quant à eux contraints de supporter les pannes.
La fiabilité du réseau, quant à elle va être garantie par la surveillance des équipements (monitoring) qui permet de détecter et d'identifier des anomalies dans leur fonctionnement. Quand une anomalie ou une panne est identifiée une alarme est transmise au système de gestion, qui doit alors exécuter une procédure de traitement pour rétablir un fonctionnement normal. Afin de réduire le temps entre la détection d'une

anomalie et le rétablissement du fonctionnement normal du réseau, il est fait de plus en plus appel à des technologies issues de l'intelligence artificielle comme les systèmes experts [Ter98] [PNC99]. L'inconvénient de cette approche est que la corrélation des alarmes, et l'identification de leurs causes, se fait au niveau du système centralisé, et qu'une panne importante peut non seulement en cacher d'autres, mais aussi bloquer le système s'il se trouve débordé.

II.3.7 Extensibilité

Il y a d'un côté les problèmes liés à l'extensibilité fonctionnelle, et de l'autre côté les problèmes d'extensibilité de capacité qui nécessitent le changement du système de gestion s'il n'est pas suffisamment dimensionné pour supporter la croissance du réseau à gérer.
Les systèmes de gestion de gros réseaux doivent pouvoir traiter plusieurs centaines de messages à la seconde, et au-delà d'une certaine taille, il est obligatoire de découper le réseau en plusieurs sous-réseaux qui doivent être gérés séparément. Apparu depuis un peu plus de dix ans, le concept de gestion par délégation [YGY91], serait à même de délester le système central d'une partie de ses activités en les reportant directement sur les systèmes à gérer. Ce concept a évolué pour permettre de déléguer dynamiquement des fonctions de gestion [GY95] et rompre les modèles organisationnels et fonctionnels statiques des systèmes de gestion. La gestion distribuée par délégation devrait donc être une solution aux problèmes d'extensibilité des NMS, mais jusqu'à présent aucune implémentation importante ne permet de valider ce concept.

II.3.8 SLA et QoS

S'il y a actuellement une forte demande pour intégrer la gestion de service à la gestion de réseaux, peu d'outils permettent cette intégration. Ce manque est d'autant plus grave qu'en plus des services de base, la demande du marché correspond à la fourniture de services sur mesure (customisation) avec garantie de la qualité de service (QoS). Ce type de service est l'objet de contrats appelés SLAs (service level agreements). Du point de vue gestion, il augmente les besoins en planification, en suivi de clientèle, et en suivi de QoS. De plus les utilisateurs n'étant pas formés à la gestion de réseaux, il est impossible de leur fournir des données directement issues de MIBs. Pour les utilisateurs les informations importantes sont celles représentant le service qu'ils ont demandé, en faisant "abstraction des détails" concernant les équipements utilisés pour rendre ce service. L'objectif est à moyen terme de leur offrir les moyens de configurer, d'activer, de stopper, automatiquement ces services. Actuellement, l'introduction d'un nouveau service de ce type est très coûteux car il nécessite la création de nouveaux objets de gestion difficiles à développer et la modification du flot de traitement des informations de gestion pour les rendre accessibles aux utilisateurs. Sous la pression du marché, des opérateurs mènent des recherches pour développer le SLA [MAZ99], mais le manque de souplesse des systèmes sous-jacents les oblige à n'utiliser que des services "préconfigurés" et donc limités, et à réduire le suivi de la qualité de service à des rapports journaliers ou à la

demande. La déficience des standards pour traiter ces problèmes de contrôle de QoS a déjà été analysée et reportée [HCCB94], mais malgré les efforts de normalisation (SC21 ISO) aucune solution satisfaisante n'a encore été trouvée.

II.3.9 Sécurité

Avec le raccordement quasi obligatoire des réseaux à internet et l'avènement du commerce électronique, la sécurité des réseaux est un élément des plus sensibles de la gestion de réseaux. Classiquement cinq types de sécurité sont définis (ISO 7498-2):

- la *confidentialité*: assure que les données ne seront lisibles que par le destinataire
- l'*authentification*: assure le destinataire de l'identité de l'expéditeur, et vice versa
- l'*intégrité*: assure que les données n'ont pas été modifiées
- la *non-répudiation*: assure qu'un message a bien été envoyé et reçu par les bonnes personnes
- le *contrôle d'accès*: garantit que les personnes n'accèdent qu'aux ressources qui leur sont autorisées.

La plupart des services de gestion de la sécurité font appel à des mécanismes de cryptage [Puj97]. Mais le cryptage ne protège pas le système de gestion lui-même des attaques possibles et [Roz99] souligne par exemple que les interfaces définies par le TMN sont particulièrement sensibles aux agressions. De plus en plus "perfectionnées" les attaques et intrusions contre les réseaux nécessitent la corrélation d'événements anormaux provenant de tous les équipements. Des études récentes ont permis de démontrer qu'un système distribué coopératif offrait une bonne protection contre ce type d'attaques [].

II.3.10 Conclusion – Orientation des NMS

Les études entreprises afin de résoudre les problèmes que nous venons de classer nous permettent de dégager les tendances et les orientations qui caractériseront les systèmes de gestion de demain. De celles-ci ressortent principalement l'importance de la distribution aussi bien de l'information, de l'intelligence ou des systèmes, et l'apparition et l'utilisation de nouveaux niveaux d'abstraction comme les services, rôles ou politiques.

La distribution

Le réseau représente à lui seul un système distribué. L'information se trouve répartie sur ce système et il apparaît de plus en plus évident qu'il faille la traiter le plus près possible de son lieu de production. Pour cela l'intelligence des systèmes de gestion doit se distribuer elle aussi. Cette *distribution de l'intelligence* du traitement de l'information a plusieurs corollaires. Le premier est la nécessité de laisser une certaine *autonomie* aux entités de gestion, car contrôler systématiquement toutes les opérations du centre de gestion principal serait en contradiction avec les intérêts de la distribution. L'avantage

immédiat serait une plus grande fiabilité en cas de panne, chaque sous-entité de gestion pouvant continuer à opérer. Cette autonomie entraîne aussi une plus grande flexibilité dans l'organisation du système de gestion.
Le deuxième corollaire est l'utilisation de langages de gestion *orientés objectifs* et donc ayant un plus *haut niveau d'abstraction* pour éviter de tomber dans les pièges des particularités de chaque sous-système. On entend par cela de communiquer le quoi par rapport au comment. Grâce à cela les problèmes d'hétérogénéité sont fortement diminués voir supprimés. La complexité du développement diminue aussi car il n'est plus obligatoire de prévoir comment gérer tous les systèmes, mais avant tout de savoir ce que l'on veut gérer. Les délais et les coûts de conception et d'intégration sont de ce fait eux aussi diminués.
Le troisième corollaire est la *coopération* entre les entités de gestion pour accomplir des tâches inter-domaines. Un exemple courant concerne la mise en place de VPN entre différents sous-réseaux. Les entités de gestion pouvant travailler ensemble et partager les ressources en fonction des intérêts dictés par les politiques de l'entreprise, la gestion de SLA, de QoS devient plus facile. Un dernier corollaire est la *délégation*, qu'elle soit de tâches ou d'objectifs.

L'abstraction

L'utilisation d'abstractions pour la conception de logiciels est une idée "vieille" de plus de vingt ans introduite avec la programmation objet. L'introduction de l'approche objet a permis d'améliorer le développement d'applications importantes en taille et en complexité comme les NMS. Tout d'abord en simplifiant la conception grâce à la méthodologie OMT, puis UML , ensuite en simplifiant le codage grâce aux langages de programmation comme C++.
Malgré tout, ces applications manquent de flexibilité, une fois le design terminé, il est très difficile d'en modifier les fonctionnalités. Comme nous venons de le voir les NMS doivent s'adapter en permanence à l'évolution des technologies et de la topologie des réseaux, et ce le plus souvent lors de la phase opérationnelle. Il est donc nécessaire d'offrir aux administrateurs la possibilité de modifier simplement et rapidement les vues de leurs réseaux, en introduisant de nouvelles abstractions même si elles ne sont pas prévues lors de la conception du système de gestion.
C'est pour cela que les spécialistes du développement objet se sont penchés vers d'autres abstractions plus facilement utilisables pour concevoir un système distribué. Le modèle de référence ODP (Open Distributed Processing) issu d'un travail commun entre l'ISO et ITU-T, définit cinq niveaux d'abstraction appelés vues:

- entreprise: l'objectif, la portée et les règles du système,
- information: la sémantique et la manipulation de l'information,
- traitement: la modélisation de la distribution par décomposition fonctionnelle du système des interfaces et de leurs interactions,
- ingénierie: les mécanismes et l'infrastructure, nécessaires au support de la distribution,
- technologie: le choix technologique pour l'implémentation du système

Dans une étude sur les paradigmes utilisables pour développer les systèmes de gestion distribués, [MZH99] classe trois niveaux d'abstraction:

- l'objet géré (faible niveau)
- l'objet traité (haut niveau d'abstraction)
- l'objectif (très haut niveau d'abstraction)

Si les deux premiers niveaux d'abstraction sont ceux utilisés actuellement pour développer les NMS, la notion d'objectif, de rôle, ou de politique sont par contre de nouvelles abstractions qui marquent les tendances de la recherche.
Au plus haut niveau se place la politique, qui doit permettre de définir les règles de fonctionnement du système de gestion indépendamment des services fournis. Puis les rôles qui définissent les capacités fonctionnelles des entités de gestion, et enfin les objectifs qui permettent de déterminer les opérations à effectuer.
Ces niveaux d'abstraction, lorsqu'ils sont utilisés par un système de gestion composé d'entités autonomes et coopératives, permettent d'offrir une très grande flexibilité aussi bien pendant la phase conceptuelle que pendant la phase opérationnelle. Bien entendu les systèmes de gestion doivent pouvoir interpréter ces abstractions, et les concepteurs et administrateurs doivent avoir à leur service les outils permettant leur création et leur manipulation.

II.4 Evolution du paradigme objet

Nous avons constaté dans la section précédente que la distribution des applications de gestion était une nécessité, cette section passe en revue l'évolution des paradigmes qui contribuent au support de cette distribution.
Apparu il y a un peu plus de vingt ans, le paradigme objet a constamment évolué pour passer d'un code statique à un code mobile. Cette section présente cette évolution vue au travers de la distribution du code d'une application et des contraintes de couplage entre les composants du code de l'application.

II.4.1 Distribution et contrôle du code

La conception objet utilise les notions de méthodes et de propriétés, la modification d'une propriété d'un objet se faisant par un appel de la méthode correspondante. Cette conception sous-entend que la procédure ou méthode appelante possède les références à la fois de l'objet et de la méthode à appeler. Nous pouvons dire que dans une application classique les objets sont fortement couplés, et que ce couplage se fait à la compilation de l'application, opération pendant laquelle sont résolus les problèmes d'adressage.
Dans le cas d'une application distribuée, c'est-à-dire s'exécutant simultanément sur au moins deux systèmes différents, il est impossible de connaître au moment de la compilation les adresses même relatives des objets et des méthodes. Les objets, ou plutôt

les classes à ce moment, ne sont pas couplés lors de la compilation, mais uniquement à l'exécution (runtime) grâce à des opérations comme les "binds". Les objets d'une application distribuée sont regroupés sous forme de modules ou composants, chaque composant pouvant être exécuté sur un système différent.
Les sections suivantes présentent les paradigmes qui ont contribués à augmenter la distributivité des applications.

II.4.1.1 Appel de procédures à distance

L'appel de procédure à distance ou RPC en anglais (Remote Procédure Call) est le plus ancien des paradigmes utilisés pour la création d'applications distribuées [Ste90] sur système Unix.
Le principe est ici de proposer au concepteur une transparence de la localisation du code (procédure) à exécuter, et le mécanisme d'appel identique à celui utilisé au sein d'une application monolithique, la localisation des objets étant déterminée à l'exécution de l'application [BN84].
Les paramètres nécessaires à l'exécution du code distant sont encodés (marshaling) sur le système appelant, puis décodé sur la machine (unmarshaling) cible. De même que pour une procédure locale la procédure appelante est suspendue en attendant la fin de l'exécution de la procédure distance appelée (synchronisation de l'exécution).
Le RPC a été développé à l'origine pour des systèmes unix et sans considération de la programmation objet. Il a été récupéré par l'OSF pour son architecture distribuée DCE ouverte à tous les systèmes, et par l'OMG (Object management Group) avec CORBA qui l'a intégré dans une approche objet. Ce concept est aussi utilisé par Java/RMI.
C'est une *délégation statique d'exécution* dans le sens qu'il n'est possible de déléguer l'exécution du code que lorsque celui-ci est déjà chargé sur le système de gestion de l'équipement.

II.4.1.2 Evaluation à distance

Le problème de l'approche client serveur traditionnelle est que si plusieurs opérations doivent être faites successivement et utilisant les même codes et données distantes, nous auront alors une augmentation du nombre de communication et une diminution des performances de l'application. De même on peut comprendre qu'il est intéressant de prévoir la possibilité de faire exécuter du code sur des machines en fonction de leur charge. C'est dans ces objectifs qu'a été introduit le concept d'évaluation à distance ou REV (Remote EValuation) en anglais. L'application est alors composée de sous-programmes indépendants que l'on peut envoyer s'exécuter sur des noeuds différents [Dal97] [SG90].
Il s'agit d'une *délégation dynamique d'exécution*, puisque le code à exécuter peut être transmis sur le site distant quand c'est nécessaire.

II.4.1.3 Serveur élastique

Maintenant si nous nous plaçons du côté du "serveur" c'est-à-dire du système où les ressources vont être utilisées, on doit s'occuper de la gestion des différents codes qui sont reçus, c'est le concept du serveur élastique ou ES (Elastic Server) en anglais.
Le client envoie le code sur l'équipement et peut faire autre chose pendant ce temps, c'est le système de l'équipement qui contrôle l'évaluation. Le contrôle de l'exécution est à la charge du système distant, mais le manager peut communiquer avec celui-ci pour savoir quel est le statut de l'exécution, et décider de suspendre ou stopper celle-ci grâce à un protocole de contrôle (Remote Delegation Protocol). Il s'agit ici d'une *délégation directive de tâche*. Ce concept de serveur élastique a été introduit par [YG91] et a été revu dans [GY95] pour décrire une architecture complète représentant une combinaison du REV et du CS (Client Serveur). Tout comme avec le REV le manager peut avec l'ES préparer une série de commandes avec le code à exécuter et envoyer le tout sur un système distant, mais l'exécution est alors asynchrone par rapport au REV qui est synchrone. Une autre différence est que le serveur élastique permet le contrôle de l'exécution du code distant contrairement au REV.

II.4.2 Code à la demande

Toujours en se plaçant du côté serveur par rapport aux approches que nous venons de citer une autre approche permet d'optimiser la gestion dynamique des ressources, c'est le code à la demande. Avec ce concept, le serveur peut rechercher les dernières versions d'un code à exécuter et de le lier dynamiquement avec d'autres codes objets pour étendre une application.

Il demande aux systèmes distants d'exécuter la tâche de collecte, et laisse le système distant le soin de récupérer le code correspondant à la tâche.
Le code à la demande n'est pas une notion de délégation mais participe au mécanisme de délégation dans le sens ou il permet à un système de récupérer du code pour exécuter une tâche qui lui a été confié. Le COD intervient donc ici dans la *délégation de tâche dynamique*. Il peut être associé à la notion de serveur élastique pour rendre celui-ci dynamique.

II.4.3 Code mobile.

Les mécanismes de sérialisation/désérialisation offerts par les langages du type JAVA/RMI ou des architectures de type CORBA, permettent de créer facilement des applications comportant des objets mobiles, plus communément appelés codes mobiles.
Le code mobile s'intéresse au déplacement et l'exécution du code (composant) d'un système à un autre sous le contrôle d'une application. Dernièrement le concept de code mobile a été le point de départ de travaux sur les réseaux actifs où chaque message ou "capsule" échangé entre les noeuds d'un réseau correspond à un fragment de programme.

II.4.4 Agent mobile

En étendant le concept de code mobile à l'application elle-même, on arrive à l'agent mobile, qui représente une application complète, capable de se déplacer ou d'être déplacé d'un système à un autre.
L'agent mobile est différent du paradigme de code mobile dans le sens où les interactions qui lui sont associées impliquent la mobilité de son état d'exécution [PIC98] [EB99].
Par rapport aux notions précédentes l'agent mobile représente une unité complète d'exécution (programme), et contient donc son propre contrôleur d'exécution. Entre l'appel de procédure à distance, et l'agent mobile, en passant par l'évaluation à distance et le code mobile il y a une progression constante de l'autonomie d'exécution. Le contrôle d'exécution passe en effet du manager à l'agent de gestion statique (ex: SNMP) puis à une entité logicielle appelé agent mobile, qui se déplace de son propre chef (mobilité forte) ou sous le contrôle du manager (mobilité faible). Cette notion d'*autonomie* a des implications importantes dans la conception de logiciels comme nous le verrons plus loin. Il s'agit de *délégation dynamique et mobile de tâches*. Dernièrement de très nombreuses études ont porté sur les agents mobiles, afin de prouver qu'ils étaient "la" solution aux problèmes d'échelles rencontrés par les systèmes de gestion [BP98]. Nous attirons aussi le lecteur sur le fait que la confusion a été souvent faite entre agents du système de gestion, agents mobiles et agents intelligents, concept que nous introduisons plus loin dans cette thèse.

II.4.5 Conclusion

Les codes mobiles et les agents mobiles sont les dernières propositions d'évolution du monde objet pour rendre indépendant et autonome chaque objet. Mais aucun des deux paradigmes ne traite le problème de la gestion des ressources de l'environnement d'exécution sur lequel sont amenés le code ou l'agent. Seul l'aspect sécurité est méticuleusement étudié, le code mobile ressemblant fortement à quelque chose connu sous le nom de virus ou ver (worm). C'est malgré tout un pas de plus vers la distribution du traitement des informations avec la possibilité de rapprocher ce traitement du lieu de production ou de stockage de l'information.
Malgré l'apparition des langages objets la complexité de la conception des applications est toujours d'actualité. Depuis maintenant deux ans l'UML (Unified Modeling Langage) semble reconnu comme la méthodologie à utiliser pour le développement objets, alors qu'il en existait une cinquantaine en 1994 [BRG99]. Il reste à vérifier maintenant que cette méthodologie est applicable au développement d'applications basé sur des objets mobiles, ce qui est loin d'être fait.
D'autres problèmes ne sont pas non plus résolus par cette approche. La communication entre les codes mobiles (ou agents mobiles) n'est pas prise en compte si ce n'est au travers d'un système centralisé, réduisant d'autant l'extensibilité des applications. La coopération entre entités autonomes, bien que nécessaire, n'est donc pas encore possible.
Malgré tout les concepts, d'agent, d'autonomie, de délégation de tâches, représentent une véritable révolution dans le monde objet.

II.5 Evolution des concepts et technologies de NM

II.5.1 Gestion par délégation

Dans les sections précédentes, nous avons souligné l'importance de la notion de délégation dans le développement des technologies de programmation issues du monde objet. L'intérêt de la délégation a aussi été remarquée lorsque nous avons examiné les différents problèmes soulevés par la gestion de réseaux. Dans cette section nous nous intéressons plus particulièrement à l'utilisation de ce concept dans la gestion de réseaux.
A l'origine de la gestion par délégation proposé par [YG91] nous trouvons le principe du serveur élastique qui est un serveur amélioré pouvant étendre ou diminuer ses capacités fonctionnelles en cours d'exécution. Un protocole de délégation permet à un client de passer une nouvelle fonction au serveur et de lui demander de l'exécuter. Ce protocole permet aussi de contrôler l'exécution du code correspondant à la fonction, c'est-à-dire d'interrompre, de redémarrer ou de stopper l'exécution.
Afin de réduire les problèmes d'intégration dans un NMS, le serveur élastique peut-être placé entre le manager et l'agent et jouer le rôle de proxy [Sta96]. Ces serveurs élastiques sont appelés agents MbD (Management by Delegation) [YG91], agents de gestion [Tro97], agents flexibles de délégation [Mou96], ou aussi agents élastiques [GY95]
Une alternative est de faire évoluer les agents classiques CMIP et SNMP en serveurs élastiques. Bien que très intéressante, cette proposition concrétisée par un mémo [draft-ietf-AgentX-rfc-update-03] de l'IETF ne connaît qu'un succès mitigé auprès des constructeurs qui rechignent encore à ouvrir leurs systèmes. Dans ce contexte un manager peut utiliser le protocole de délégation pour charger un script de gestion sur l'agent SNMP flexible (ou extensible) et lui demander la permission de l'exécuter. Une certaine autonomie est donnée ici à l'agent pour exécuter les scripts, qui sont appelés "agents délégués" par Goldszmidt [Gol93].
Une autre approche est proposée par [SKN97]. L'idée est que les opérations de gestion que doit effectuer l'agent se portent essentiellement sur les objets de gestion connus de l'agent. Dans cette optique il n'est pas nécessaire que le manager transfère du code à exécuter mais simplement et donc plus économiquement un squelette de script que l'agent doit relier aux objets concernés, et aux primitives opérationnelles qu'il possède dans une librairie. L'objection que l'on pourrait faire ici est qu'il n'est donc pas possible d'étendre le système d'information de l'agent, ni ses primitives.
Ces études sur la gestion par délégation ont pour l'instant tout au moins permis de confirmer l'importance de la délégation pour résoudre les problèmes de flexibilité et d'extensibilité des NMS.

II.5.2 CORBA et le monde OSI

L'OMG a été formé dans l'objectif de réduire la complexité, diminuer les coûts, et accélérer le développement d'applications. La vision principale des quelques 700 entreprises membres de cette organisation est que les applications doivent pouvoir profiter de la puissance de calcul constituée par l'ensemble des ordinateurs connectés à un

réseau. Basé sur l'approche objet, l'architecture OMA (Object Management Architecture) défini autour de CORBA (Common Object Request Broker Architecture), spécifie l'infrastructure indispensable à la conception d'applications distribuées. Le modèle de référence identifie et caractérise les composants, interfaces et protocoles composants l'architecture OMA. Il inclut l'ORB (Object Request Broker) en charge de gérer la communication entre les objets, et quatre catégories d'interfaces:

1. les services objet: ce sont les services de bases indispensables à la manipulation des objets (nomage, événement, persistance, ...)
2. les services communs: ce sont des interfaces utilisées par la plupart des applications (échange de documents,...)
3. les interfaces de domaines: ce sont les interfaces communes à un domaine d'application (finance, télécommunication, ...)
4. les interfaces d'applications: ce sont les interfaces spécifiques à une application

Très tôt après l'apparition de CORBA, de nombreuses propositions d'utilisation de cette technologie ont été avancées et concrétisées pour permettre la construction d'applications distribuées conformes aux spécifications de l'OSI SMA (Systems Management Architecture). Les avantages, en termes de flexibilité d'implémentation, de transparence de la localisation des objets et donc des applications, d'intégration des concepts objet [Pav99] ont pesé suffisamment pour que CORBA soit préconisé comme la technologie à utiliser en remplacement de CMIP pour certaines interfaces du TMN [Sid99].
Il est intéressant de noter que la version 3 de CORBA qui a été adopté fin 1999 permet de gérer des composants, et propose un langage programmation interprété (CORBA scripting language). Les composants représentent au même titre que dans le TMN une entité fonctionnelle composée d'objets.

II.5.3 De SNMP à SNMPV3

Avec la version SNMP V3.0, les problèmes de sécurité qui étaient l'un des points les plus critiqués de la première version de ce protocole, sont globalement résolus. L'architecture prévoit la sécurisation des échanges de messages (intégrité, autentification, encryptage), le contrôle des accès, et autorise aussi la coexistence de plusieurs modèles. Ce dernier point garantit les mises à jour ultérieures du protocoles.
Si SNMP V3 autorise le chargement dynamique d'applications de gestion, [JVS99] argumente qu'un des problèmes soulevé par le développement des agents SNMP est la forte imbrication du code de l'agent et de sa MIB. Le problème apparaît lorsqu'a un moment donné le gestionnaire de réseau veut enrichir la MIB d'un agent avec nouvel objet. S'il a la possibilité de lancer une nouvelle application qui lui permettra de monitorer ce nouvel objet il lui sera par contre impossible de l'intégrer dans la MIB de l'agent, et donc de le manipuler dans une opération sur la MIB. Dans cet étude il est donc proposé l'instrumentation dynamique de la MIB grâce à du code Java, la représentation de la MIB étendue elle-même se faisant en XML.

Ces travaux reflètent le besoin d'étendre dynamiquement le système d'information directement sur le système administré, sans avoir a modifier le système de gestion en place. On peut légitimement en déduire que le problème de la décentralisation du traitement et donc de l'intelligence des NMS n'est pas encore complètement résolu par cette toute nouvelle version de SNMP. La MIB Script [SQ99] représente néanmoins un premier pas encourageant vers la gestion par délégation.

II.5.4 De RMON à RMON2

Les sondes RMON sont les premières formes de délégation utilisées en conformité avec le standard SNMP. RMON permet de surveiller le fonctionnement du réseau en collectant les données qui y transitent et cela sans impact sur les performances de ce dernier. Les objectifs du groupe de travail ayant produits les spécifications RMON (RFC 1757 et RFC 2021) sont:

- l'autonomie de fonctionnement. La sonde RMON doit pouvoir continuer à récolter les informations, calculer les statistiques, surveiller les seuils, sans être connectée en permanence avec la station de gestion.
- le monitoring proactif. Lorsque certains seuils sont atteints la sonde peut envoyer une alarme à la station de gestion pour l'informer d'un risque de panne et monitorer et conserver ces informations pour pouvoir diagnostiquer les raisons de la défaillance.
- la détection de problèmes. La sonde peut-être configurée pour surveiller certaines conditions de défaillance.
- la fourniture de données à valeur ajoutée. En étant directement sur le réseau la sonde peut surveiller l'utilisation de celui-ci et donner des informations comme le classement des systèmes en fonction de leur consommation de bande passante.
- la gestion multiple. La sonde peut être configurée par plusieurs stations de gestion dont les objectifs peuvent être différents, et renseigner par exemple sur la consommation des ressources, tout aussi bien que sur les causes d'une panne.

RMON participe à la distribution de la gestion dans le sens qu'il traite un certain nombre d'opérations de filtrage et de collecte d'informations sur l'utilisation des segments de réseaux. Il est possible grâce à cela de récupérer en une seule requête SNMP des historiques sur des trafics, des statistiques, ou bien de programmer des alarmes, déchargeant ainsi le travail du système de gestion et diminuant du même coup le trafic SNMP.
La version 2 de RMON introduit la notion de protocole de haut niveau (application) et permet par cela de monitorer à distance les activités des utilisateurs. A partir de ces informations, il est possible de contrôler les besoins en terme de ressources, de détecter des comportements anormaux comme des tentatives d'intrusions [GT99].
Les avantages de la délégation de certaines opérations de gestion sont mis en valeur par RMON. Bien que limités à des opérations prédéfinis, cela décharge d'autant les NMS, et

le réseau n'est pas encombré par le trafic qui aurait été nécessaire si les informations n'étaient pas traitées localement par l'agent RMON.

II.5.5 Des éléments passifs aux réseaux actifs

Il devient de plus en plus rare de trouver dans un réseau des éléments passifs. Les concentrateurs non gérables sont, au fur et à mesure, remplacés par des commutateurs configurables. La notion de réseau actif étend le concept d'élément actif pour satisfaire les besoins de répartition du travail de gestion dans le réseau. Un autre élément motivant la recherche sur les réseaux actifs est, pour les constructeurs, l'indépendance que donne ce concept vis à vis des standards de gestion. En pouvant programmer les équipements intermédiaires (ex: routeurs, commutateurs) aussi facilement que les systèmes (ex: PCs, stations de travail), ils peuvent introduire plus rapidement des produits innovants et évolutifs [YS96].
Il existe deux types d'approches pour la réalisation de réseaux actifs. La première consiste à conserver les formats de paquets ou de cellules utilisés sur les réseaux traditionnels, l'utilisateur devant alors envoyer à l'élément visé le programme à exécuter, c'est l'approche discrète. La deuxième approche, quelquefois appelée intégrée consiste à remplacer les paquets par des *capsules* qui contiennent des fragments de programmes que chaque élément actif du réseau va filtrer et envoyer dans une entité d'exécution qu'il possède, s'il est concerné [TSSWM97].
Dans un article sur la supervision et le contrôle des réseaux actifs [ACFF99] les auteurs mettent en avant le challenge que représente une telle technologie. En effet les réseaux actifs servent avant tout à déployer de nouveaux services, qu'ils soient de gestion pure, comme pour la configuration ou la gestion de fautes, ou orientés utilisateur, en offrant des capacités de contrôle sur la qualité de service. Démarrés depuis peu, ces projets soulèvent de nombreuses questions, et des recherches sont en cours pour définir un modèle d'information générique applicable à tous réseaux actifs afin de contrôler les ressources gérables.
Ces études qui sont suivies de près par des constructeurs, nous permettent de confirmer l'intérêt que revêt la distribution des opérations de gestion que nous avons souligné plus tôt.

II.5.6 Technologies Internet

Si l'on demande à l'homme de la rue ce qu'évoque pour lui le terme "réseau informatique", il y a de fortes chances pour que sa réponse soit Internet. La croissance ou plutôt l'explosion d'Internet ces dernières années, et le renouveau économique qui lui est associé n'est pas un hasard. Si depuis plus longtemps, on sait que le traitement de l'information est un élément clef de l'économie, le transport de cet information est tout aussi primordial. Mais les clefs du succès d'Internet, en dehors de la baisse des coûts d'accès qui ont longtemps représenté un frein indéniable, tout au moins en France, sont les technologies qui s'y sont développées et déployées.

En premier lieu les langages *hyper-texte*, comme HTML supporté par le protocole HTTP, qui ont permis la transmission facile et quasi standard d'informations structurées multi-média. Ensuite les *browsers* (brouteurs en français) permettant l'affichage de ces informations (décodage des fichiers HTML), et les moteurs de recherche ont bien sûr fortement contribué au succès d'Internet.
Bien qu'en marge d'Internet le langage JAVA, et sa machine virtuelle qui a permis le développement d'applications indépendantes des systèmes, a aussi été popularisé grâce aux *applets* (application graphiques java) que l'on peut exécuter à partir d'un browser.

L'apport de ces technologies sur la gestion de réseaux se situe à plusieurs niveaux. Le premier apport se situe au niveau des interfaces des systèmes de gestion. Depuis peu la plupart des constructeurs proposent des interfaces dites Web, c'est-à-dire utilisables à partir d'un browser pour piloter leurs équipements. De ce fait, les utilisateurs d'un réseau n'ont plus besoin d'utiliser le protocole SNMP pour savoir par exemple ou en est l'impression de leur document, ils peuvent afficher toutes les informations utiles à partir de leur browser en se connectant sur l'URL de celle-ci (c'est-à-dire dans ce cas son nom ou adresse). De la même manière les administrateurs de réseaux peuvent se connecter sans passer par un système de gestion complexe et modifier les configurations en utilisant un mot de passe. Nous voyons donc qu'en proposant de telles interfaces les constructeurs augmente la *flexibilité* et diminue la *complexité* des interfaces de gestion, mais au détriment de l'intégration si c'est la seule interface qui est fournie.

Le deuxième apport se situe au niveau de l'indépendance des systèmes. Bien qu'encore plus du domaine de la recherche que du domaine industriel, le langage java permet le développement facile d'agent de gestion (ex: SNMP) indépendamment du système utilisé pour peu qu'une machine virtuel java soit disponible sur l'équipement. Une approche de ce type permettrait une mise en place plus facile de mécanisme de push comme le préconise [Mar99] pour diminuer la charge du réseau. En attendant l'utilisation de machines virtuelles par les constructeurs et le développement de la JMAPI (Java Management API) pour le développement d'applications de gestion, des industriels comme Atos proposent l'utilisation d'agents Java servant de passerelles entre les équipements et les browsers internet.

Enfin un dernier apport se situe au niveau du model d'information, grâce aux efforts du DMTF (Distributed Management Task Force) avec leur modèle CIM (Common Information Model) qui se veut aussi indépendant que possible des modèles de gestion existant. Des recherches entreprises notamment par [FFYA99] ont permis de démontrer qu'il était assez facile d'intégrer cette approche dans un environnement de gestion OSI [AFB99]. Cette facilité étant grandement due à un formalisme de modélisation uniforme [RC97].

II.5.7 Evolution de la conception

Le développement de systèmes et d'applications de gestions de réseaux utilise deux approches complémentaires. La première correspond à une conception ad hoc de

solutions pour résoudre un problème précis. La deuxième correspond à une conception lourde basée sur les standards. Cette dernière approche a été suivie et largement diffusée quand les standards étaient eux-mêmes suffisamment simples (ex: SNMP) pour être développés rapidement et utilisés facilement. Dans le cas contraire (ex: OSI, TMN) la diffusion est restée très confidentielle. Le développement ad hoc quand à lui a continué à progresser et l'on trouve sur le marché de multiples outils de gestion simples pour des utilisations spécifiques. Les scripts développés par les administrateurs en fonction des besoins correspondent eux aussi à cette approche ad hoc.

Les propositions pour de nouvelles architectures de gestion sont maintenant basées sur une approche modulaire, généralement par composants [Der97] que l'on peut charger à la demande, ou par agents mobiles [Pic98]. Ces évolutions vont de paire avec l'évolution des technologies comme par exemple CORBA V3.0 qui propose un support pour la gestion de composants, ou comme JAVA avec les fameux "beans". Au niveau de la conception, des standards comme ODP (Open Distributed Processing) ont permis de considérer des niveaux abstractions plus élevés que l'objet. Il est donc tout à fait possible et logique de passer du **micro-management** au **macro-management**, et même au delà comme nous le verrons plus loin. Le micro-management se réfère aux opérations atomiques sur les éléments d'une MIB que doit exécuter par exemple un manager SNMP pour gérer son réseau, et qui génère de ce fait beaucoup de trafic [BGP97][Yem93]. Par opposition le macro-management correspond à la délégation que fait un manager à un agent de ces mêmes opérations.
Un changement de "granularité" équivalent est décrit par [MZH98] sous la dénomination de micro-tâches et de macro-tâches. Dans ce même article les auteurs notent que les concepteurs de systèmes peuvent maintenant utiliser un model d'information plus riche et plus puissant que ne leur permettait la technologie objet d'il y a vingt ans.
La technologie issue des recherches sur les paradigmes de conception d'applications distribuées, permettent de produire des systèmes beaucoup plus flexibles. Une des raisons est par exemple que le système d'information n'a pas besoin d'être entièrement défini a priori, car il peut être augmenté et complété pendant la phase opérationnelle [CCOL98], [GY95].
Globalement nous pouvons constater que la conception objet, laisse peu à peu la place à la conception de composants fonctionnels (CORBA V3.0), et que les recherches se concentrent maintenant sur les étapes suivantes qui sont, par exemple, la conception par objectif [MZH98].

II.6 Conclusion: vers une nouvelle technologie

Nous avons constaté dans ce chapitre l'importance qu'ont les concepts de distribution, d'indépendance du code, de délégation, dans la résolution des problèmes de gestion de réseaux identifiés. Nous avons vu les différents mécanismes de distribution utilisables et pu remarquer que la gestion par délégation était efficace en raison de l'autonomie qui était accordé aux agents des systèmes de gestion pour accomplir une tâche déterminée.
Les langages et la technologies de programmation ont aussi progressé dans ce sens. L'évolution du monde informatique vers le monde du réseau entraîne une mutation des

concepts objets vers des concepts de composants indépendants. Les protagonistes de cette mutation ont constaté la nécessité de prévoir une certaine autonomie attachée à ces composants, autonomie leur permettant de migrer ces composants d'un système à un autre ou de réaliser du plug-and-play.

Les mécanismes mis en jeu sont une première étape vers la délégation de tâches dans le sens ou l'agent du système de gestion va pouvoir exécuter sans un contrôle stricte du manager, un ensemble d'opérations correspondant à une activité de gestion plus complexe que ne le permettrait par exemple une opération CMISE. Néanmoins, comme nous l'avons vu, la technologie objet et ses promesses avec CORBA V3.0 ne nous offrent pas encore la possibilité de manipuler un modèle d'information sémantiquement très riche tel que la gestion de réseaux en aurait besoin [MZH99].
Comment en effet transformer les appels de méthodes typiquement orienté "objet" en requêtes de plus haut niveau pour pouvoir faire abstraction des "méthodes" à utiliser. Les avantages de tels abstractions étant de réduire l'impact de l'hétérogénéité des équipements des réseaux, et d'optimiser le dialogue entre le manager et ses agents.
Si cette question se pose au niveau technologie de programmation, elle se pose aussi au niveau de la conception comme [Lew99] nous le fait remarquer. Dans le processus de développement de services, il pose la question de la correspondance entre les composants de gestion et les services. Le composant étant une entité logiciel capable d'abstraire des données de bas niveau, comme la quantité de ressources utilisée, le débit maximal, les informations utilisés par le service comme la disponibilité du service. L'auteur propose de donner à différent types d'agents la responsabilité de cette correspondance, chaque type d'agent ayant une vue différente de son environnement. Des agents ayant une vue sur les équipements de base comme les commutateurs, aux agents entreprises ayant une vue sur l'utilisation de l'ensemble des ressources.

Une autre approche intéressante nous est proposée par [Oli98] qui consiste à utiliser des ***agents intelligents*** pour veiller à la satisfaction des utilisateurs. L'autonomie de ces agents est poussée aux maximum dans le sens ou ils travaillent sans l'autorité d'un manager. Ils utilisent un langage de haut niveau pour s'échanger des requêtes de type objectifs à atteindre.

Le concept d'agent intelligent que nous détaillerons dans le prochain chapitre, est aussi référencé par [MZH98] dans leur étude sur les principaux paradigmes de distribution utilisable dans le cadre de la gestion de réseaux. Ces agents intelligents sont vu comme étant *orienté objectif*, c'est-à-dire qu'ils ont une totale autonomie de moyens pour atteindre les objectifs qui leurs sont fixés.

L'intérêt lié aux possibilités qu'offrent les agents intelligents n'est pas uniquement perçu et compris par quelques chercheurs isolés. En 1999 lors de la conférence Distributed Systems: Operations and Management qui s'est tenu à Zurich, il a été reconnu que les agents intelligents faisaient partie des approches permettant de répondre aux besoins qu'on les opérateurs, les fournisseurs de services ou même les entreprises d'avoir un réseau "auto adaptatif". Depuis quelques années cette "nouvelle technologie" prend une place croissante dans les conférences spécialisées sur la gestion de réseaux, et un workshop spécifique sur les agents intelligents pour les applications de

télécommunications et de gestion de réseaux a lieu régulièrement depuis 1996 dans le cadre de l'European Conference on Artificial Intelligence (ECAI).

Il est dès lors assez tentant de voir en l'agent intelligent la prochaine évolution de la technologie de conception de logiciels (voir figure 10). Mais à la lecture des publications de plus en plus nombreuses sur le sujet, il apparaît vite évident qu'une grande confusion

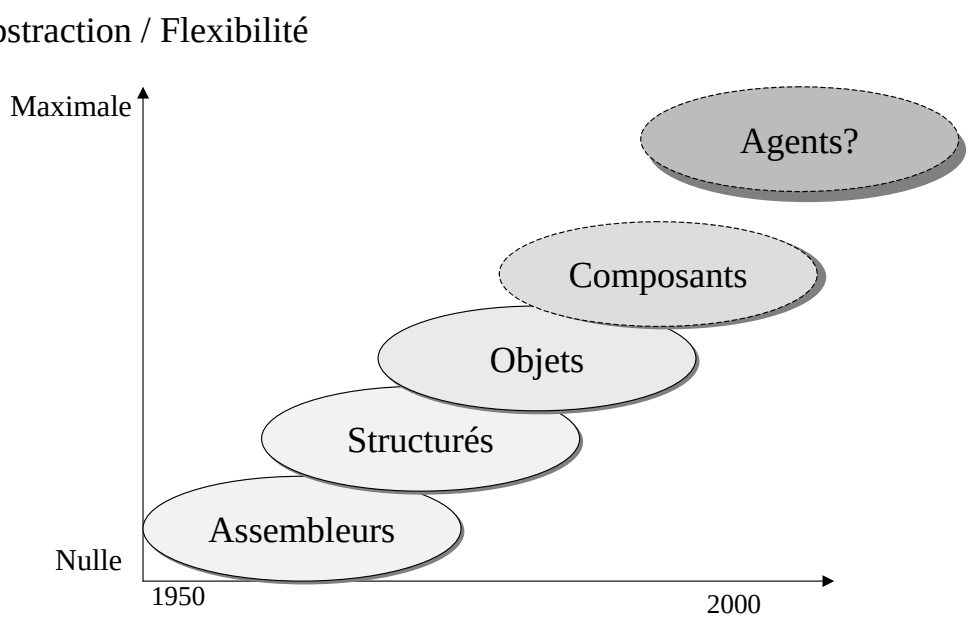


figure 10 Evolution des abstractions dans les technologies de programmation

existe dans la définition de l'agent, ou plutôt qu'il existe une multitude d'agents.
La question que nous devons dès lors nous poser, est: existe-t-il réellement une technologie "agent intelligent" ou au moins les bases pour une telle technologie?
Nous avons déjà, dans ce document, évoqué les agents de la gestion de réseaux, les agents mobiles du monde objet, que sont donc les agents intelligents, et apportent-ils réellement une solution aux problèmes de la gestion de réseaux?

III Principes et architectures des SMA

Si l'on en croit [Woo99], le seul nouveau et important paradigme qui ait émergé du domaine du logiciel depuis le début des années 1990 est celui des agents intelligents ou systèmes multi-agents (SMA). Pourtant, au début de l'année 2000, ce paradigme ne semble pas être encore sorti des laboratoires de recherche, et ce, malgré le nombre impressionnant de publications concernant les agents intelligents. Parmi ces publications une partie importante concerne l'utilisation de ce paradigme pour solutionner les problèmes rencontrés dans le domaine des télécommunications et de la gestion de réseaux.
Nous allons voir dans ce chapitre l'essentiel de ces travaux, en expliquant l'enjeu que représente ce paradigme pour la gestion de réseaux, mais avant tout, nous présentons les origines des agents intelligents que [WJ95] décrivent comme étant un système possédant les propriétés suivantes:

- Autonomie: opérant sans une intervention humaine, et possédant un contrôle sur ses actions et la connaissance de son état.
- Sociabilité: coopérant avec d'autres agents intelligents ou des êtres humains pour atteindre ses objectifs, en utilisant un langage de communication.
- Réactivité: répondant aux changements de son environnement de manière appropriée
- Proactivité: prenant des initiatives pour atteindre ses objectifs, et sans attendre la survenue d'un événement extérieur.

III.1 Agents intelligents, le paradigme

En observant l'évolution des capacités informatiques à la fois en terme de puissance, de calcul et en terme d'applications, il n'y a aucun doute que nous approchons de l'ère du logiciel intelligent, qui travaillera pour nous, et prendra des décisions en notre nom sans un contrôle permanent de notre part. Cette section a pour but d'introduire le concept relativement nouveau d'agent intelligent, de rechercher ses origines et d'expliquer ses principales caractéristiques.

III.1.1 Origines

Lors de nos recherches nous avons trouvé pléthore d'agents différents. Parmi les plus connus les agents mobiles, les agents personnels, mais aussi beaucoup d'agents dit intelligents comme les « softbots » qui observent notre comportement et copient celui-ci pour nous présenter des informations pertinentes. Mais à une époque où la plupart des entreprises de logiciels ont jeté leur dévolu sur le monde objet et formés leurs équipes à la conception et au développement objet, pourquoi semble-t-il y avoir de plus en plus d'intérêt pour ces agents et comment expliquer leur diversité? La lecture des publications faites sur ce sujet nous éclaire sur l'intérêt que présente ce paradigme, et sur la diversité de ces agents qui s'explique par leurs origines que nous pouvons classer en trois domaines principaux : l'intelligence artificielle distribuée, le développement objet, la gestion de réseaux.

III.1.1.1 Intelligence artificielle distribuée

Au début des années 1980, l'intelligence artificielle a été perçue comme la solution la plus prometteuse pour résoudre les problèmes complexes que l'on ne pouvait traiter en utilisant les concepts et outils de programmations structurées. De nombreuses études ont été menées pendant une dizaine d'années sur le développement de systèmes experts. Les types de raisonnements ont été améliorés ou affinés, et de nouvelles logiques comme la logique floue [Zad84] ont été développées pour découvrir finalement qu'au delà d'une certaine taille de problème, la complexité du système expert à développer est supérieure à la complexité du problème lui-même [Bro86]. Face à ce constat, des chercheurs du domaine de l'intelligence artificielle ont compris l'intérêt de travailler sur des systèmes plus simples, plus flexibles et plus robuste, [Bro86]. Ces concepts se rapprochent des études sur le comportement social des insectes et plus particulièrement des fourmis à partir duquel il a été prouvé qu'un ensemble de systèmes simples pouvait avoir un comportement global intelligent. Plus précisément, il a été prouvé « qu'un groupe d'agents présente une capacité supérieure à la somme des capacités de chacun de ses membres » [Mul96].

Ces résultats sont issus de nombreuses années de recherche puisque l'idée du partage et de la distribution des connaissances et des capacités de raisonnement ont fait l'objet de recherches dès 1980 [Hat91]. Bien que l'intelligence artificielle distribuée (DAI) soit à

l'origine de systèmes multi-agents ayant des propriétés comme la coopération, on ne peut néanmoins considérer que ce soit là la seule origine des agents intelligents. En effet, d'autres domaines de recherche ont fortement influencé ce paradigme jusqu'à pratiquement en faire disparaître les éléments classiques de l'intelligence artificielle. Parmi ces autres domaines celui de l'approche objet est certainement le plus important.

III.1.1.2 Développement objet

A l'origine du succès du concept objet dans le monde du développement logiciel, il y a la modularité et la réutilisabilité du code développé. Le passage du mode artisanal à l'ingénierie du logiciel, et les études sur le cycle de vie du logiciel, ont fait découvrir qu'un des plus gros facteurs de coût était celui de la maintenance corrective et évolutive. L'idée d'avoir un code modulaire, dont on pourrait changer des éléments sans avoir à revoir toutes les spécifications et le codage, a permis d'imposer l'approche objet dans tous les gros projets. Pour finaliser et valider cette approche la dernière étape était la réunification et l'uniformisation des méthodologies de développement qui avaient été produites autour du concept objet. En 1991 [HB91] faisaient remarquer que "la conception d'une hiérarchie de classes constitue l'une des difficultés majeures de la programmation objet car il n'existe pas de méthodologie universelle de conception". C'est maintenant chose faite avec UML (Unified Modeling Language), méthodologie qui a en 1999 réunit les trois principales méthodologies utilisées:

- La méthodologie de Booch (Grady Booch)
- OMT, la méthodologie de James Rumbaugh
- La méthodologie de Ivar Jacobson

Mais en même temps que s'affirmait cette approche, le développement des réseaux entraînait l'évolution des systèmes d'informations de la centralisation vers la distribution. Le coût des réseaux diminuant ainsi que le prix des stations de travail et des ordinateurs personnels, il est apparu plus économique d'utiliser une puissance de calcul distribuée que centralisée. Puisque grâce aux objets les logiciels devenaient des assemblages d'objets, la distribution de ceux-ci devenait envisageable. L'OMG a été créé sur cette idée afin de fournir un support standard pour l'accès aux objets distribués.
L'application étant maintenant un ensemble d'objets distribués, il devenait possible et intéressant de leur donner une certaine autonomie d'exécution. En tenant compte de cette autonomie les architectes peuvent ne définir que les interfaces de communication entre objets, et il devient envisageable de construire une application en utilisant des objets ou composants (groupes d'objets) fournis par différents vendeurs. Un autre avantage recherché est la mobilité du code qui doit permettre une mise à jour facile des composants de l'application. Des travaux sur les agents ont donc été lancés par l'OMG qui définit un agent comme étant "un logiciel qui agit d'une manière autonome au nom d'une personne ou d'une organisation".
D'autres organisations comme W3C (World Wide Web Consortium) et des entreprises comme IBM ont démarré des programmes de recherche sur ces thèmes.

Actuellement la majorité des travaux sur les agents (au sens général) issus du paradigme objet concerne les agents mobiles. Ces recherches sont essentiellement fédérées par CLIMATE/OMG (Cluster for Intelligent Mobile Agents for Telecommunication Environments). A la lecture des publications nous pouvons remarquer que les agents qui sont développés sous l'égide de l'OMG sont souvent présentés comme étant des agents intelligents, avec souvent comme seule raison leur autonomie. De telles affirmations participent à la confusion qui règne autour de la définition de l'agent intelligent comme nous le verrons par la suite. Cette confusion ne peut que s'amplifier lorsque l'on parle avec CORBA du comportement des objets ou du comportement des composants.
Pourtant nous pouvons faire la distinction entre le comportement:

1. d'un objet, dont une définition nous est donnée dans [MAZ97]: « Le comportement d'un objet caractérise l'ensemble des changements d'états possibles qui peuvent l'affecter et définit l'ensemble des actions potentielles auxquelles l'objet peut prendre part »
2. d'un agent, en considérant qu'un agent possède le contrôle de ses actions, et peut de ce fait démarrer ou stopper des actions sans une demande directe extérieure.

Pour marquer la différence entre les objets et les agents, il est aussi dit "qu'un agent peut utiliser un objet, et qu'un objet ne peut utiliser un agent". Les agents ne sont donc définitivement pas des objets, mais tout comme il est possible de faire de la programmation objet avec un langage structuré, il est possible (et cela est souvent fait) de programmer des agents intelligents en utilisant des langages objets.

III.1.1.3 Gestion de réseaux

La troisième source importante de recherche sur les agents intelligents est la gestion de réseaux où l'on utilise un concept d'agent un peu différent depuis plus d'une dizaine d'années. Les agents SNMP ou CMIP sont des programmes qui récoltent et enregistrent les informations nécessaires à la gestion des équipements sur lesquels ils se situent généralement. Sous le contrôle de managers avec qui ils communiquent au travers de protocoles du même nom, et effectuent des opérations de gestion. Des propositions pour rendre ces agents intelligents ont été faites [Par95] mais sans une revue complète des principes classiques de la gestion de réseaux, il est à notre avis impossible de résoudre les problèmes que nous avons identifiés. Le développement des réseaux a modifié la manière dont nous utilisions les moyens informatiques. Les ressources sont distribuées à travers le réseau sur des équipements hétérogènes et peuvent être partagées. L'approche client serveur à un seul niveau doit donc être revue, et conscientes des nouvelles possibilités qui s'ouvraient, des entreprises comme Oracle ont pressenti l'approche agent comme étant LA solution du futur et proposent maintenant une architecture serveur/agent/client.
Depuis quelques années maintenant, il est admis que l'intelligence des systèmes de gestion doit être distribuée [STE95], et la distribution de cette intelligence a utilisé plusieurs formes depuis la présentation du principe des serveurs élastiques [YEM91], des agents élastiques [GOL95], jusqu'aux agents mobiles.

III.1.1.4 Conclusion

En suivant les conférences de plus en plus nombreuses sur les agents intelligents ou en étudiant les publications, il est évident que de nombreux autres domaines de recherche ont été attirés par ce nouveau paradigme et l'ont aussi influencé à leur tour. Une des conséquences de ces multiples influences est qu'il est actuellement impossible d'avoir une définition commune reconnue de l'agent intelligent. A l'occasion de la conférence sur les théories agents, les architectures et les langages (ATAL), [Tok96] nous propose un article avec un titre provocant "un agent est un individu qui a une conscience". Mais malgré les longues discussions sur ce sujet, la seule conclusion a été qu'il n'était pas urgent de trouver une définition unique de l'agent intelligent.
En prenant un peu de recul, on peut prévoir qu'une convergence devra se faire ne serait-ce que pour des raisons économiques, et que des groupes actifs comme l'OMG peuvent imposer leur point de vue, au risque de compromettre l'apparition d'une nouvelle technologie. Conscient de cette réalité les chercheurs et industriels se sont regroupés autour d'organisations comme la FIPA (Foundation for Intelligent Physical Agents) avec comme objectif de proposer des normes autour desquelles les industriels pourraient développer leur propre technologie agent.
En attendant le succès de ces démarches et pour cerner ce que l'on entend par agents intelligents nous pouvons décrire les propriétés de ceux-ci.

III.1.2 Propriétés

Les agents sont donc des entités logicielles, plus communément des programmes qui vont être caractérisés par leurs propriétés.
Certains pourront considérer le résultat infructueux des tentatives de définition commune de l'agent intelligent comme un échec. Mais il faut admettre que les définitions comme celle de l'intelligence amènent un tel degré de subjectivité qu'il est impossible de réunir toutes les interprétations.
De la même manière, la définition des propriétés peut être sujet à interprétation comme le prouvent les nombreuses discussions sur la mailing liste des agents [agents@cs.umbc.edu] concernant l'autonomie ou la réactivité.
[FG96] ont proposé une taxonomie des agents en fonction de leurs propriétés et du monde dans lequel ils ont été produits. Nous nous contenterons ici de la définition des propriétés qui sont le plus couramment rencontrées et nous discuterons de leur intérêt lors de l'étude de l'architecture DIANA.
Avant de donner ces définitions voici un exemple qui permet d'apprécier la relativité de ces propriétés.
Considérons un agent en charge de la surveillance d'un routeur. Maintenant plaçons nous en tant qu'observateur de l'agent et du routeur sur lequel il s'exécute. L'agent est donc vu comme une boîte noire, et nous observons les deux cas suivants. Dans le premier, on observe à un moment donné un lien qui tombe. Aussitôt l'agent reconfigure le routeur pour utiliser le lien de secours et il prévient le système d'administration qu'il faut

intervenir. On peut raisonnablement en déduire que l'agent a un comportement réactif. Dans le deuxième cas (autre agent et autre routeur) on observe à un moment que l'agent reconfigure le routeur pour utiliser le lien de secours et qu'il envoie un message au système d'administration en indiquant que la configuration a été modifiée pour utiliser le lien de secours, car l'interface du premier lien était défaillante, que le lien allait tomber et qu'il faut envoyer un technicien en intervention. Dans ce deuxième cas on peut dire que l'agent à un comportement proactif, si ce n'est intelligent.
Maintenant revenons sur ces deux cas en nous plaçant au coeur (au cerveau?) de l'agent. Dans le premier cas celui-ci utilise l'indicateur de la MIB II *ifOperStatus* qui peut prendre les valeurs *up* et *down* (testing, unknow et dormant n'étant pas considéré ici) Il exécute à chaque modification de la valeur de l'événement la règle suivante "*si ligne down alors reconfigurer et prévenir"*.
Dans le deuxième cas il utilise un indicateur interne *interfaceState* qui donne une valeur proportionnelle de la dérivée du taux d'erreur enregistré par l'interface sur le temps (entre 0 et 1). Et il exécute à chaque modification de la valeur la règle suivante "*si interfaceState > 0,2 alors reconfigurer et prévenir"*. Dans ce deuxième cas l'action de prévenir est simplement un peu plus complète.
Nous pouvons admettre que dans ces deux cas en observant l'agent comme une boîte blanche, celui-ci manifeste un comportement réactif, alors qu'il apparaissait proactif dans le deuxième cas pour un observateur externe. Le point de référence est donc important et nous nous placerons en tant qu'observateur externe pour les définitions que nous allons donner.
Ces propriétés ont quelquefois été utilisées comme base pour créer des taxonomies d'agents [Nwa96][FG96], et globalement on peut considérer que les agents présentent tous un caractère d'autonomie, que la mobilité est une propriété accessoire, que la sociabilité implique de la communication au même titre que la coopération et la délégation. L'apprentissage, la réactivité, la proactivité, et la "délibérativité" sont des propriétés indépendantes qui peuvent être associées (figure).

III.1.2.1 Autonomie

"Qui fait preuve d'indépendance, qui se passe de l'aide d'autrui" (Hachette) L'agent décide quand, comment et sous quelle condition il va effectuer quelle tâche. [Fra96] nous propose la définition suivante: "un agent autonome est un système qui est situé et fait partie d'un environnement, qui perçoit cet environnement et agit sur lui, tout le temps, à la recherche de ses propres ordres du jour afin d'effectuer ce qu'il prévoit comme futur". Cette dernière définition a entraîné de nombreuses réactions dont [Woo96A] qui reproche à cette définition d'inclure les notions de réactivité et de proactivité qui selon lui ne sont pas liées à l'autonomie. Il nous propose à la place dans [Woo99] la définition suivante "l'autonomie veut dire sous son propre contrôle. Plus précisément l'assomption que l'agent n'agit pas sous l'intervention directe d'un humain, bien qu'il soit considéré comme agissant pour notre compte, et qu'il garde le contrôle de son état intérieur et de ses actions". Ayant préalablement choisi de ne considérer que la vision extérieure nous retiendrons la définition suivante de l'autonomie:

"capacité à agir sans nécessairement une intervention externe"

III.1.2.2 Communication

Une des propriétés les plus importantes et les plus étudiées des systèmes multi-agents est la communication. Notre point de vue est que la communication sous entend l'utilisation d'un support de transport de l'information, d'un protocole, d'un langage et d'ontologies. Les premiers systèmes multi-agents ont été créés pour fonctionner sur une seule machine et utilisaient des mécanismes de communication où ces trois composants étaient difficilement identifiés. C'est le cas par exemple de la communication par "tableau noir". Le protocole y est réduit au contrôle d'accès de l'information du tableau noir, et dans ce cas langages et ontologies sont souvent confondus. L'utilisation de systèmes physiquement distribués a clarifié cette distinction, bien que certains discutent encore de la différence entre un protocole et un langage. C'est pour cela que l'ACL[4] (Agent Communication Language) de la FIPA est considéré pour beaucoup comme un langage alors qu'il ne sert qu'à contrôler les échanges d'informations entre agents sans a priori sur le contenu. ACL.
De même pour KQML (Knowledge Query and Manipulation Language) qui provient des travaux sur les langages acteurs issus eux-mêmes des recherches sur l'intelligence artificielle distribuée. Si ces langages permettent le transport de l'information d'un point à un autre et utilisent une sémantique bien définie, le contenu de ces échanges, c'est-à-dire l'information elle-même doit être codé en utilisant un autre langage. Nous pouvons comparer le dialogue entre deux agents et deux personnes. Supposons que vous vouliez appeler une personne en Chine. Vous allez utiliser le téléphone et composer un numéro de téléphone qui va vous permettre de transporter votre voix et d'échanger des messages. Le téléphone est alors utilisé au même titre que le protocole de transport comme par exemple TCP/IP entre les deux agents. Maintenant votre correspondant décroche et vous devinez qu'il vous répond par l'équivalent d'un "allo". Si vous ne parlez pas la même langue, vous pourrez néanmoins comprendre que cette personne vous pose des questions ou vous ordonne quelque chose au ton de sa voix. C'est un peu le rôle d'ACL ou de KQML qui permet à deux agents de comprendre que quelque chose est demandé, mais sans savoir quoi si le langage de contenu (le chinois ou le français) n'est pas partagé. Il y a bien un transport de message grâce à la voix ou à ACL et une certaine compréhension de l'intention grâce à l'intonation ou aux primitives (ACL), mais pas de compréhension du contenu. A ce titre nous considérons définitivement ACL ou KQML comme un protocole d'échanges d'informations, et non pas comme un langage au sens humain du terme.
Dans la plupart des cas le langage agent est développé d'une manière ad hoc. Car il est actuellement difficile de développer des systèmes multi-agents, et il est très improbable que deux agents d'origine différente aient à échanger des informations.
Pourtant certaines études ont été menées sur le transfert de connaissance, principalement autour du langage KIF (Knowledge Interchange Format). KIF est un langage déclaratif comme Lisp qui définit à la fois une sémantique et une syntaxe, permettant de décrire des

[4] Ce langage a été développé par une équipe du CNET Lanion sous le nom d'Arcol.

informations complexes, qui reste quand même peu utilisé pour les raisons évoquées précédemment.
Si comme nous l'avons dit il y a encore beaucoup de discussions sur ce qu'est un langage dans le cadre des systèmes multi-agents, la définition de la communication est, quant à elle, simple et indiscutable:

> *"faculté de transmettre des informations compréhensibles par les participants"*

III.1.2.3 Réactivité

[Oli98] nous indique que la réactivité d'un agent est la propriété qui le différencie clairement d'un système expert dans le sens où c'est une propriété qui implique que celui qui l'utilise soit en relation avec son environnement. [Woo99] conteste cette affirmation en citant le cas de systèmes experts qui ressemblent fortement à des agents comme par exemple ARCHON [JCLMSV96], et il nous propose une définition de la réactivité que nous adoptons:

> *"capacité qu'a un agent de percevoir son environnement et à répondre aux changements qui y interviennent"*

III.1.2.4 Proactivité

Nous avons déjà quelque peu abordé la propriété de proactivité en introduction de cette section. [Woo99] nous propose comme définition "le comportement orienté objectif, par prise d'initiative, afin de satisfaire les objectifs de sa conception". Dans [CCOL98] nous avons fait remarqué qu'il y avait souvent une assimilation du comportement proactif et délibératif (que nous verrons plus loin). Cette définition en est un exemple et nous ne pouvons l'accepter. En effet le Larousse nous indique que la proactivité caractérise un phénomène "qui s'exerce d'amont en aval dans le temps" cela sans tenir compte d'une prise quelconque d'initiative qui se rattachera elle à la notion de délibération. La définition de la propriété de proactivité que nous utilisons est la suivante:

> *"capacité qu'a un agent de percevoir et d'agir sur son environnement en anticipant les changements de celui-ci"*

III.1.2.5 Délibération

[WJ95] définissent un agent délibératif comme "contenant une représentation explicite d'un modèle du monde, et à partir duquel des décisions sont prises via un raisonnement logique (ou au moins pseudo-logique), basé sur du pattern-matching et la manipulation de symboles". [Nwa96] classe les agents en deux catégories principales: réactive et délibérative. Pour lui la propriété de délibération provient des recherches sur le raisonnement en intelligence artificielle où l'agent doit posséder un modèle de

raisonnement interne symbolique qu'il utilise pour des opérations de planification et de négociation pour accomplir une action coordonnée avec d'autres agents. Il nous rappelle cependant que [Bro91] a prouvé que des agents peuvent exhiber des comportements délibératifs sans utiliser des représentations symboliques. Ce dernier dénonce dans cet article l'affirmation qui veut que seule l'intelligence artificielle classique est capable de traiter les comportements intelligents. Nous utilisons la définition suivante, qui n'implique pas l'utilisation d'une technologie provenant du domaine de l'intelligence artificielle:

> *"capacité qu'a un agent d'utiliser ses connaissances et ses capacités afin d'atteindre les objectifs qui lui sont fixés "*

III.1.2.6 Apprentissage

La propriété d'apprentissage est un des plus vieux objectifs de l'intelligence artificielle, car elle est souvent considérée comme un préalable à la capacité d'adaptation. Il y a classiquement deux niveaux d'apprentissage. Le premier se réfère à la capacité d'engranger de la connaissance, le deuxième correspond à la capacité de produire de nouveaux comportements sans programmation manuelle en utilisant l'expérience passée. Une autre approche de l'apprentissage est envisagée par [Wei99], c'est l'apprentissage distribué. Dans ce cas, chaque agent d'un système multi-agents participe à un objectif commun d'apprentissage.
Si dans le domaine de l'intelligence artificielle l'apprentissage est un sujet souvent traité, ce n'est pas le cas dans le domaine des agents, excepté les très spécifiques agents personnels. [Bro91a] nous donne la définition suivante que nous adoptons:

> *"l'apprentissage correspond au développement de méthode pour l'acquisition automatique de nouveaux comportements et l'amélioration des comportements existants"*

III.1.2.7 Délégation

La délégation joue un rôle particulièrement important dans notre contexte puisque comme nous l'avons vu, c'est aussi un principe applicable et recherché dans le cadre de la gestion de réseaux. Malgré tout cette propriété n'apparaît pas dans [ABC98] et n'est pas traitée dans le volumineux [Wei99] où elle est implicitement incluse dans les études sur la coopération. De plus cette propriété est implicite dans un bon nombre de définitions de l'agent autour du principe suivant: "un agent est un logiciel qui agit au nom d'une autre entité" [ABC98] [Oli98]. Nous reviendrons plus en détail sur les différents aspects de cette propriété, mais nous pouvons déjà retenir la définition suivante:

> *"capacité qu'a un agent d'exécuter une opération, ou d'atteindre un objectif pour le compte d'un autre. La délégation implique deux rôles: celui de délégateur, et celui de délégué."*

III.1.2.8 Coopération

La coopération est contrairement à la délégation une propriété qui a été souvent étudiée. La raison principale est qu'elle est partie intégrante du domaine de l'intelligence artificielle distribuée. Une coopération peut être coordonnée ou désorganisée, négociée ou imposée [Fer95]. Pour certains auteurs comme pour nous, la coopération doit être considérée de l'extérieur, c'est-à-dire par un observateur qui n'a pas accès aux états mentaux (notion d'intelligence artificielle) des agents [DLC89] [Bou92].
[MIR92] introduit un indice de coopération permettant de distinguer: la coordination des actions, le degré de parallélisation, la robustesse, l'unicité des actions (non redondance), et la non persistance des conflits (deadlock). Dans ce document nous considérons la coopération comme la:

> *"capacité qu'a un groupe d'agents de travailler ensemble pour atteindre un objectif"*

III.1.2.9 Conclusion

Il existe d'autres propriétés utilisées pour définir les agents intelligents, comme la mobilité ou la capacité de négociation que certains considèrent comme primordiales. A ce sujet nous montrerons dans cette thèse que la mobilité n'est qu'un moyen utilisable par les agents intelligents, et que l'approche et l'architecture que nous proposons dispensent les agents de cette propriété.
Suivant l'*origine* des agents proposés, c'est-à-dire, le domaine de l'IA, celui de l'objet ou de la gestion de réseaux, l'importance de ces propriétés est différente. Dans le domaine de l'IA, c'est avant tout le comportement de l'agent au travers de ses capacités de délibération ou d'apprentissage, qui est mis en valeur. Les agents conçus dans le cadre de la recherche influencée par l'approche objet auront généralement comme première propriété la mobilité. Cette constatation est moins évidente en ce qui concerne les travaux menés dans le contexte de la gestion de réseaux et des télécoms, ces domaines étant eux-mêmes déjà influencés par l'approche objet. On trouvera donc beaucoup d'agents mobiles, mais aussi des agents capables de coopérer ou de négocier.

III.2 Principales architectures

Nous avons fait remarquer à plusieurs reprises qu'il existe une multitude d'agents intelligents différents, ce qui rend difficile la compréhension de ce domaine, et ce qui explique l'impossibilité de s'accorder sur une définition de l'agent intelligent. Néanmoins dans la section précédente nous avons vu les principales propriétés attribuées à ces agents intelligents, et retenu une définition pour chacune d'elles. Maintenant se pose le problème de la conception des architectures nécessaires à la réalisation de ces propriétés.

En tenant compte de la diversité des agents intelligents, on peut légitimement se demander s'il existe suffisamment de points communs entre ces agents pour dégager des architectures communes. Ce problème est exprimé autrement par [Mul98] qui propose de rechercher la bonne architecture agent en fonction de l'application à développer. L'auteur cite cinq types d'architectures:

1. Réactive
2. Délibérative
3. Interactive
4. En couche
5. Autres

Si les trois premières décrivent le comportement des agents intelligents, les deux autres décrivent leur structure interne. [Woo99a] différencie deux catégories d'architectures, les architectures abstraites et les architectures concrètes. Les premières étant basées sur la description du comportement des agents, les deuxièmes sur l'implémentation de ces comportements. Les deux études que nous venons de référencer portent ensemble sur plus d'une trentaine d'architectures différentes d'agents intelligents. Il semble donc tout aussi difficile de classer les architectures que les agents intelligents.

La réalisation d'un état de l'art exhaustif sur les agents intelligents est pratiquement impossible en raison de l'engouement que connaît ce domaine. La plupart des domaines liés à l'informatique semblent en effet s'intéresser à ce, ou plutôt à ces concepts, et il existe des projets sur les agents intelligents aussi bien dans le domaine de l'intelligence artificielle, des bases de données, de la conception objet, de la gestion de réseaux et télécommunications, que dans de nombreux autres domaines.

Nous avons choisi de présenter ici les trois types d'architectures que nous avons déjà présentées dans [CCOL98] et que nous considérons comme les plus importantes dans le sens où elles sont applicables dans le cadre de la gestion de réseaux; il s'agit des architectures réactive, délibérative, et hybride. Ces trois architectures partagent quelques concepts de base.
Le premier est celui de *l'environnement* de l'agent qui est perceptible au travers de senseurs, et sur lequel l'agent va pouvoir agir grâce à des effecteurs, quelquefois appelés actionneurs (voir figure 11).

Le deuxième est le concept *d'objectifs*. C'est-à-dire que les agents vont agir de manière à remplir les objectifs qui leur sont assignés.

Et le troisième concept, le concept de *couches*, est un concept commun aux trois architectures et concerne leur implémentation. Chaque couche étant en charge de traiter des abstractions différentes. Sur ce dernier point, il faut toutefois signaler que quelques spécialistes de l'intelligence artificielle sont opposés à une représentation en couche des architectures délibératives telle que l'architecture BDI (Belief Desire Intention).

[Woo99a] argumente par contre en faveur de cette représentation en la considérant comme naturelle, car elle permet une décomposition évidente en sous-systèmes ayant des

comportements différents. Nous pourrons constater que cette proposition est très proche du découpage par composants décrite par l'approche objet.

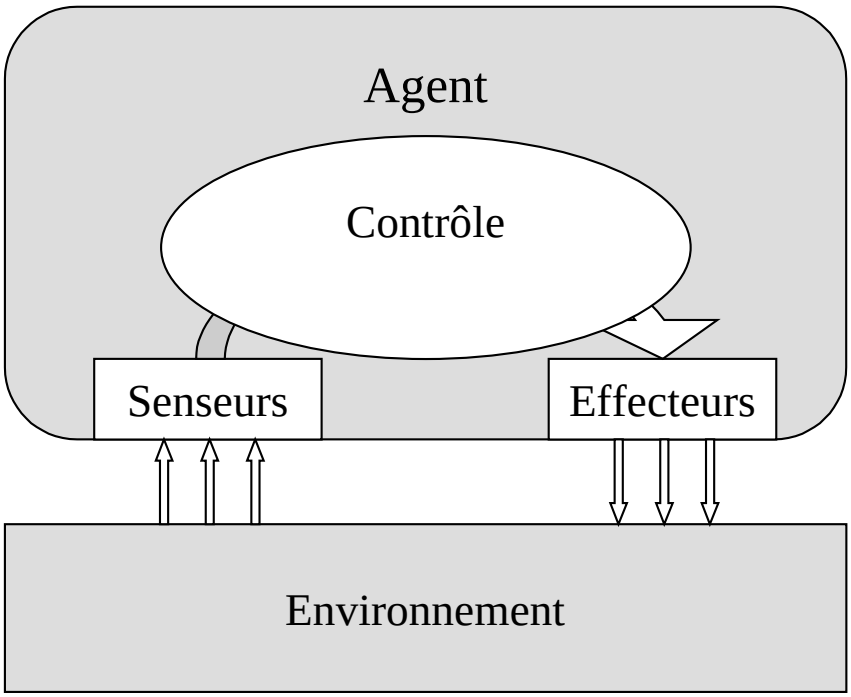


figure 11 Relation agent-environnement

III.2.1 Architecture réactive

L'architecture réactive est certainement l'une des plus utilisée pour le développement d'agents. Elle est à la base de la plupart des architectures d'agents mobiles, car elle ne nécessite qu'un minimum de logique de raisonnement. Elle est basée sur le principe:

$$situation \rightarrow action$$

Cette architecture est source de confusions, certains considérant que tout programme est réactif. Des logiciels se trouvent de ce fait présentés comme étant des agents intelligents, sous prétexte qu'ils sont mobiles et réactifs. A l'opposé d'autres considèrent que ne peuvent être des agents intelligents que des entités possédant une représentation symbolique de leur monde.
Comme tous les agents intelligents, l'agent réactif va agir pour atteindre ses objectifs. Pour cela [Bro91] [Bro87] suggère une architecture orientée activités composée de modules appelés couches, qui travaillent en parallèle. Chaque couche a une activité et un comportement différents, les couches les plus basses ayant les comportements les plus primaires, les couches plus hautes ayant un comportement plus élaboré. Si nous prenons le cas d'un robot qui doit se déplacer d'un point à un autre, les couches basses seront en

charge d'éviter les obstacles, tandis que les couches les plus hautes seront chargées de maintenir le "cap" vers la destination. L'idée qui est utilisée ici est qu'un comportement intelligent émerge de l'ensemble des comportement simples et réactifs. Brooks admet cependant qu'il faut prévoir un mécanisme capable de gérer les conflits qui peuvent se créer du fait que chaque couche travaille en parallèle. Une amélioration de cette architecture est proposée par [Ark90] qui y ajoute une gestion de plan. Ces plans sont des actions prédéterminées qui vont être choisies en fonction de la position connue (connaissance de l'environnement) du robot et de l'objectif à atteindre.
Plus proche de la gestion de réseaux [SR96] proposent une architecture réactive à base de règles de réaction qui sont utilisées par un moteur d'inférence. Les réactions sont considérées comme des services et sont liées aux événements sous la forme <event-id, reaction-id>.
Et plus généralement nous pouvons comprendre qu'une architecture réactive se caractérise par l'utilisation par l'agent de la connaissance qu'il a de son environnement pour réagir aux événements. Plus précisément la réception d'un événement n'est source d'une réaction que si elle entraîne la modification de la situation que connaît l'agent et que la nouvelle situation est associé à une réaction:

$$\textit{événement} \rightarrow \textit{modification de la situation} \rightarrow \textit{action}$$

par opposition à:

$$\textit{événement} \rightarrow \textit{réaction}$$

Enfin l'architecture réactive suppose le contrôle des actions ou réactions à effectuer, c'est-à-dire la possibilité d'inhiber ou d'activer les réactions.

III.2.2 Architecture délibérative

Historiquement le concept de "délibération" provient du domaine de l'intelligence artificielle où s'est développée l'idée de raisonner sur des symboles. Ces symboles représentant l'état de l'agent (état mental) et celui de son environnement.
[Sin94] argumente en faveur de l'utilisation de représentations symboliques en disant que sans représentation, l'agent ne peut que réagir simplement aux modifications imprédictibles de son environnement qui sont provoquées par les autres agents, et être de ce fait inefficace. Il est donc pour lui, important voir indispensable, que les agents puissent s'échanger des informations sur leurs états, leurs intentions, et donc posséder cette représentation. C'est dans ce contexte que la plupart des architectures délibératives sont construites, c'est-à-dire sur la base des architectures BDI (Belief Desire Intention). Pour certains auteurs il n'existe d'ailleurs pas d'architecture délibérative, mais uniquement des architectures BDI comme on peut le constater en lisant [Wei99].
Pour [Mul98a] il existe pourtant d'autres approches, mais toutes sont basées sur la manipulation des connaissances de l'agent. Ces recherches ont pour objectif de déterminer les mécanismes de raisonnement utilisables par un agent ou un système d'agents. Par exemple la planification est un mécanisme qui permet à l'agent de

sélectionner les actions à entreprendre en fonction des conditions initiales (l'état de l'agent et de son environnement) et de l'état à atteindre. Dans [Mae89] P. Maes propose une architecture délibérative basée sur ce qu'elle appelle des modules de compétences qui correspondent à des opérateurs classiques en IA. Chaque opérateur possède une précondition, un résultat et une action qui consiste à rajouter ou supprimer une liste dans un ensemble. L'algorithme décrit permet de rechercher la suite logique des actions à entreprendre pour atteindre un objectif qui décrit un état à atteindre.
Cette architecture est donc différente de l'approche délibérative BDI promue par les spécialistes de l'intelligence artificielle, puisqu'elle ne tient pas compte des intentions ou désirs de l'agent. Les architectures BDI sont d'un autre côté critiquées par Rao et Georgeff. Pour ces derniers les logiques modales utilisées dans les architectures BDI n'ont pas d'axiomatisations complètes, et de ce fait ne sont pas efficaces, ce qui explique qu'il n'y ait pas d'implémentation correcte disponible d'une telle architecture [RG92].
Ces chercheurs proposent de simplifier le modèle en restreignant les assomptions aux *beliefs* (information permettant de décrire la connaissance qu'a l'agent de son état et de son environnement) et *goals* (informations décrivant les objectifs de l'agent) pour obtenir une architecture pratique c'est-à-dire réalisable. Afin de rester proche des concepts initiaux du BDI, les auteurs proposent de considérer les intentions comme étant implicites dans la pile d'exécution de l'agent.

Globalement nous pouvons décrire qu'une architecture délibérative est une architecture incluant un mécanisme de sélection des actions à exécuter pour atteindre un objectif.

Un peu plus complexe, l'architecture délibérative peut être considérée comme l'enchaînement:

$$objectif \rightarrow recherche\ de\ la\ meilleur\ action \rightarrow action$$

III.2.3 Architecture hybride

Intuitivement l'architecture réactive semble plus adaptée à la réalisation de comportements temps réels, tandis que l'architecture délibérative est plus efficace pour résoudre des problèmes plus complexes. Pour obtenir une architecture capable d'exhiber à la fois des comportements réactifs et délibératifs, des chercheurs ont proposé un nouveau type d'architecture, l'architecture hybride. En utilisant le concept d'implémentation en couches, ils proposent qu'une couche délibérative soit en charge de "programmer" des comportements réactifs dans la couche réactive.
Les avantages de la conception en couches sont d'après [Mul98a]:

- ▷ Le support de modules fonctionnels clairement séparés et liés par des interfaces bien définies
- ▷ Une conception plus compacte et plus robuste facilitant le débugage
- ▷ Une capacité de calcul qui peut être accrue du fait que chaque couche peut s'exécuter en parallèle

- ▷ Amélioration de la réactivité, car la couche réactive n'est plus utilisée pour des opérations complexes
- ▷ Une conception plus facile du fait que chaque couche peut gérer sa propre base de connaissances

[Wag96] présente des agents appelés Vivid possédant une architecture hybride, dont la programmation s'effectue en décrivant des règles d'actions et des règles de réactions. L'auteur indique que l'intérêt d'une telle architecture par rapport à une architecture délibérative est que l'agent peut réagir directement à la perception d'un événement sans considération de son état interne. Un des inconvénients majeurs de l'architecture proposée est que l'agent ne peut travailler que sur un objectif à la fois, et peut passer par conséquent beaucoup de temps sur un objectif peu important alors qu'un objectif plus important, et donc plus prioritaire, sera ignoré.
[Mul96] propose avec INTERRAP une architecture hybride à trois couches, une troisième couche étant ajoutée au dessus de la couche délibérative, appelée couche de comportement. Cette troisième couche est chargée de coordonner l'exécution de tâches en coopération avec d'autres agents INTERRAP.

Dans la majorité de ces architectures la couche basse possède les senseurs et actionneurs, possède un mécanisme réactif, et est chargée d'alimenter la couche supérieure en connaissances (beliefs) . La couche supérieure ne peut donc pas agir directement sur l'environnement de l'agent.

III.2.4 Conclusion

L'architecture des agents est essentielle pour déterminer les capacités que possédera le système multi-agents [CCOL98][Mul98]. Il est important de souligner ici que la plupart des travaux sur les SMA ont été faits dans des domaines très différents du domaine de la gestion de réseaux, et les résultats obtenus ne sont pas applicables à ce dernier. Par exemple les systèmes multi-agents dont le mécanisme d'échange est un tableau noir sont inutilisables en raison de l'augmentation de la complexité, et de la consommation des ressources en communications lorsqu'on passe dans un véritable système distribué. D'autres systèmes n'autorisent qu'un fonctionnement synchrone et ne supportent pas de traitement en parallèle ce qui réduit considérablement leur intérêt.
Il existe aussi d'autres architectures décrivant non pas un type d'agent, mais un système multi-agents autorisant par exemple le déplacement de ces derniers (agents mobiles). Ce côté obscur des agents sera traité plus loin dans cette thèse.

Enfin, il faut rappeler que la distinction que nous venons de faire entre les différents types d'architectures est avant tout conceptuelle, car la très grande majorité des agents intelligents dont nous avons parlé sont restés sous la forme d'études théoriques, et au mieux sous la forme de prototypes. Le choix de l'architecture, s'il est essentiel comme nous l'avons dit, ne garantit pas pour autant la validité de l'implémentation qui en sera faite.

III.3 Agents intelligents pour la gestion de réseaux

Nous connaissons maintenant un peu mieux les agents intelligents au travers de leurs origines, propriétés, et architectures. Pour restituer les agents intelligents dans le contexte qui nous intéresse, c'est-à-dire la gestion de réseaux, cette section présente quelques travaux qui ont été menés plus spécifiquement dans ce cadre, et qui ont influencé notre approche.

III.3.1 Contexte

En raison de la diversité des travaux sur les agents intelligents, il est pratiquement impossible de faire un état de l'art exhaustif de ce domaine. Des tentatives ont été réalisées [CCOL98], mais la plupart du temps les auteurs ont principalement listé les expérimentations rendues publiques. L'excellent tour d'horizon que nous proposent [HB98b] n'échappe pas à ce constat. Après avoir classé des travaux en fonction du domaine d'application (ex: réseaux intelligents, gestion de réseaux), en fonction de quelques propriétés (ex: négociation, mobilité), en fonction des protocoles de communications (ex: ACL), ou même des concepts (ex: gestion par délégation), les auteurs citent directement des projets sans les classer.
D'un autre côté une grande majorité des articles publiés dans le domaine de la gestion de réseaux par agents intelligents concerne les agents mobiles qui ne sont pas forcement des agents intelligents, comme nous l'avons déjà fait remarquer et nous reviendrons d'ailleurs sur cette distinction importante. Il existe aussi des publications concernant des travaux théoriques dont les concepts et les champs d'application correspondent bien à notre domaine, mais qui manquent encore de maturité pour être applicables en dehors de quelques études de cas précis.
Par exemple [FF98] qui proposent une décomposition du réseau en vues abstraites et hiérarchiques permettant d'avoir une vue synthétique et facilement utilisable des ressources disponibles en bande passante. L'idée correspond parfaitement à l'utilisation effective des agents intelligents, et leur solution doit permettre de résoudre les questions d'arbitrages nécessaires quand les agents n'ont qu'une vue locale des ressources. Cette théorie est basée sur une technique d'abstraction appelé "*blocking Islands*" et qui permet dans cet article de déterminer la disponibilité d'une route dans un réseau ATM en ayant une contrainte sur la bande passante. L'approche agent proposé permet de construire des arbres de sélection dont chaque noeud peut être un agent en relation avec un père et des fils, qui lui transmettent les informations sur les ressources disponibles. La décomposition du réseaux en niveaux d'abstractions résumant les ressources disponibles entre deux noeuds, permet de rechercher une route d'une manière distribuée, efficace et optimisé. Le problème qui n'est malheureusement pas traité est celui de la multiplicité des paramètres de QoS utilisables avec ATM. Si un utilisateur demande non seulement une bande passante particulière, mais aussi un délai maximum de transmission, la solution proposé perd tout sont intérêt. En effet il faudra prévoir autant de hiérarchies de *blocking Islands* que de paramètres de QoS, ainsi qu'un mécanisme de sélection

supplémentaire pour vérifier qu'il existe une route commune parmi les routes retournées par les agents en charges des *blocking islands*, et enfin déterminer la meilleure route.

III.3.2 Mobilité

Les agents mobiles quant à eux représentent une tendance "forte" de la recherche sur les agents intelligents. Forte, car il existe déjà des produits industriels comme Voyager d'ObjectSpace, ou des produits issus des laboratoires de recherche et disponible à la communauté scientifique comme les aglets d'IBM Tokyo Research Laboratory [LC97], qui permettent de développer des agents mobiles. Forte aussi dans le secteur de la gestion de réseaux, parce que le concept d'agent mobile semble être le meilleur support pour gérer la gestion par délégation [Mag95].
Par exemple [PLBS98] proposent d'utiliser des agents mobiles pour configurer des PVCs (Permanent Virtual Circuit) dans un environnement ATM hétérogène. L'agent mobile utilise une infrastructure particulière et standard qui doit être fourni par l'équipementier sur chaque commutateur ATM pour prendre connaissance de la configuration existante et la modifier. Les auteurs posent comme axiome que les constructeurs qui n'ont pas pu se mettre d'accord sur l'utilisation d'une MIB standard, vont fournir une interface standard appelée VMC (Virtual Managed Component) pour les agents. En supposant que cela soit réalisé, l'utilisation des agents mobiles pour configurer un réseau ATM est peu performante, car les délais sont longs, et les agents mobiles consomment plus de bande passante que ne le ferait un système d'agents intelligents fixes. Les agents n'ayant a priori pas connaissance de la route optimale, ni même la possibilité de réserver des ressources, vont devoir en effet se déplacer d'un commutateur ATM à un autre, configurer et revenir sur leurs "pas" s'ils sont à un moment bloqués par le manque de ressources.
Des projets exploratoires importants comme le projet d'EURESCOM (European Institute for Research & Strategic Studies in Telecommunications) P712 intitulé "Intelligent and Mobile Agents and their Applicability to Service and Network Management" qui s'est terminé en 1999, montrent des résultats mitigés [CGH00] [CTC98] beaucoup de questions restant en suspens.

[HB98b] nous rappellent que les agents mobiles sont particulièrement intéressants lorsqu'ils sont utilisés dans le cadre de communications non permanentes. Dans ce cas en effet un agent mobile peut continuer à se déplacer et travailler alors que l'administrateur du réseau n'est plus connecté. C'est un avantage que la société Oracle utilise pour proposer des opérations de transactions non connectées.
Malgré tout, l'agent mobile souffre d'handicaps irréversibles. Le premier est son poids ou sa taille, comme l'on veut, qui ne doit pas être trop importante pour qu'il puisse se déplacer sans consommer toutes les ressources en bande passante. De ce fait son "intelligence" doit être réduite au maximum, de même que ses capacités de communication et sa connaissance des équipements de réseaux qu'il doit gérer. Le deuxième handicap est qu'il doit être surveillé pratiquement en permanence pour qu'on soit sûr que les tâches qu'il doit effectuer sont traitées. La fiabilité d'une application avec des agents mobiles est donc difficile à mettre en place quand le travail de l'agent est complexe. Il faut savoir ce qui a été réalisé et ce qui doit l'être encore en cas de

défaillance d'un agent, de configuration par exemple. Ce problème est mineur comme nous le verrons plus tard avec des agents statique.
Un troisième handicap est la quasi impossibilité de créer des agents mobiles coopératifs, en raison de la lourdeur des systèmes de communications inter-agents mobiles. Un autre handicap est la difficulté de sécuriser un agent mobile.
Le concept d'agent mobile présente néanmoins des avantages principalement par rapport à l'approche classique client serveur utilisée en gestion de réseaux comme nous le verrons par la suite. Une étude théorique sur l'utilisation du code mobile et des agents mobiles comme support à la gestion de réseaux décentralisée a été menée par [BGP97] [Pic98], et [Mag99] nous présente quant à lui l'ensemble des projets CLIMATE (Cluster for Intelligent Mobile Agents for Telecommunication Environments) .

III.3.3 Standards et technologies

La gestion de réseaux est en grande demande de standards, qui comme nous l'avons vu précédemment permettent de minimiser les risques de "non–interopérabilité" entre les protocoles, les équipements ou même entre les applications. Ce problème existe aussi pour les applications basées sur une approche agent, et c'est dans cet objectif de standardisation que la FIPA a été créée. Actuellement plusieurs normes ont été élaborées dont ACL (Agent Communication Language) déjà cité comme support au dialogue inter-agents. La société Nortel Networks diffuse depuis le début de l'année 2000 sa version de l'implémentation des spécifications FIPA, sous le nom de FIPA-OS. Le projet P815 d'EURESCOM a été développé autour des spécifications de la FIPA pour offrir un service de location de lignes télécoms [HKKOS99] dont l'un des objectifs était de vérifier l'intéropérabilité de différentes implémentations des spécifications FIPA. Il est actuellement très difficile de savoir si ces tentatives de standardisations ont un sens réel ou pas. En ce qui concerne par exemple le "langage" que doivent utiliser les agents, la communauté scientifique semble plus favorable à KQML, tandis que les industriels prônent ACL.
En ce qui nous concerne, la démarche de standardisation nous semble prématurée, car il faut avant tout dégager le concept d'agent intelligent de ses origines, pour en faire un paradigme clair et précis, et pour cela il faut tout d'abord être d'accord sur une définition et des caractéristiques précises.
A côté des standardisations on trouve un ensemble de technologies "agents", censées permettre de développer des SMA. Ces technologies varient du simple langage à l'environnement complet de développement ou framework. Dans les langages utilisés on trouve Telescript qui est sans doute l'un des plus vieux langages permettant de développer des agents sous forme de script, Jess, la version Java de Clisp qui permet de développer des systèmes experts, JACK pour développer des agents sur la base des concepts BDI, ou April++. Les plates-formes orientées sur la communication entre agents comprennent JFMAS, JATLite dont les concepteurs [JPC00] défendent maintenant le point de vue comme quoi il est erroné de baser le développement de systèmes d'agents sur leur état mental (motivations, croyances,...), car les agents ne peuvent lire la "pensée" des autres agents [Sin98].

Si la plupart de ces technologies sont présentées comme pouvant permettre le développement d'applications distribuées de toutes natures, y compris de gestion de réseaux, d'autres comme la plate-forme JIAC (Java Intelligent Agent Componentware) ont été développées spécifiquement pour concevoir des applications dans le domaine du service et du commerce électronique [WA98].
Toutes ces technologies hétérogènes reflètent bien le manque de maturité du domaine des agents intelligents, surtout quand on constate la disparité des concepts sous-jacents. Il est d'un autre côté bien évident que l'on peut faire du développement objet sans langage objet (bien que cela ne soit pas particulièrement facile), mais c'est avant tout la définition précise du paradigme objet qui le permet. Et c'est le manque d'une définition du paradigme "agent intelligent" admise par tous qui reste un handicap majeur dans ce domaine.

III.3.4 Services et Qualité de Services

Toujours sur la gestion de service, mais avec un objectif un peu plus ambitieux sur le concept d'agents intelligents, [Bus98] présente une architecture de services. Le travail est basé sur un système d'agents intelligents ayant des rôles et des capacités différents, et qui vont travailler en coopération pour offrir des services adaptés aux ressources et au profil d'utilisateurs mobiles. Ceux-ci en fonction des capacités offertes par les stations cellulaires pourront se voir offrir (proactivité) des services comme de la vidéo ou tout simplement des services de base comme un service d'appel d'urgence. L'un des enseignements que l'auteur retire de ces expérimentations est avant tout l'importance qu'il faut donner à l'adaptabilité du SMA en termes de connaissances et de capacités fonctionnelles.
Une approche originale de la gestion de qualité de service nous est présentée par [OL98] qui introduit un nouveau paradigme pour la gestion de réseaux en l'occurrence la *gestion avertie* (aware management). Ce paradigme de gestion est basé sur un SMA dont les agents sont développés sur le principe d'une architecture hybride dont la partie délibérative utilise des catégories mentales telles que croyances, désirs, intentions, objectifs, et engagements. Ces agents communiquent entre eux grâce au support du langage KQML. Dans cette nouvelle vision de la gestion de réseaux les utilisateurs sont vus comme des entités utilisant de applications et exprimant des exigences en QoS. Ces exigences sont capturées dans des profils d'utilisateurs, qui sont par la suite transférés et utilisés par les agents en charge du maintien de cette qualité de service. A chaque type de service est associé une dimension de QoS qui permet de sélectionner les paramètres utilisables qui seront compris par les agents de service. Les agents peuvent déléguer des objectifs de gestion, coopérer en échangeant le contenu de leur base de connaissance virtuelle, coordonner leur actions en utilisant une organisation hiérarchique. Ce paradigme a été validé par l'implémentation d'une étude de cas décrite dans [Oli98]. Malgré tout cette architecture utilise une hiérarchie de classes (héritage du monde objet) qui si elle est appropriée à la gestion de la QoS n'est pas extensible à l'ensemble des problèmes de gestion de réseaux.
Plus flexibles, les agents Hybrid présentés par [Som98] sont conçus pour intégrer des modules de compétences comme par exemple un module de négociation et travaillent en

coopération dans une organisation hiérarchique et autoritaire. L'autorité est composée de trois couches hiérarchiques (globale, régionale, locale), les agents des couches les plus hautes jouant le rôle de centre de gestion et déléguant aux agents subalternes les tâches de gestion. Ce SMA est considéré par [KSERS98] comme permettant une approche unifiée de la gestion de réseaux par agents intelligents. Pour les auteurs l'utilisation de la technologie CORBA et du langage KQML sont des garanties d'intéropérabilité entre les agents et les systèmes de gestion existants. Les modules de compétences utilisés par ces agents sont des scripts qui représentent des plans d'exécution de tâches réalisables par l'agent. Si ce SMA semble plus flexible que la majorité des SMA ayant été développés pour expérimenter le paradigme des agents intelligents dans le cadre de la gestion de réseaux, le découpage hiérarchique imposé, la limitation des tâches que peut exécuter un agent sont des problèmes potentiels, et des limites certaines au comportement adaptatif du SMA.

III.3.5 Conclusion

Les agents intelligents représente un champ de recherche multi-disciplinaire. La diversité des technologies développées, des domaines d'applications, des expérimentations, rend très difficile voire impossible l'unification des avantages qui apparaissent ça et là dans les travaux de recherche. Les langages de développement orientés agent comme Agent0, Lalo, AKL, April, n'offrent qu'un support limité au développement, car ils sont conçus avant tout pour traiter un problème particulier, comme l'expression de catégories mentales, ou la communication inter-agents. Les même limitations sont visibles avec les produits permettant le développement de systèmes multi-agents que nous avons évoqués. Les architectures agents qui sont fournies sont majoritairement très spécifiques et ne permettent pas de développer facilement des applications au comportement adaptatif.
Une des raisons que l'on pourrait trouver à cet état de fait est que les travaux de recherche portent généralement sur une propriété particulière que doivent posséder les agents, et que l'approche globale et comportementale du SMA est ignorée. En effet nous avons présenté au début de ce chapitre les agents comme étant des entités logicielles ayant certaines propriétés que nous avons décrites. Mais ces propriétés prises individuellement ne représentent pas les agents. Ce que nous recherchons pour résoudre les problèmes de gestion de réseaux est un paradigme permettant le développement d'applications distribuées, dont les composants distribués doivent pouvoir s'adapter, coopérer, montrer une certaine autonomie, et non pas avoir uniquement une propriété particulière.

III.4 Entre mythe et réalité, les avantages attendus

Depuis le début de ce chapitre nous avons essayé de faire la distinction, entre ce qui était du domaine de l'agent intelligent et ce qui ne l'était pas, et tenté de faire la distinction entre ce qui devait être fait pour développer le paradigme agent de ce qui ne devait pas l'être. Notre intention n'est pas de nous poser en censeur, mais de contribuer à la naissance d'un paradigme que nous croyons très important, pour non seulement la gestion

de réseaux, mais aussi pour l'informatique en général. Une vingtaine d'année auparavant un autre paradigme avait beaucoup fait parler de lui c'était le paradigme de l'intelligence artificielle. Ce paradigme faisait naître l'espoir que dans peu de temps, les machines allaient devenir intelligentes, et qu'il serait facile de faire des programmes pour résoudre les problèmes compliqués. L'histoire nous a montré que cela n'était pas si facile, et les revers de cette discipline ont été d'autant plus cuisants que l'on attendait des miracles de cette nouvelle voie.
L'esprit scientifique doit donc se garder des griseries commerciales, et il est important de faire la distinction entre ce qui est du domaine du possible de ce qui est du domaine de l'imaginaire pour ne pas reproduire le "fiasco" de l'intelligence artificielle. Depuis deux ans environ des spécialistes des agents intelligents se sont posés la question du devenir de cette discipline. En conclusion de son état de l'art sur les architectures d'agents intelligents, Jorg Muller souligne qu'il semble actuellement impossible d'espérer une convergence des travaux de recherche qui aboutirait à une architecture unique et acceptée par tous [Mul98a]. L'auteur se veut néanmoins optimiste et considère que tout comme pour les langages de programmation, il faut choisir la technologie la mieux adaptée au problème [Mul98].
Lors de la quatrième conférence sur les applications pratiques des agents intelligents et de la technologie multi-agents qui s'est tenue à Londres en avril 1999, les plus grands spécialistes se sont interrogés sur les barrières qui freinent le passage de ce paradigme, du monde de la recherche vers le monde industriel. Les causes identifiées sont entre autres:

- le mental: souvenir de l'échec de l'IA, défiance vis à vis d'un logiciel qui peut décider pour nous.
- La connaissance: nous ne maîtrisons pas encore les abstractions qui pourraient être utilisées par les agents
- la technologie: il n'existe pas encore de technologie agent universelle, et pour développer des agents, il faut d'abord construire l'environnement de développement
- la technique: la plupart des implémentations ne prennent pas en compte les problèmes techniques que sont pas exemple les problèmes de performance.

Le paradigme d'agent intelligent est-il un mythe? [Wei99] répond à cette question en rappelant que la conception de SMA répond à un besoin réel aussi bien au niveau technologique qu'au niveau applicatif. Il revient sur les problèmes théoriques et techniques de la distribution d'applications qui rendent nécessaire le développement de technologie dont les propriétés sont justement celles qui sont prêtées aux agents intelligents, comme l'autonomie et la coopération. Il soutient aussi que la vision d'une application à base d'agents présente des avantages supérieurs à la vision objet.
Pour [HS99] les motivations pour le développement d'une technologie agents sont de plusieurs ordres. Les auteurs admettent que toute application qui peut être développée par une technologie de distribution peut être déplacée sur un système unique et optimisée sur ce système. Néanmoins il est d'après eux plus facile de comprendre et de concevoir une application en utilisant des composants distribués, surtout quand le problème est lui-même distribué. Ils remarquent aussi que l'approche centralisée est quelquefois

impossible à utiliser, comme par exemple lorsque l'application fait intervenir plusieurs organisations voulant chacune conserver la confidentialité de ses données. Avec les réseaux l'information est nécessairement distribuée. Elle peut-être géographiquement distribuée, elle peut être distribuée dans plusieurs composants d'un même système, elle peut être distribuée dans des domaines importants en taille. La recherche et le traitement de l'information deviennent plus complexes du fait même de cette distribution. De plus la distribution est souvent synonyme d'hétérogénéité, et de modification dynamique de topologie. Les principales techniques pour traiter les problèmes de taille et de complexité des systèmes d'information sont:

- la modularité,
- la distribution,
- l'abstraction,
- l'intelligence.

[HS99] constatent que les agents intelligents combinent ces quatre concepts. [Boo94] rajoute l'organisation comme moyen supplémentaire de maîtriser la complexité.
Les articles cités reflètent une crise dans le domaine de la recherche sur les agents intelligents, ou plutôt une prise de conscience des risques et des enjeux. Un long et récent article de Nicholas R. Jennings expose d'une manière différente ces mêmes interrogations [Jen00].
La conclusion que nous pouvons faire est donc que si la technologie agent-intelligent n'existe pas encore, le paradigme agent-intelligent est, lui ,bien en train de s'affirmer comme la seule solution complète aux problèmes du développement d'applications distribuées.
Comment pouvons nous alors développer cette technologie? Notre point de vue est que pour créer une nouvelle technologie il faut se détacher de celles existantes. Il ne s'agit pas de réinventer la roue mais plutôt d'oublier la roue et d'inventer le moteur.

IV DIANA - Principes et Architecture

Nous avons constaté dans les chapitres précédents que le paradigme *agent-intelligent* répondait parfaitement aux problèmes identifiés des systèmes de gestion. Nous avons vu aussi qu'actuellement les technologies développées autour de ce paradigme n'implémentent que très partiellement les propriétés qui y sont associées. Comment valider alors une solution de gestion à base d'agents intelligents? Faut-il utiliser un des nombreux environnements de développement disponibles, en négligeant le fait qu'ils implémentent uniquement une ou deux propriétés, et se restreindre à celles-ci? Ou faut-il se lancer dans la création d'une nouvelle plate-forme de développement d'agents intelligents?
Dans un premier temps nous avons pensé utiliser des outils existants, et pour les choisir, il nous fallait tout d'abord identifier les propriétés indispensables que devait posséder notre SMA, pour couvrir l'ensemble des problèmes des systèmes de gestion de réseaux. Puis il fallait sélectionner les outils qui permettaient l'implémentation de ces concepts. Et enfin développer quelques études de cas propres à démontrer les propriétés comportementales de notre solution. Mais s'il nous a été relativement facile de retenir un ensemble de propriétés, la sélection des outils ou plates-formes de développement s'est vite relevée impossible. Les raisons de cette impossibilité ont été exposées précédemment, et sont dues à l'absence d'une technologie incluant plus d'une ou deux propriétés "agent". L'expérimentation d'un outil de développement d'agent mobile, Voyager en l'occurrence, nous a permis de comprendre très rapidement les limitations qu'imposait l'utilisation d'agents mobiles, limitations que nous avons déjà évoquées dans le chapitre III3.2..

Nous avons alors considéré l'utilisation de systèmes permettant de développer "l'intelligence" des agents au sens IA. Nous avons pour cela utilisé Jess (Java Expert

System Shell) développé par Ernest Friedman-Hill du Sandia National Laboratories, après avoir rapidement écarté ‘Ilog rules for Java’ encore instable dans sa version beta. Le problème que nous avons alors rencontré est celui de la rigidité de l'algorithme RETE utilisé par Jess et Ilog rules. En effet, cet algorithme permet d'optimiser la vitesse du chaînage avant (recherche des conséquences), et nécessite pour cela une précompilation de l'ensemble des règles avant de pouvoir utiliser le moteur d'inférence. Il est donc impossible de charger une nouvelle règle, ou de prendre en compte un nouveau fait extérieur, au cours du déroulement de l'algorithme. Le traitement de tâches en parallèle est donc quasi impossible. Si ce problème ne vient pas directement en contradiction avec les propriétés agent, il n'en reste pas moins un obstacle au comportement adaptatif que nous attendons des applications agent.
Le concept de coopération a été le sujet de la production de quelques technologies agent comme JATLite (Java Agent Template, Lite) ou JAFMAS (Java-based Agent Framework for Multi-Agent Systems). Si nous pouvons reprocher au premier l'approche centralisée de l'échange de messages, ce n'est pas le cas de JAFMAS qui permet des communications directes et multi-destinataires, ainsi qu'un mécanisme de synchronisation entre agents. JAFMAS aurait donc pu être un point de départ pour le développement de l'architecture de notre SMA, mais deux facteurs nous ont fait renoncer à son utilisation. Le premier est l'existence de nombreux bugs qui nous auraient rendus dépendants de l'équipe de développement de JAFMAS, le deuxième plus critique est que l'architecture JAFMAS est construite autour d'un modèle de communication rigide et incompatible avec les propriétés de délégation et de transfert de compétences que nous voulions expérimenter.
Devant le manque de maturité et la limitation des produits disponibles, nous avons alors choisi de créer une nouvelle architecture d'agent intelligent qui pourrait réunir l'ensemble des propriétés que nous avions retenues. Ce projet a été nommé **DIANA** pour Distributed Intelligent Agents for Network Administration.

Ce chapitre présente tout d'abord les concepts et propriétés que nous avons choisis d'utiliser pour expérimenter les agents intelligents comme solution aux problèmes de la gestion de réseaux. Puis nous détaillerons l'architecture qui a été développée pour implémenter ces concepts.

IV.1 Concepts et propriétés recherchés

Les agents intelligents sont présentés comme des entités logicielles possédant certaines propriétés, et utilisant des concepts abstraits. Si des éléments conceptuels ont une correspondance directe avec l'architecture DIANA que nous avons définie, comme les *croyances* que nous définirons plus loin, d'autres ont été utilisés sans faire l'objet d'une implémentation particulière comme par exemple le rôle.
Dans les sections suivantes, nous revenons sur ces propriétés en abordant à la fois les concepts qui y sont attachés, et le côté technique nécessaire à leur développement.

IV.1.1 Autonomie et flexibilité

Beaucoup de systèmes multi-agents mettent en relief l'importance de l'autonomie de leurs agents. L'autonomie étant une propriété permettant à l'agent d'agir sans intervention directe d'un être humain pour atteindre ses objectifs. Cette propriété est quelquefois invoquée pour légitimer le développement de capacités de négociation. Chaque agent ayant alors la possibilité de refuser une tâche. Aussi intéressante que soit cette approche, elle nous paraît opposée à la notion de contrôle que chaque gestionnaire de réseaux peut attendre du système qu'il utilise.
D'un autre côté, l'agent intelligent a le contrôle de son activité et doit gérer ses ressources. Il est donc normal qu'il puisse refuser d'endosser un rôle, ou d'accomplir une tâche si ses ressources ne lui permettent pas de le faire.
Enfin, une facette importante du concept d'autonomie que nous devons retenir pour tout SMA et donc pour nos agents, est la possibilité de choisir comment seront accomplies les opérations qui lui sont demandées.
Cette approche de l'autonomie est connue dans le monde des agents sous la dénomination "*orienté objectifs*". Un agent *orienté objectifs* reçoit des demandes pour obtenir des résultats (les objectifs), et sélectionne les actions à exécuter pour atteindre ses objectifs. Grâce à cela, l'agent offre une partie de la flexibilité recherchée dans le développement de solutions de gestion de réseaux. Par exemple lorsqu'un agent doit alerter le gestionnaire de réseaux d'un problème qu'il ne peut résoudre lui-même, il va rechercher parmi les moyens de communication qu'il connaît ceux qui permettront de satisfaire l'objectif "gestionnaire prévenu". Si lors de la conception de l'application, il n'existait que le courrier électronique (email) et que le gestionnaire de réseau possède maintenant un téléphone portable, il suffira de prévenir l'agent qu'il peut utiliser une transmission par téléphone pour prévenir le gestionnaire pour que cette solution soit utilisée.
Il n'y a donc pas besoin de modifier l'application de gestion pour que ce nouveau paramètre soit pris en compte.

IV.1.2 Généricité

A l'opposé des développements agent qui ont été produits dans la littérature sur les agents intelligents et la gestion de réseaux, et qui se limitent à un seul type d'applications, il nous est apparu important d'offrir une architecture ouverte et générique. Dans un article sur les pièges du développement orienté agent [WJ98], les auteurs conseillent de résister à la tentation de considérer toute nouvelle architecture comme générique. En effet, selon eux, beaucoup d'architectures agent ont été conçues pour résoudre un problème, et améliorées par la suite pour les rendre aptes à en résoudre d'autres. Nous sommes sur ce point d'accord avec eux pour considérer cette démarche comme vouée à l'échec. Dans notre cas nous étudions au préalable les mécanismes à utiliser pour rendre une architecture la plus générique possible, en tenant compte non pas d'un problème particulier mais d'un domaine, celui de la gestion de réseaux. La généricité qui nous intéresse se limite donc à ce domaine.

Comme nous l'avons vu précédemment, l'évolution de la conception des applications va vers l'indépendance des composants. Si de nos jours il est classique de trouver ces composants sous forme de classes d'objets, cette évolution est aussi visible sous d'autres formes de composants, comme les fameux haricots (beans) proposés par Sun sous forme d'agents mobiles, ou encore sous forme de capsules qui sont des micro-programmes servant à programmer des commutateurs dans les réseaux actifs. Cette approche par composants peut être considérée comme une évolution logique de la programmation objet, mais elle n'est pas directement compatible avec l'approche agent.
En effet, ainsi qu'il est décrit dans les chapitres précédents, l'approche agent utilise une notion de comportement qui n'apparaît pas dans l'approche objet. Les objets et leurs classes sont connus au travers de leurs méthodes et propriétés, et ne donnent aucune indication automatiquement exploitable sur leurs comportements.
Dans le monde objet, pour qu'une application utilise des composants (objets), elle doit posséder la description des interfaces de ces composants (stubs). Cette description est utilisée pour créer le code exécutable de l'application au moment de la compilation, c'est une méthode d'invocation purement statique. La technologie CORBA offre un deuxième mécanisme plus souple, l'interface d'invocation dynamique, qui permet de construire dynamiquement une requête pour accéder à un objet CORBA dont on ne connaît pas l'interface à la compilation.

La technologie des «beans» contourne aussi cette contrainte statique en imposant un principe de nomage et en offrant une technique dite d'introspection permettant de découvrir les méthodes et les propriétés accessibles de chaque bean. Malgré tout, le comportement final de l'application est figé au moment de son design, et aucun support n'est fourni pour comprendre le comportement des composants pendant l'exécution (runtime).

Afin de conserver les avantages de l'approche par composants et pour servir l'architecture DIANA, nous avons donc introduit une notion utilisée en programmation logique (déclarative) qui consiste à exposer les causes et les conséquences du déclenchement d'une règle ou d'une action dans notre cas. Nous pouvons alors définir des composants de compétence, que nous appellerons modules de compétences, en utilisant une interface standard permettant de décrire un ensemble d'opérations dont les pré et post conditions seront comprises par l'agent.

L'adoption d'un système à base de composants dont le contenu serait découvert par l'agent au runtime a un double objectif:

- ▹ d'un côté mettre en place les conditions nécessaires au comportement délibératif de l'agent, en lui donnant la possibilité d'utiliser des moteurs d'inférence utilisés en IA,
- ▹ de l'autre utiliser la puissance de l'approche par composants déjà évoquée.

Nous pouvons alors concevoir le cœur de l'agent, ou plutôt le cerveau autour d'un système « intelligent » de gestion de composants. Le cerveau est alors en charge de

rechercher les composants nécessaires à l'exécution des activités associées aux rôles qui sont alloués à l'agent.
Le développement de ce nouveau concept où déclaratif et procédural sont étroitement liés nous a permis de proposer une architecture d'agent générique, c'est-à-dire utilisable pour la construction de toute application distribuée.
Dans des études sur les modèles d'interaction de composants, [Esk99] propose d'autres concepts comme celui du «bus de composants» sur lequel les différents composants vont enregistrer les informations qu'ils offrent ou demandent. Par rapport à notre approche nous pouvons noter que si ces modèles permettent aussi une utilisation dynamique des composants, le traitement des comportements n'est pas pris en compte.

IV.1.3 Rôles et compétences

En traitant le problème de la généricité de l'agent, nous avons mis en relief le côté technique de l'implémentation en invoquant les composants logiciels. Au niveau de l'approche agent, ces composants représentent une certaine compétence, c'est-à-dire la capacité de traiter certaines informations ou d'effectuer certaines opérations.

La comparaison avec l'organisation d'entreprise nous permet de comprendre la relation qu'il y a entre le rôle et les compétences. La figure 12 nous montre cette relation où le rôle est traduit par des compétences, elles-mêmes matérialisées par les opérations qui peuvent être effectuées. Ces *opérations* que peut effectuer l'agent sont aussi appelées *capacités*.

Ces composants appelés ***modules de compétences*** sont les briques de toute application DIANA, et du fait même de l'architecture générique un même SMA peut supporter plusieurs applications différentes.

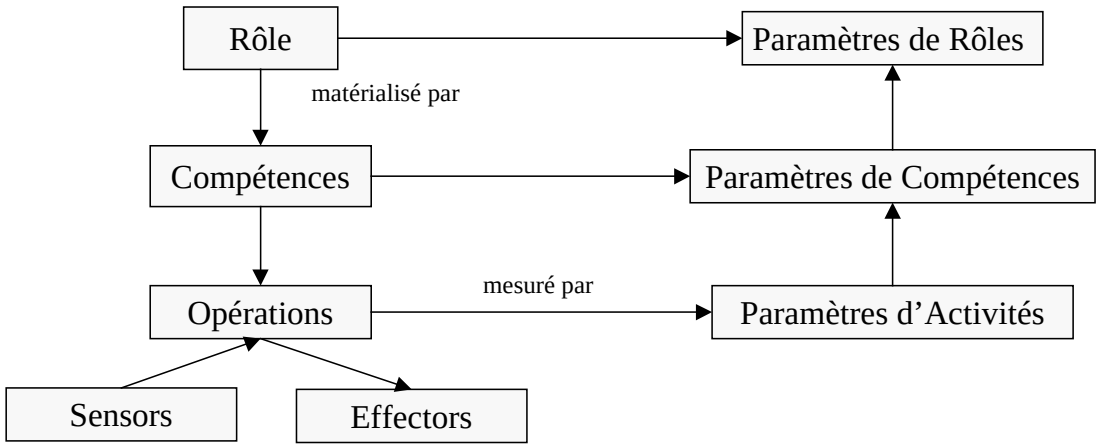


figure 12 Relation rôles - compétences - opérations

Si nous étudierons plus en détail, lors des études de cas, la conception d'une application à base d'agent, nous pouvons déjà comprendre que la conception d'une application se fera en premier lieu par l'identification des différents rôles nécessaires.
L'application peut être comparée à une entreprise où des rôles sont affectés aux différents postes de l'entreprise indépendamment des personnes occupant ces postes. Le déploiement de l'application correspond alors à l'affectation des rôles et donc des compétences composant l'application aux agents du SMA.

IV.1.4 Objectifs et croyances

Lorsqu'un agent se voit assigner un rôle, il utilise ses compétences pour jouer ce rôle. Ces compétences se traduisent par des capacités à effectuer certaines opérations qui correspondent à des *tâches* ou à des *activités*. Nous pouvons faire la distinction entre ces deux termes en considérant qu'une **tâche** correspond à l'exécution d'une opération ponctuelle, et qu'une **activité** correspond à l'exécution permanente d'une opération.
Cette distinction se retrouve couramment dans la littérature agent sous les termes *motivation* et *objectif*. La motivation est dans ce cas un état à maintenir, tandis que l'objectif est un état à atteindre.
Dans la suite de notre exposé nous ne ferons néanmoins pas de différence entre tâche et activité, considérant que cette différence doit être prise en compte non pas génériquement par l'agent mais au niveau de chaque compétence.
Dans ces deux cas, motivation ou objectif, l'état à maintenir ou à atteindre est exprimé sous forme de *croyances* (beliefs). Le terme croyance exprime qu'un agent n'a qu'une vue partielle et temporelle de la réalité de son environnement. Une information qui peut être vraie à un instant t peut très bien être erronée à un instant $t+1$. La **croyance** est une information connue et utilisée par l'agent de son état ou de celui de son environnement.
Nous étendrons cette définition de la croyance, en considérant qu'une croyance ou belief est une information manipulable par l'agent.
Lorsque deux agents s'échangent des informations, ces informations sont considérées avant tout comme des croyances.
C'est grâce à la sémantique du langage utilisé que l'agent peut faire la différence entre les informations sur des états, et les informations sur les opérations à effectuer.

IV.1.5 Délégation

Un autre principe important que nous avons utilisé est le principe de délégation qui permet comme un grand nombre de recherches l'ont démontré, de résoudre les problèmes d'échelle rencontrés par les NMS [YGY91]. Cette section a pour but de décrire notre point de vue sur ce paradigme et les choix que nous avons faits pour que l'architecture DIANA supporte la délégation.

L'approche traditionnelle de la délégation utilise une relation *un à un* entre deux entités le délégateur et le délégué. C'est en effet l'approche logique et simple qui est utilisée depuis longtemps pour le développement des clients/serveurs. L'utilisation d'une

approche *un à plusieurs* est comme nous le rappelle [MZH99] déconseillée et même interdite par la plupart des auteurs d'articles sur la gestion de réseaux. Ce n'est pas le cas dans le domaine de l'intelligence artificielle distribuée où l'approche un à plusieurs est un sujet important de recherche sur la coopération entre agents. Un exemple connu est celui de la planification globale partielle (Partial Global Planning) qui permet la décomposition d'une tâche globale en sous-tâches. Dans ce cas un agent a la connaissance globale de l'ensemble des tâches, alors que les agents délégués n'ont qu'une vision partielle de la tâche globale. Cette approche est comme nous le verrons par la suite importante pour réduire voire supprimer les problèmes de facteurs d'échelle que connaît la gestion de réseaux.

IV.1.5.1 Délégation un à un

Lorsque nous avons étudié l'évolution du paradigme objet et des technologies, nous avons vu comment le concept de la distribution permet d'invoquer du code distant et même de le transférer d'un système à un autre.
Nous devons maintenant examiner si ces différents types de délégation sont vraiment utiles dans le cadre de la gestion de réseaux. Pour cela nous devons fixer un moyen de les comparer.
Nous pouvons alors nous inspirer du système de comparaison proposé par [Pic98] qui distingue les éléments suivants:

- code à exécuter
- paramètres d'exécution
- sites d'exécution
- agents d'exécution

La délégation est ensuite étudiée en fonction de la position des éléments avant et après l'exécution du code.

Paradigme	**Avant**		**Après**	
	Site 1	**Site 2**	**Site 1**	**Site 2**
RPC	*A* Paramètres	*B* Code Ressources	*A*	*B* Paramètres Code Ressources
Evaluation à distance	*A* Paramètres Code	*B* Ressources	*A*	*B* Paramètres Code Ressources
Serveur élastique	*A* Paramètres Code	*B* Ressources	*A*	*B* Paramètres Code Ressources
Code à la demande	*A* Paramètres	*B*	*A* Paramètres	*B*

	Ressources	Code	Code Ressources	
Agent Mobile	*A* Paramètres Code	Ressources		*A* Paramètres Code Ressources

Le tableau comparatif que propose [Pic98] et que nous présentons ci-dessus ne donne qu'une indication sur le déplacement du code et des paramètres d'exécution. Il ne donne pas, par exemple, d'indications sur l'agent en charge du contrôle de l'exécution du code, ce qui fait que l'évaluation à distance et le serveur élastique ont l'air identiques, alors que toute la différence est dans le contrôle de l'exécution qui est très poussé avec le serveur élastique.

Il est important de souligner les avantages et inconvénients de chaque approche en fonction de l'utilisation qui est faite des communications. Par exemple l'approche client/serveur RPC qui est avantageuse lorsqu'une seule commande est invoquée (micro-tâche), devient un inconvénient si l'application demande de nombreux échanges entre les systèmes. Cet argument est utilisé par les défenseurs des agents mobiles qui préfèrent envoyer le code nécessaire à l'exécution d'un ensemble d'opérations (macro-tâche), pour diminuer les échanges entre le client et le serveur.
Il est donc difficile voire impossible d'avoir une vue objective des avantages et inconvénients de chaque paradigme. Dans sa thèse [Pic98] développe une évaluation quantitative de ces approches pour démontrer le gain apporté par l'approche *agent mobile*. Mais si ce gain ressort dans certains cas, il conclut néanmoins que le choix du bon paradigme dépend de beaucoup de facteurs, et qu'il est impossible à déterminer a priori.

En étudiant les analyses de G. P. Picco, nous avons pu remarquer que l'approche *agent mobile* pouvait être avantageusement remplacée par une approche d'agents statiques supportant à la fois le paradigme du code à la demande, de l'évaluation à distance, et du serveur élastique. En effet dans le cas où un agent doit exécuter une nouvelle tâche sur un équipement distant, l'application codée sous forme d'agent mobile doit non seulement contenir le code correspondant à la fonctionnalité de cette tâche, mais aussi tout le mécanisme de contrôle de l'exécution de la tâche. Si cette tâche doit être réalisée plusieurs fois, il faut alors déplacer autant de fois l'agent mobile, ce qui logiquement augmente la consommation de la bande passante. Dans le cas d'un agent statique utilisant le paradigme du code à la demande, celui-ci, une fois reçus la demande d'exécution de la tâche ainsi que les paramètres correspondants, peut récupérer le code et le conserver localement. Et dans ce cas, le transfert du code à travers le réseau ne s'effectue qu'une seule fois.

S'il est donc assez facile de montrer dans la plupart des cas l'avantage des agents statiques DIANA sur les agents mobiles, il est encore plus facile de démontrer l'avantage en terme de performance. Les performances ne sont d'ailleurs que très rarement traitées dans le contexte des agents mobiles. Le temps de sérialisation, de désérialisation, de transfert et de chargement est un effet un handicap pour cette forme d'application.

Pour ces différentes raisons nous avons choisi de ne pas prendre en compte la mobilité dans la conception de nos agents DIANA, mais d'offrir en contrepartie de supporter les paradigmes suivants:

- ▹ code à la demande,
- ▹ évaluation à distance,
- ▹ serveur élastique.

En ce qui concerne l'appel à distance de procédure (RPC), il est évident que ce paradigme est en contradiction totale avec l'approche agent intelligent. Si nous faisons une analogie avec deux être humains, cela correspondrait à ce que l'un d'eux puisse actionner à distance le bras de l'autre, en activant directement les nerfs moteurs correspondants.
Par contre l'approche agent rend concevable que l'une des personnes demande à l'autre de bouger son bras, respectant dans ce sens l'autonomie d'action de cette autre personne. L'architecture des agents DIANA prend donc en compte un modèle équivalent, et un agent peut demander à un autre d'effectuer une opération, sans pour cela lui préciser obligatoirement l'objectif de cette opération.

IV.1.5.2 Délégation de un à plusieurs

Les études des différentes formes de délégation que nous venons de voir ont été menées dans le but d'améliorer les performances des systèmes de gestion construits autour du concept manager-agent, c'est-à-dire de systèmes hiérarchiques à deux niveaux.

Dans un tel système, nous pouvons concevoir les relations de délégation en fonction de leur sens et direction, [MZ97] nous propose une typologie de ces relations (figure 13).

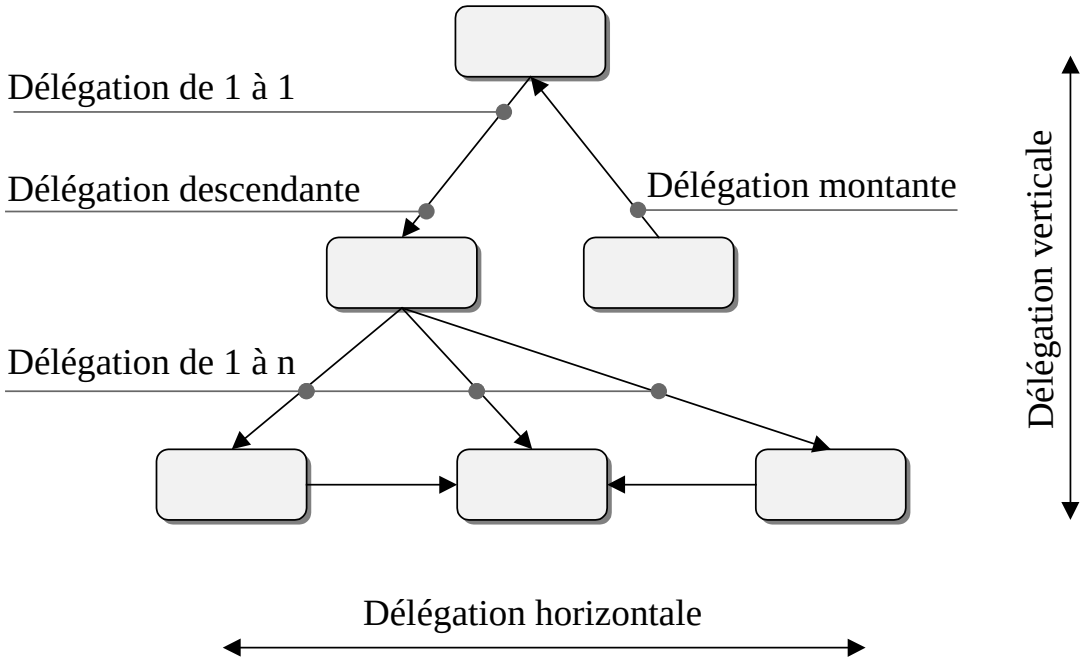


figure 13 Les types de délégation

Cette typologie de délégations nous montre les différentes relations que nous pouvons trouver dans une organisation. Dans le sens horizontal, la délégation exprime une coopération entre les entités sans tenir compte d'une quelconque autorité. Au contraire le sens vertical descendant exprime l'autorité d'un agent sur un autre, alors que le sens montant est utilisé quand un agent est défaillant et ne peut continuer son activité.

Si nous avons jusqu'à présent considéré une relation *un à un*, les relations un à plusieurs (1 à n) sont toutes aussi importantes et applicables à la gestion de réseaux. Un exemple que nous utiliserons dans une étude de cas est la délégation de l'activité de monitoring non pas vers un agent mais vers un ensemble d'agents, ensemble que nous appellerons *domaine*. Le domaine représente une abstraction d'un ensemble d'éléments (agents ou équipements) qui vont pouvoir être manipulés comme un tout.
Dans le cas du monitoring des équipements, les fréquences de polling sont généralement dépendantes du type d'équipement, et indépendantes de l'équipement lui-même. Il suffit donc de rajouter ou de supprimer un équipement dans le domaine pour que celui-ci soit monitoré avec les mêmes contraintes que les autres. De la même manière un agent affecté à un domaine de gestion va se voir affecter les mêmes tâches que les autres agents appartenant au domaine.

IV.1.5.3 Opérations de délégation

Nous venons de voir le côté technique, et donc l'implémentation de la délégation qui correspond au mécanisme de transfert du code et des paramètres de contrôle, ainsi que la relation qui existe entre la délégation et l'organisation des agents. Ici nous utilisons une vue plus abstraite de cette délégation, conforme à l'approche agent.

Pour notre architecture nous avons retenu deux types de délégation:

1) délégation de tâche
2) délégation de rôle

Dans ces deux cas les agents travaillent sous la responsabilité d'un autre agent qui est au plus haut de la hiérarchie l'administrateur réseau. Cette dépendance de responsabilité est traduite par l'identification de l'auteur d'une requête, requête qui peut être soumise à un contrôle de sécurité (autorisation).

La **délégation de tâche** correspond à la demande qui est faite par un agent (le délégateur) à un autre agent (le délégué) pour exécuter une tâche en son nom. Par défaut l'exécution d'une tâche se traduit par la création d'au moins une information indiquant le résultat de la tâche, information qui sera automatiquement transmise au délégateur. Dans certains cas le requérant peut vouloir ne pas être notifié des résultats, ou n'en recevoir qu'une partie. Cela correspond à un filtrage des informations et se traduit par une demande explicite de désabonnement des informations résultantes auprès du délégué.

[MZH98] nous propose de distinguer la délégation par domaine de la délégation de tâche. Pour ces auteurs, le domaine est une entité géographique dont la gestion est déléguée à un manager local. La délégation de tâche offre pour eux une meilleure finesse de gestion. Ils distinguent pour cela les micro-tâches, quelquefois appelées micro-management, et qui correspondent à des opérations sur les éléments d'une MIB d'un équipement, des macro-tâches qui correspondent à l'exécution d'un ensemble d'opérations par l'agent sans intervention du manager.
Notre définition est assez différente, car nous associons le *domaine* à un ensemble quelconque sur lequel porte une opération de gestion. Il peut donc s'agir aussi bien d'un ensemble d'équipements sur lequel va porter l'activité de gestion, que d'un ensemble d'agents qui va effectuer une tâche de gestion.
Comme nous le verrons plus tard, il est courant que la délégation de tâches se rapporte à un domaine de gestion. Par exemple la tâche qui consiste à monitorer le nombre d'erreurs sur les ports des concentrateurs ou des commutateurs s'applique à des domaines qui sont définis par le gestionnaire de réseau et connus de l'agent. Ainsi lorsque l'agent délégué reçoit la demande de monitoring correspondante, il met en place le monitoring pour l'ensemble des équipements du domaine de monitoring.

La **délégation de rôle** correspond par contre au transfert d'un rôle. C'est-à-dire que l'agent délégateur demande à un autre agent de prendre en charge le contrôle de l'ensemble des activités correspondant à ce rôle. L'agent délégué doit donc posséder les compétences nécessaires à la réalisation des activités.
La délégation de rôle implique la délégation du contrôle de l'exécution des tâches distribuées. Le cas typique est celui de la gestion de grands réseaux où il est nécessaire de traiter localement les informations de monitoring afin de ne pas saturer les liaisons inter-réseaux. Cette délégation ne fait donc pas appel uniquement à un transfert des capacités opérationnelles d'un agent à un autre, mais demande aussi l'interprétation des résultats nécessaire au contrôle de l'exécution des tâches.

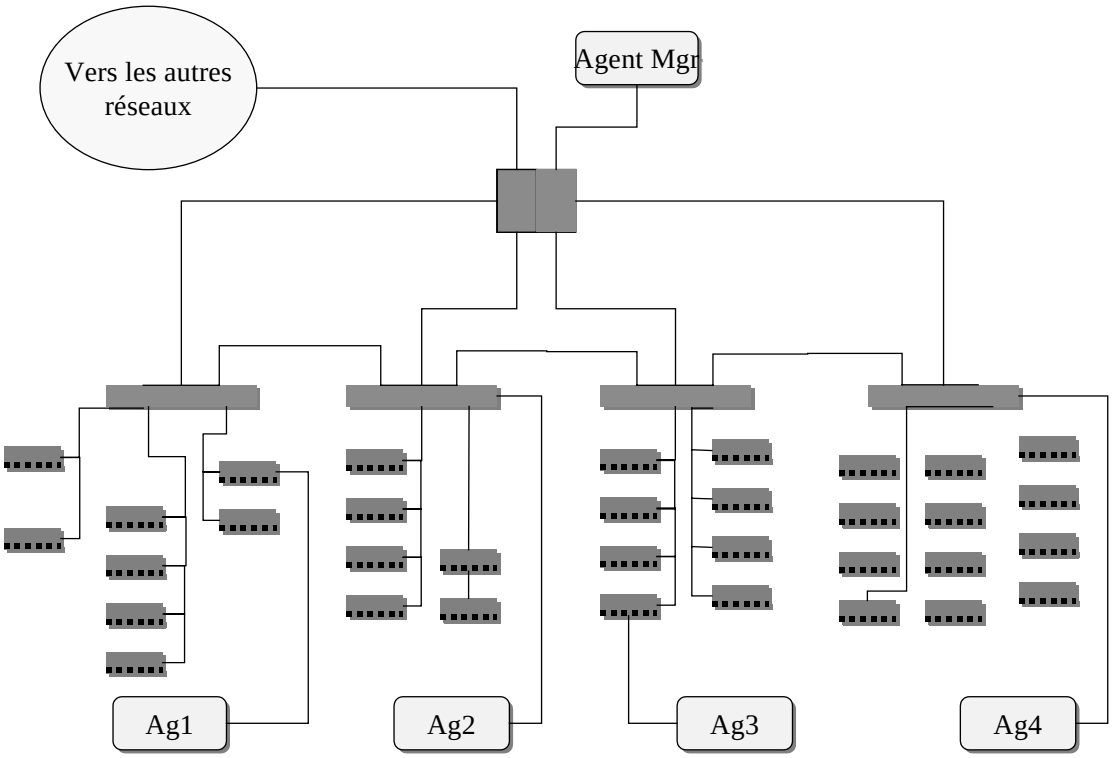


figure 14 Délégation du rôle de moniteur dans un réseau

Pour comprendre un peu mieux la différence entre délégation de rôle et délégation de tâche, prenons l'exemple d'un agent manager gérant un routeur et quatre sous-réseaux. Considérons que cet agent manager a à son service un agent par sous-réseau, donc quatre agents Ag1,Ag2,Ag3,Ag4 (voir figure 14).

L'agent manager joue à la fois le rôle de superviseur de réseaux et le rôle de moniteur. Le rôle de superviseur consiste dans notre cas à détecter toute anomalie dans le fonctionnement des équipements. Le rôle de moniteur consiste à interroger les équipements et créer des informations sur l'état de ceux-ci.
Une vue simplifiée de la relation entre les rôles et les opérations peut être:

Rôle superviseur ->
compétence superviseur ->
opération "évaluation taux d'erreurs/ équipement"
opération "évaluation taux d'erreurs moyen"
Rôle moniteur ->
compétence moniteur ->
opération "évaluation des interfaces"

Naturellement, pour que la surveillance soit précise, il faut que les informations soient mises à jour fréquemment. Pour éviter de saturer le routeur avec ces informations, nous pouvons envisager que l'agent manager délègue à chaque agent le soin d'évaluer le taux d'erreurs par équipement correspondant à son propre sous-réseau.
Considérons maintenant:

1) que chaque agent (Ag1, Ag2, Ag3, Ag4), possède une croyance (belief) sur le domaine des équipements qu'il a en charge:
   NeDomain = {e1,...,en}
2) que l'agent manager possède une croyance sur le domaine des agents qui sont sous son autorité:
   AgFaultMgtDomain = (Ag1, Ag2, Ag3, Ag4)
3) que le nom de la compétence associée au rôle de superviseur soit:
   SkFaultMgt
4) que le nom de la compétence associée au rôle de moniteur soit:
   SkMonitor

Une demande de *délégation de tâche* créée par l'agent manager pourra ressembler à:

SendTo AgFaultMgtDomain SkFaultMgt errorRate :on NeDomain :freq high

dans laquelle:

1) *SendTo* est l'expression permettant d'indiquer que l'information doit être transmise aux agents appartenant au domaine *AgFaultMgtDomain*.
2) *SkFaultMg errorRate* est la référence à la tâche déléguée
3) *:on NeDomain :freq high* représente les informations complémentaires pour l'exécution de la tâche

Chaque agent du domaine *AgFaultMgtDomain* va être le délégué de l'agent Manager pour l'exécution de l'opération *errorRate*. Cette délégation de tâche entraîne une délégation implicite du monitoring des équipements, car chacun des agents va devoir produire les informations de monitoring nécessaires à la création d'une information sur le taux d'erreurs des équipements appartenant au NeDomain de chacun de ces agents.
Pour effectuer cette tâche chaque agent doit mettre en place le monitoring adéquat portant sur chacun des équipements du domaine NeDomain, en produisant par exemple

SkMonitor :on NeDomain :MIB_Desc ifInOctects ifOutOctects ifInErrors ifOutErrors :freq high

Les agents ont donc une délégation de tâche pour la création de l'information sur le taux d'erreurs et une *délégation de rôle* implicite pour la mise en place et la gestion du monitoring des équipements impliqués.
La différence entre ces deux types de délégation est que dans la délégation de tâche, l'agent manager (délégateur) exerce un contrôle direct sur l'activité liée à la tâche de calcul du taux d'erreur auprès de l'agent délégué, mais ne peut modifier les tâches de monitoring qui en découlent. En effet, il est possible que l'agent ait une activité de monitoring pour un autre agent manager et n'ait pas à relancer le monitoring sur certains équipements. C'est donc l'agent délégué qui gère l'organisation de ses tâches.

Nous pouvons comprendre de cet exemple que le rôle ne correspond pas nécessairement à une activité effective, c'est-à-dire qu'un agent peut avoir un rôle mais ne pas l'exercer. Dans notre exemple en effet, l'agent manager n'a pas exercé son rôle de moniteur mais l'a laissé jouer à l'agent délégué.

Cet exemple soulève aussi un problème, celui de la *concordance* des tâches, que nous décrirons dans le chapitre suivant.
En effet si ce problème est induit par la propriété de délégation, il peut être ignoré dans une structure hiérarchique stricte où un seul agent peut déléguer un seul type de tâche. Dans ce cas en effet, l'agent délégateur a le contrôle des délégations c'est-à-dire ici la connaissance des tâches en cours chez les agents délégués, et peut alors décider si une tâche peut être déléguée plusieurs fois au même agent sans affecter la logique de l'application.

IV.1.5.4 Conclusion

Pour supporter les opérations de délégation combinant les avantages du serveur flexible et du code à la demande, l'architecture DIANA doit donc comprendre un mécanisme de gestion des tâches, et un mécanisme de chargement des modules de compétences. La gestion de la délégation telle que nous l'avons décrite nécessite pour chaque agent la connaissance de ses voisins, c'est-à-dire la possibilité de les identifier, de les contacter et d'utiliser un protocole d'échange de messages et un langage de communication commun. Chaque agent doit aussi savoir qui a demandé quoi, ceci afin de communiquer les croyances produites aux agents concernés ou d'arrêter les tâches si un agent demandeur n'existe plus.

IV.1.6 Coopération et concordance des tâches

Coopération veut dire que plusieurs agents vont travailler ensemble pour atteindre un objectif. Comme exemple de coopération, nous avons le cas de la configuration d'un ensemble de commutateurs ATM pour la création d'un PVC (permanent virtual circuit) entre deux stations, où sur chaque commutateur, un agent intelligent va s'occuper de la configuration en fonction des informations sur la QoS recherchée.

Il existe plusieurs formes de coopération suivant que l'on se place en observateur externe au système et que l'on analyse a posteriori un comportement global comme coopératif [Fer95], ou que l'on considère le système comme volontairement coopératif.
Nous nous plaçons ici dans ce dernier cas, en nous concentrant sur un système de coopération hiérarchisée. Plus précisément nous avons choisi le cas où la coopération se fait sous le contrôle d'un agent responsable (manager) qui a à sa disposition un groupe d'agents. Les agents du groupe vont travailler ensemble pour accomplir une tâche, c'est-à-dire qu'ils vont coopérer, mais ils n'auront pas la possibilité de négocier entre eux.

Le choix que nous avons fait ici est justifié premièrement par la nécessité de limiter l'autonomie des agents dans un contexte de gestion de réseaux, et deuxièmement par la possibilité qui doit être offerte au gestionnaire de réseaux d'organiser la répartition du rôle de ses agents de gestion. Des agents qui négocieraient l'affectation de leurs rôles et de leurs tâches seraient potentiellement en conflit avec le gestionnaire de réseaux.
L'agent manager dans une telle configuration va avoir la responsabilité de surveiller le travail coopératif, et de résoudre les problèmes ou conflits qui peuvent se produire.
Nous avons vu que la coopération était une délégation horizontale, et celle-ci pose des problèmes spécifiques du fait que les agents délégateurs n'ont pas a priori la connaissance de l'activité du délégué.

Par exemple la coopération dans un système multi-agents soulève le cas où un agent peut être le délégué de plusieurs agents différents pour l'exécution d'une tâche identique. Si la tâche est atomique, doit-il exécuter celle-ci autant de fois qu'il en a la délégation? S'il s'agit d'une activité, doit-il en lancer plusieurs en parallèle? A l'inverse lorsqu'il reçoit la demande de stopper cette tâche, doit-il arrêter la tâche bien que d'autres agents lui aient demandé de l'exécuter? Que doit-il faire s'il reçoit une délégation pour désactiver une interface, et une autre délégation pour l'activer?
Ces problèmes sont classés par [Bou92] sous forme d'indices de coopération:

1. la coordination d'actions qui concerne l'ajustement de la direction des actions des agents dans le temps et l'espace
2. le degré de parallélisation qui est fonction de la répartition des tâches et de leur résolution concurrente
3. le partage des ressources, qui concerne l'utilisation des ressources et des compétences
4. la robustesse qui concerne l'aptitude du système à suppléer la défaillance d'un agent
5. la non-redondance des actions qui caractérise le faible taux d'activités redondantes
6. la non-persistance des conflits qui témoigne du faible nombre de situations bloquantes

Ces indices répertorient les problèmes que nous avons soulevés mais ne sont pas utilisables en tant qu'indices dans notre cas. L'indice (1), par exemple, qui indique qu'un agent peut agir dans un sens et son contraire, n'est pas admissible en gestion de réseaux. On ne peut pas admettre un degré d'ajustement des actions, celles-ci doivent être logiquement organisées.
Pour éviter ces problèmes, nous devons donc dans un premier temps distinguer les tâches que l'on doit sérialiser et les tâches que l'ont peut paralléliser. Les tâches sérialisables peuvent faire l'objet d'une mise en file d'attente, l'agent devant alors gérer une file d'attente avec toutes les options qu'offre celles-ci comme la possibilité d'utiliser un système de priorité. Nous devons ici remarquer que ces tâches sérialisables sont essentiellement des tâches utilisant une ressource commune non partageable. Un exemple de ce type de tâche est l'établissement d'une connexion par modem sur un site distant. Il

est impossible de traiter simultanément les demandes de connexion sur des sites différents.

Les tâches parallélisables autorisent plusieurs activations simultanées de la même tâche. Un exemple de ces tâches est le monitoring d'éléments de réseaux. L'agent peut en effet prendre en charge la surveillance d'un nouvel élément de réseau alors qu'il en monitore d'autres.

Maintenant la question à résoudre pour le développement de la coopération dans l'architecture de nos agents DIANA est: à quel niveau placer le traitement de ces demandes concurrentes? Pour répondre à cette question, plaçons-nous dans le cas où c'est l'agent qui doit régler les problèmes relatifs à la concordance des tâches coopératives.

On considère (h1) que l'agent est capable de savoir qu'une activité est parallélisable. On considère (h2) que deux agents délégateurs délèguent une activité de monitoring portant sur des domaines D1 et D2 et que l'intersection D1,D2 est non vide

(h3): $D1 \cap D2 \neq \{\varnothing\}$

L'agent délégué utilise ses propres compétences de moniteur, et transmet à son module de compétences les demandes pour le monitoring de D1 puis pour D2.
Dans le cas où l'agent délégateur n'est pas capable de comprendre que l'intersection des domaines est non nul, celui-ci va logiquement démarrer deux activités identiques, dont la conséquence est que des équipements seront monitorés une fois de trop, entraînant une consommation de ressources pénalisante.
La simple connaissance qu'a l'agent de paralléliser les tâches de gestion n'est donc pas suffisante, il faut qu'il y ait aussi compréhension de la sémantique de la tâche elle-même, et qu'il soit capable de manipuler les éléments de cette sémantique pour effectuer par exemple des opérations de vérification des intersections.
Dans cet exemple nous n'avons considéré que les équipements à monitorer, et non pas les objets ni la précision du monitoring. Si nous prenons en compte ces éléments, nous augmentons la complexité du problème que doit gérer l'agent. En effet supposons que la délégation du monitoring porte sur les mêmes équipements et les mêmes objets, c'est-à-dire que:

(h4): $D1 \cap D2 = D1 = D2$

mais que pour D1 la fréquence soit de une minute, et pour D2 cinq minutes.

Nous comprenons facilement l'optimisation qu'il est possible de faire, et qu'il s'agit bien d'une même opération mais dont les résultats devront être transmis toutes les minutes au premier agent et toutes les cinq minutes au deuxième agent.

Nous voyons donc qu'il est pratiquement impossible pour l'agent de traiter ce type de problème d'une manière générique, et que la résolution des problèmes de cohérence d'une part, de concordance des tâches d'autre part, doivent être reportées au niveau de la compétence à laquelle appartient la tâche à exécuter.
Dans le cadre bien précis de la gestion de la QoS, [Oli98] présente un SMA dont les agents sont capables de fusionner deux objectifs de gestion et de les séparer lorsque l'un

des deux n'est plus utilisé. La réalisation de ces opérations est rendue possible en raison du contexte restreint de la qualité de service, où les éléments sur lesquels vont porter l'optimisation sont prédéterminés. Il faut néanmoins souligner que les architectures supportant l'optimisation des objectifs délégués sont exceptionnelles.
L'architecture DIANA se voulant plus générique, la résolution de ces problèmes sera donc envisagée en fonction des compétences, des opérations, et de l'organisation des applications.

IV.1.7 Apprentissage et adaptation

Nous avons déjà souligné que les mécanismes d'apprentissage utilisés en intelligence artificielle étaient peu adaptés à la gestion de réseaux. Celle-ci nécessite un certain niveau de déterminisme, et dans la plupart des cas il n'est pas envisageable d'attendre qu'un agent apprenne comment effectuer certaines tâches, pour avoir un service disponible.
L'apprentissage peut par contre être utile quand une application est supposée aider un utilisateur dans ses choix en fonction de ses goûts. Il faut alors que l'agent surveille en permanence les actions de l'utilisateur pour déterminer un profil, et que de son côté l'utilisateur ait un comportement stable ou régulier.

En examinant globalement le problème de l'apprentissage, c'est-à-dire l'augmentation des capacités de traitement d'un SMA, nous avons considéré que le transfert de compétences et de connaissances représente un mécanisme d'apprentissage plus efficace que ceux fournis par l'IA. En effet, ce que nous attendons avant tout de notre système d'agents intelligents n'est pas d'inventer une nouvelle façon de gérer les réseaux, mais bien de pouvoir traiter d'une manière décentralisée les problèmes de gestion, en fonction des besoins.

Pour cela les mécanismes de délégation dynamique, associés à la possibilité de transférer des connaissances entre agents, représentent un substitut intéressant.
Dans les cas de délégation que nous avons examinés au chapitre IV.1.5, les agents en charge des sous-réseaux ont appris comment monitorer les équipements et comment calculer les taux d'erreurs sans une quelconque intervention humaine. Ils se sont en quelque sorte adaptés aux nouvelles demandes qui leur ont été transmises.

Cette vision de l'apprentissage pourrait contrarier les puristes de l'intelligence artificielle, car l'agent ici n'a pas un comportement actif d'apprentissage, c'est-à-dire que ce n'est pas lui qui va chercher en permanence à apprendre.
Rodney A. Brooks spécialiste de l'IA modère cette approche et considère que "l'intelligence est dans les yeux de l'observateur" [Bro91a].
De la même manière, comme nous avons posé comme principe de n'apprécier les propriétés des agents qu'en n’utilisant une vision externe à ceux-ci, il n'y aura pas de différences apparentes entre un agent qui va apprendre tout seul à effectuer certaines opérations et un agent qui va rechercher les compétences permettant d'effectuer ces mêmes opérations.

Nous retiendrons donc que les propriétés de transfert de compétences et de connaissances sont le mécanisme d'apprentissage à utiliser par les agents DIANA pour s'adapter aux nouvelles demandes de gestion.

IV.1.8 Organisation

Dans le chapitre II, nous avons vu que la délégation s'est d'abord imposée comme une nécessité technique, dont le premier objectif est de transférer les procédures de traitement à proximité des données à traiter. Mais l'introduction des concepts d'agents intelligents a permis de rapprocher ces contraintes techniques des modèles d'organisation d'entreprises qui intègrent naturellement la délégation dans leur fonctionnement.
L'organisation est un moyen de maîtriser la complexité et les problèmes d'échelle que nous avons déjà évoqués.
[Mzh98] nous explique que les réseaux sont souvent construits après l'organisation de l'entreprise et reflètent cette organisation. D'après les auteurs, la délégation, la distribution des activités et l'organisation sont des éléments qui ont été exploités avec succès dans les entreprises, l'entreprise représentant de ce point de vue un modèle dont nous pouvons tirer des enseignements. Nous partageons cet avis, la comparaison entre une application distribuée développée, avec une approche agent et une entreprise, est d'autant plus intéressante que l'approche agent, comme le développement de l'entreprise, se base sur le travail des agents.

Organiser hiérarchiquement la distribution d'une application prend tout son sens lorsque l'on relie chaque niveau hiérarchique à un niveau d'abstraction différent. Comme dans une organisation d'entreprise, le chef d'entreprise ne va pas travailler en utilisant les détails de chaque facture, ni sur les contrats spécifiques à chaque client. Il va par contre utiliser des chiffres indiquant les tendances, donner les orientations, indiquer les politiques de gestion à suivre.
De la même manière on peut imaginer que le rôle du ou des responsables de la gestion d'un grand réseau va être en fonction des objectifs de rentabilité ou de satisfaction des clients d'établir la planification de l'utilisation des ressources, de prévoir la mise en place de nouveaux services ou la suppression d'anciens services. Pour cela il va déléguer la recherche d'informations contenant les tendances de l'utilisation du réseau ou des sous-réseaux. Ces tendances vont être déduites d'informations statistiques sur l'utilisation du réseau, statistiques déduites elles-mêmes des mesures faites périodiquement sur l'utilisation des ressources du réseau. Ces dernières mesures vont être déléguées aux agents en charge des équipements.

L'organisation hiérarchique repose donc sur le principe d'abstraction qui permet d'avoir une vue plus synthétique de l'ensemble des activités au fur et à mesure que l'on se rapproche du sommet de la hiérarchie. Dans l'autre sens une décision prise au sommet de la hiérarchie va être traduite (instanciée) différemment en fonction du contexte propre à chaque niveau.

Ce principe d'abstraction est utilisé différemment dans l'approche objet où il sert avant tout à extraire les éléments communs d'un groupe d'objets, que ce soit des propriétés ou des méthodes. L'approche agent permet de se concentrer sur le comportement du système et à chaque niveau de l'organisation des opérations effectuées par les agents, sur les données remontantes permettant d'élever le niveau d'abstraction.

Mais l'organisation hiérarchique n'est pas qu'une question de niveaux d'abstraction. L'organisation décrit des rôles et des relations qui relient les rôles entre eux. Ces relations déterminent les échanges d'informations entre ces rôles, et sont généralement orientées.

L'architecture des agents DIANA doit donc fournir un support pour la création d'une organisation hiérarchique, mais lequel? Pour [FG98], il existe très peu de recherches sur l'organisation des agents dans un SMA alors que les concepts d'organisation, comme le groupe, le rôle, ou la structure sont des facteurs clefs pour maîtriser la conception d'une application complexe. Les auteurs proposent une solution basée sur la gestion de groupes, lesquels sont attachés à un rôle, chaque agent pouvant devenir ou non membre du groupe en fonction de la décision du créateur du groupe. Chaque agent qui devient membre d'un groupe se voit attribuer les capacités et le rôle du groupe.
Si nous considérons cette approche comme intéressante dans le cas où les agents sont complètement autonomes, nous ne pouvons néanmoins pas l'utiliser dans un contexte d'organisation hiérarchique dans lequel les agents sont choisis pour remplir un rôle sans être réellement consultés.
Nous proposons donc une approche plus pragmatique, dans laquelle c'est l'administrateur du réseau qui déterminera lors du déploiement des applications les groupes d'agents qui travailleront pour cette application.
Cette approche reste cohérente avec la délégation dynamique et la coopération que nous avons décrite, un agent pouvant toujours demander à un autre l'exécution d'une tâche, ou déléguer une partie de ses activités à d'autres agents.

IV.1.9 Conclusion

Après avoir fait l'inventaire des concepts agents que nous devions utiliser, et des propriétés importantes dans le cadre de la gestion de réseaux, nous devons maintenant préciser l'architecture de cet agent DIANA.
Jusqu'à présent l'accent a été mis sur l'importance de la partie intelligente des agents, et sur leurs capacités à distribuer et à traiter des tâches de gestion. Nous avons privilégié une approche dynamique de l'intelligence, où l'organisation, le transfert de connaissances et la coopération pouvaient être modelés pendant la phase opérationnelle. Deux types de composants importants de l'approche agent ne doivent pas pour autant être oubliés, il s'agit des senseurs et effecteurs. En effet un agent perçoit et agit sur son environnement au travers de ces composants.
Le choix que nous avons ici est de considérer que ces composants font partie intégrante de tout agent ou que ceux-ci font partie des compétences que peut acquérir dynamiquement un agent.

Dans la perspective que l'architecture des agents DIANA peut être utilisée pour développer tout type d'application, il est clair que l'on ne peut déterminer au préalable les senseurs et effecteurs qui seront utiles à l'agent. Par conséquent il est logique de considérer ces composants comme étant la manifestation d'une compétence particulière des agents. Il est à ce propos difficile de faire la comparaison avec un être humain, celui-ci ayant traditionnellement cinq sens quelles que soient ses capacités intellectuelles. Néanmoins on peut considérer que tout être humain est capable d'utiliser de nouveaux senseurs qui vont lui permettre par exemple d'observer la lumière infra-rouge, ou écouter des micro-ondes et les décoder. Ses senseurs auront pour effet de traduire l'invisible en visible, ou l'inaudible en audible. C'est donc l'approche que nous avons utilisée pour l'architecture de l'agent DIANA.
Néanmoins la gestion de réseaux, en tout cas actuellement, est en grande partie axée sur des automatismes dans lesquels la "réflexion" n'est pas toujours nécessaire. Par exemple les reconfigurations automatiques de routeurs, la création d'une alarme lors du dépassement d'un seuil, ne nécessitent pas une intelligence particulière, mais plutôt une action rapide, ou plutôt une réaction.
C'est pour cela que nous avons aussi considéré la possibilité d'intégrer une composante réactive à l'architecture, faisant ainsi de DIANA une architecture classée dans la catégorie *hybride*.

IV.2 Couche délibérative

IV.2.1 Introduction

L'objectif que nous avons suivi en concevant l'architecture DIANA a été la validation de l'approche agent intelligent dans le cadre du développement de solutions pour la gestion de réseaux. Pour cela nous avons sélectionné les propriétés et les concepts les plus adéquats parmi ceux attribués aux agents intelligents.
En ce qui concerne le choix de l'architecture, notre étude sur les principales architectures d'agents intelligents nous a convaincus que celle qui correspond le mieux à notre objectif est l'architecture hybride.
Dans une telle architecture, c'est la couche délibérative qui contient les mécanismes de raisonnement de l'agent intelligent. L'étude que nous avons entreprise sur l'évolution des technologies de distribution nous a indiqué tout l'intérêt d'utiliser une approche par composants pour le développement de cette couche, ainsi que pour le développement de modules de compétences pour enrichir dynamiquement les capacités de traitement de l'agent, et permettre ainsi un déploiement flexible de toute application.
D'un autre côté la couche délibérative doit aussi offrir des services de délégation, c'est-à-dire la possibilité de transférer des objectifs de gestion, ou des rôles à d'autres agents. Il faut donc que cette couche offre une interface de communication de base, indépendante des compétences de l'agent, ainsi qu'un langage adapté au fonctionnement de ce même agent.

Dans cette section nous expliquons comment nous avons implémenté les concepts de généricité, d'autonomie, de délégation, afin de permettre l'émergence de l'intelligence des agents et la résolution des problèmes identifiés en gestion de réseaux.
Après avoir présenté la structure et les principaux composants de l'architecture de la couche délibérative, nous expliquons le fonctionnement de cette couche.

IV.2.2 Comportements d'un agent DIANA

Pour préciser le design de cette couche, il est utile de s'interroger tout d'abord sur les comportements de base que nous attendons d'un agent DIANA. En effet avant même les concepts, c'est le comportement de l'agent, fruit de la combinaison de ces concepts, qui représente tout l'intérêt des agents intelligents.
Nous avons donc défini trois comportements de base que l'agent devra avoir quelles que soient les circonstances, et ce dès son activation.

Connaissance de son environnement de travail:

Pour assurer la pérennité des activités de gestion en cas d'arrêt d'un des éléments, lorsqu'un agent se met en route, il doit vérifier s'il existe des croyances et des objectifs qui lui ont été laissés localement.
Par exemple, par défaut sur un équipement, un agent pourra avoir la connaissance de cet équipement (son identification, adresse, MIB), et des tâches relatives à la gestion de cet équipement.
Dans le cas contraire l'agent attend qu'on lui donne des instructions.

Connaissance de son environnement social:

Lorsqu'il se met en route, l'agent doit rechercher s'il peut communiquer avec les agents dont il connaît l'existence. Il met à jour à l'occasion des échanges de messages les informations relatives aux spécificités des autres agents, comme par exemple les protocoles qu'il peut utiliser pour dialoguer avec eux.

Prise en main de ses attributions:

Lorsqu'un agent a des objectifs, il doit rechercher comment les remplir, c'est-à-dire dans un premier temps rechercher les compétences nécessaires à l'exécution de ses objectifs, puis mettre en place les conditions pour atteindre ses objectifs.

Nous avons voulu en spécifiant ces comportements de base, garantir qu'un système multi-agents puisse se mettre en activité tout seul et immédiatement, sans pour autant alourdir son fonctionnement. Bien que possible, une recherche active des agents existants a de fortes chances d'être inutile, car un agent fait partie d'un système organisé, et les agents placés aux différents niveaux de la hiérarchie sont renseignés sur l'existence des agents qui dépendent d'eux.
D'autres comportements peuvent être rajoutés aux agents, ceux-ci étant spécifiés à l'agent lorsqu'il prend connaissance de son environnement de travail.

IV.2.3 Composants de l'architecture délibérative

La partie la plus délicate de l'agent DIANA est son cerveau, qui doit lui permettre de raisonner sur ses connaissances et ses capacités. Nous avons donc conçu ce cerveau comme un centre de répartition des informations. Ces informations sont aussi bien les messages venant des autres agents avec qui il communique, que les croyances produites entre autre par les modules de compétences.

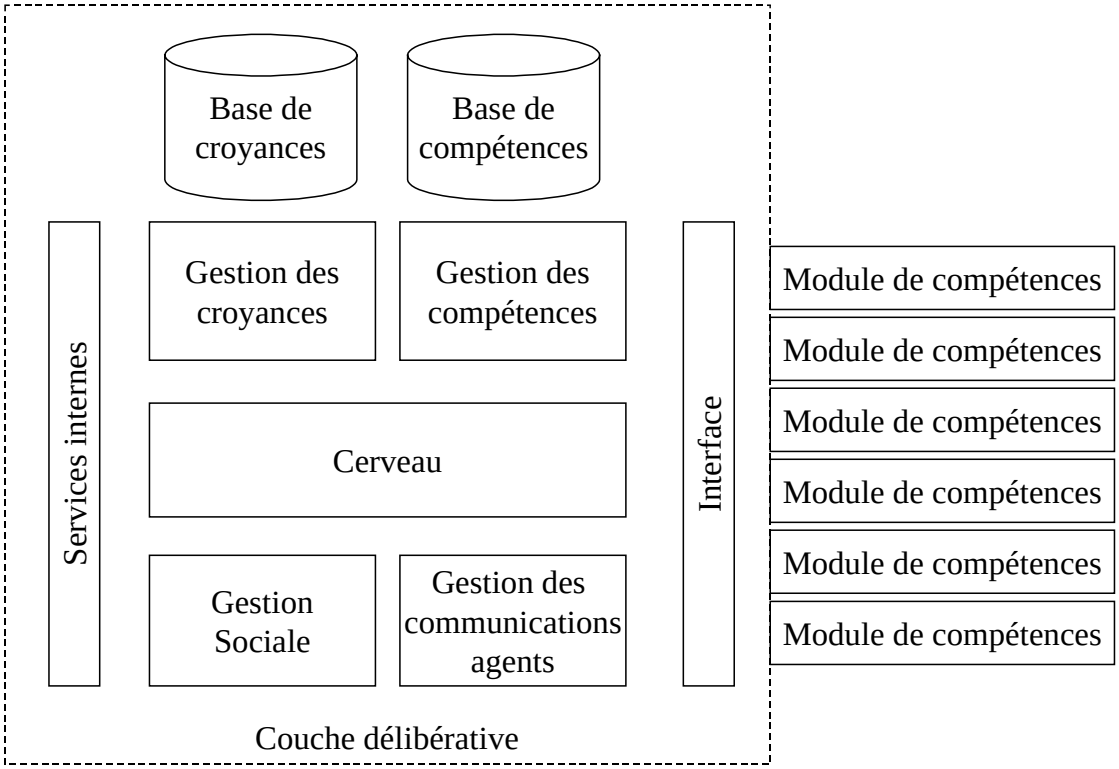


figure 15 Architecture DIANA: couche délibérative

Le schéma ci-dessus présente les composants essentiels de cette architecture. Une interface permet à l'agent premièrement de prendre en compte les capacités apportées par les modules de compétences, deuxièmement d'activer ou de stopper une tâche.
Deux composants de gestion utilisant des bases de données et fournissant des services de recherche, de mise à jour, de comparaison, sont à la disposition du cerveau. Il s'agit du composant de gestion de compétences et du composant de gestion des croyances. La communication avec les autres agents se fait en utilisant un composant de gestion sociale qui maintient à jour les informations sur les relations qu'entretient l'agent avec les autres agents, et un composant de gestion de communications qui dispose de plusieurs protocoles pour communiquer avec les autres agents.
Enfin un composant offre des services internes par l'intermédiaire du langage de l'agent pour suivre les tâches en cours, ou les activités démarrées par l'agent, et permet de demander à l'agent de charger ou de décharger un module de compétences.

Afin d'offrir un *comportement interactif* l'architecture de l'agent a été étudiée pour fonctionner suivant un *principe événementiel*, tous les échanges entre composants se faisant grâce à des événements. Les principaux événements que va gérer l'agent sont des opérations sur les croyances, comme par exemple la création, la suppression ou la modification des croyances (voir section croyances). Le travail du cerveau de l'agent étant de rechercher les composants qui peuvent être intéressés par ces événements et de les leur transmettre.

Enfin l'agent devant assumer plusieurs rôles simultanément, les modules de compétences sont chargés dans des threads (unités d'exécutions) séparés, permettant ainsi la *parallélisation de l'exécution des tâches.*

IV.2.4 Description des composants

IV.2.4.1 Gestion des croyances

Une croyance représente toute information manipulée par le cerveau de l'agent. Ces informations peuvent provenir des modules de compétences, d'autres agents, ou directement d'administrateurs. La croyance est caractérisée par plusieurs informations, qui sont renseignées et utilisées par le cerveau ou les composants de l'agent.
En tout premier lieu la date de création de la croyance, qui associée au type de croyance (temporaire, pérenne, privée) et à sa durée de vie permet au gestionnaire de croyance de supprimer toute croyance qui n'est plus valide.
Par exemple une croyance qui contient le nom et l'adresse d'un agent, ou les références et l'adresse d'un équipement est considérée comme pérenne, tandis qu'une croyance sur le nombre d'erreurs signalé par l'interface d'un équipement est une information temporaire, dont la durée de vie peut être fixée en fonction de la période de récupération des informations.
La deuxième information caractérisant une croyance est l'identifiant du créateur de la croyance. Ce créateur est soit un module de compétences, soit un autre agent, soit l'agent lui-même. Cette information est particulièrement importante pour que l'agent puisse par exemple satisfaire son service d'abonnement. Si nous verrons plus en détails à la fin de ce document le principal intérêt de ce mécanisme, on peut déjà citer le cas où un agent superviseur veut être informé lorsqu'un agent démarre une activité.
Par défaut, c'est-à-dire automatiquement lorsqu'un agent délègue la réalisation d'une tâche à un autre agent, il est prévenu grâce à ce mécanisme d'abonnement, des croyances produites par la tâche. Pour cela le gestionnaire de croyances utilise les références du créateur des croyances pour savoir à qui adresser les informations (elles-mêmes étant des croyances).
Enfin la dernière information est le contenu même de la croyance. La sémantique des croyances utilisée par les agents DIANA a été maintenue la plus simple possible afin d'être facilement et rapidement utilisable par des êtres humains sans expérience approfondie des agents.

Croyance ::= [Terme | Identifiant] Paramètres
Terme ::= Chaîne
Identifiant ::= Sk Chaîne " " Opération
Opération ::= Chaîne
Paramètre ::= ":" Chaîne " " Valeur Paramètre
Valeur ::= Variable | Chaîne
Variable ::= "$" Chaîne
Chaîne ::= Lettre { Lettre | Digit }

Le gestionnaire de croyances offre les services suivants:

1. *Service de création/suppression/mise à jour*: permet de maintenir la connaissance qu'a l'agent de son état et de son environnement
2. *Service de recherche et de comparaison*: permet à un module de compétences ou à un autre agent de rechercher un ensemble de croyances correspondant à un modèle (pattern)
3. S*ervice d'abonnement*: permet d'informer un module de compétences ou un agent qu'une action particulière (ex: création) a été effectuée sur une croyance correspondant à un pattern fourni par le module de compétences ou l'agent.

IV.2.4.2 Modules de compétences

Lorsqu'une compétence est chargée par l'agent un certain nombre d'informations sont récupérées par le gestionnaire de compétences. Ces informations concernent à la fois les *capacités* de traitement attachées au module de compétences, et aussi les compétences connexes prérequises.
Les **capacités** sont décrites par les informations suivantes:

1. (o) Opération: identifiant de l'opération permettant l'utilisation de la capacité avec ses paramètres d'exécution
2. (p) Préconditions: liste de pattern de croyances qui indique que l'opération peut avoir lieu
3. (u) Croyances utilisées: liste de pattern de croyances que l'agent doit communiquer au module de compétences lorsque l'opération est en cours
4. (r) Opérations invoquées: liste d'opérations que le module de compétences peut activer pendant le déroulement de l'opération
5. (c) Croyances Produites: liste de pattern de croyance que le module produit en cours ou à la fin de l'opération

Les compétences prérequises identifient quant à elles les modules de compétences que doit utiliser l'agent si celui-ci veut établir les préconditions d'activation des opérations

La figure 16 décrit les principaux états d'un module de compétences et des opérations.

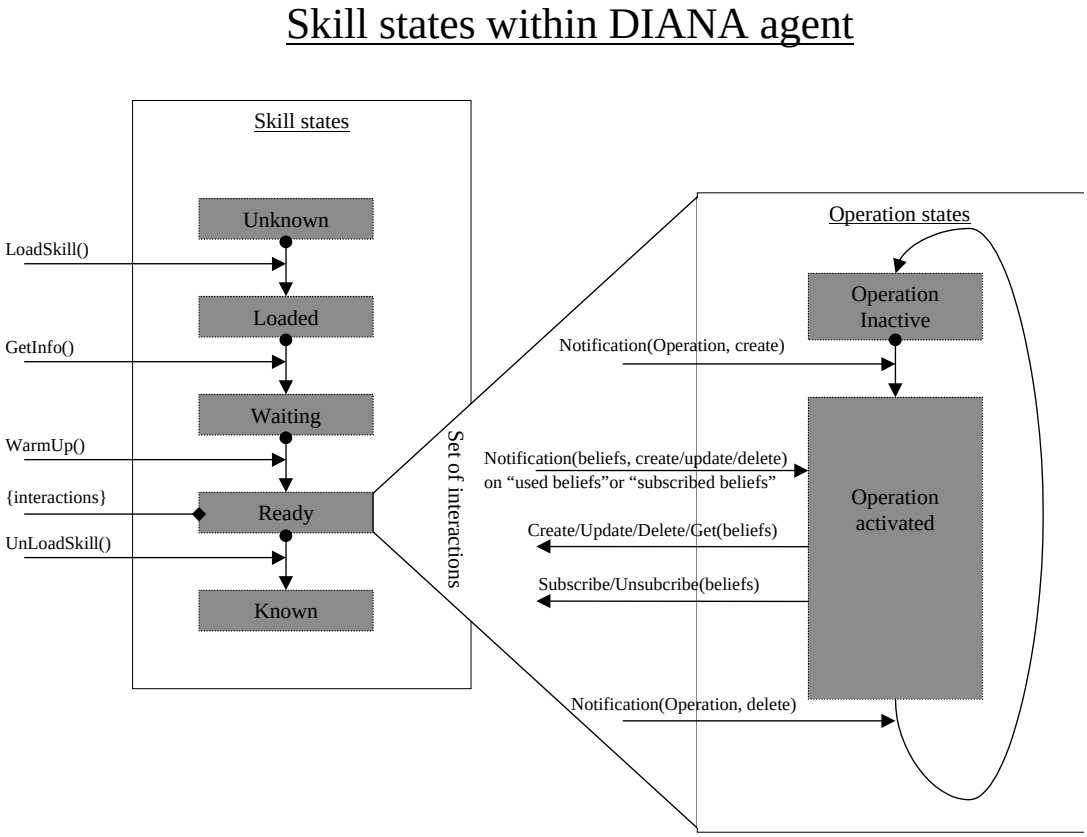


figure 16 Etats d'un module de compétences

Nous voyons sur cette figure qu'après son chargement, le gestionnaire de compétences récupère grâce à la commande *GetInfo* les informations sur les capacités contenues par le module, et ne le considère comme opérationnel qu'après l'exécution d'une opération de mise en route (*warmup*). Cette opération de mise en route est quelquefois indispensable pour que le module ait le temps de préparer son environnement d'exécution avant d'être utilisé.
Une fois la mise en route effectuée l'agent peut utiliser le module de compétences en lui transmettant des événements avec la commande de notification.
Le module de compétences reçoit et émet des événements dont le contenu sont des croyances, au travers des commandes offertes par l'interface.
Pour utiliser une capacité fournie par le module de compétences, il faut créer une croyance comportant l'identifiant de l'opération (o) et les paramètres associés. Le module de compétences, à la réception d'une notification indiquant la création d'une telle croyance, va activer l'opération. La vérification de l'existence des préconditions étant à la charge de l'agent.
De la même manière pour stopper une opération, l'agent doit notifier le module de compétences que la croyance associée à cette opération a été supprimée (commande delete de l'interface).

IV.2.4.3 Interface

L'interface permet à l'agent et aux modules de compétences de travailler ensemble. Il faut en effet se rappeler qu'un module de compétences possède sa propre unité d'exécution (un thread) et que les échanges entre le cerveau et le module de compétences sont des événements asynchrones.
Le cerveau de l'agent accède au module de compétences par l'intermédiaire de l'interface avec les commandes suivantes:

- *Getinfo*: pour avoir la description complète des capacités du module de compétences
- *Notify*: pour informer le module de compétences de l'existence d'une croyance le concernant.

De son côté les modules de compétences ont besoin d'échanger des informations avec d'autres modules, et ils utilisent pour cela les commandes suivantes de l'interface:

- *Create*: pour créer une nouvelle croyance
- *Delete*: pour supprimer une croyance
- *Update*: pour mettre à jour une croyance
- *Get* : pour rechercher des croyances
- *Subscribe*: pour recevoir des croyances particulières ou être informé des actions effectuées sur ces croyances
- *Unsubscribe*: pour se désabonner
- *Verify*: pour vérifier qu'un objectif peut être atteint
- *Achieve*: pour demander qu'un objectif soit atteint

Ces différentes peuvent être transmises à d'autre agents comme nous le verrons plus loin, grâce au module de communication.

IV.2.4.4 Gestionnaire de compétences

Ce composant est responsable du chargement des modules de compétences et de la maintenance des capacités totales de l'agent. C'est grâce à lui que l'agent peut répondre ou non aux demandes de prise en charge de rôles et exécuter des tâches de gestion. Il établit les relations entre les modules de compétences ou entre les opérations proposées par ces modules, et sait quelles opérations activer pour obtenir les croyances désirées.

Pour obtenir ce résultat, le gestionnaire de compétences manipule des tuples de capacités $(o_i, p_i, u_i, r_i, c_i, s_i, t_i)$, ou s_i indique l'état des opérations (sans, prise en compte, activée, suspendue, arrêtée), et t_i identifie le demandeur. Grâce à cela et en relation avec le gestionnaire de croyance, il peut assurer un certain nombre de services:

- Service de planification (*verify, achieve*): vérifie que les préconditions nécessaires pour activer une opération peuvent être produites
- Service d'activation des opérations (*create, subscribe*): mise en place de l'environnement de suivi, d'utilisation et de contrôle d'une opération
- Service de suppression des opérations (*delete, unsubscribe*): suppression des opérations liées, et relâchement des ressources utilisées par l'opération

IV.2.4.5 Gestion des communications

Les communications entre agents se font par défaut en utilisant directement le protocole TCP/IP, mais d'autres protocoles sont disponibles comme HTTP, ou TELNET, permettant ainsi à un administrateur de dialoguer sans autre interface avec un agent.
Pour s'échanger des messages les agents utilisent le service interne *sendTo*, les opérations sont alors directement traduites en format texte et envoyées à l'agent distant.
La syntaxe est la suivante:

SendTo agent croyance

où *Agent* désigne le ou les agents destinataires et *croyance* est une croyance telle que décrite précédemment.
Lorsqu'un module de compétences émet une telle requête, l'agent vérifie que l'agent est connu grâce au gestionnaire social, et il transmet alors la demande au service interne, qui s'occupe de convertir l'opération souhaitée en texte, et transmet ensuite la demande au gestionnaire de communication.
Dans le cas où *agent* désigne plusieurs agents, les demandes sont envoyées séparément, le composant de communication ne gérant pas le multi-cast.
Tous les échanges sont identifiés afin d'assurer un suivi de la conversation. Par exemple lorsqu'un module de compétences utilise la commande suivante:

*getbelief ("sendTo Master task * started * ")*

cette demande est transmise au module de communication après traduction pour devenir:

*from Diana to 193.55.114.44 33 12/04/2000 13:24 678 GETBELIEF Master task * started **

Bien qu'inspiré par les langages de communications en vogue dans le milieu de la recherche sur les agents, comme KQML ou ACL, le système de communication proposé ici se veut plus pragmatique. Tout d'abord, les notions de langage et d'ontologie défendues par les langages classiques ne sont pas utilisées. Car comme nous avons déjà eu l'occasion de le défendre, elles sont essentiellement utilisées pour des communications entre des agents d'origines diverses, et nous considérons comme fortement prématuré le rêve que des agents de conception différentes puissent travailler ensemble. Ensuite des langages comme KQML imposent l'utilisation d'un agent facilitateur (une sorte de facteur) pour la distribution des messages, intervenant de ce fait dans l'organisation des

agents. Or nous avons voulu laisser au gestionnaire de réseaux la liberté du choix de l'organisation de ses agents. Enfin, la sémantique et la syntaxe que nous avons conçues sont beaucoup plus simples à utiliser par un être humain sans avoir besoin d'interface particulière.

IV.2.4.6 Cerveau

Le travail du cerveau est essentiellement un travail de répartition des croyances et de coordination des opérations.

Lors de la présentation de l'interface, nous avons vu qu'il existait un certain nombre d'opérations sur les croyances. Ces opérations sont transmises au cerveau, qui va utiliser ses composants périphériques pour analyser la croyance associée et déclencher les comportement adéquats.
Toutes les opérations sur les croyances pouvant s'adresser à des agents distants, le premier terme des croyances est vérifié pour savoir s'il s'agit d'une demande à transmettre à un agent distant (*sendTo*). Dans ce cas là, l'opération n'est pas effectuée localement, mais transmise au composant de gestion sociale pour identification des correspondants, puis au gestionnaire de communication. Dans le cas contraire les opérations sur les croyances sont traitées localement.

Les opérations de création peuvent avoir trois significations différentes:

- déclenchement d'un service interne à l'agent comme par exemple la mise en fonction du mode de traçage des activités.
- demande d'exécution d'une tâche ou d'une activité comme par exemple la modification de la configuration d'un routeur, ou la surveillance du comportement des agents
- enregistrement d'une information sur l'environnement de l'agent ou sur son activité

Une opération de suppression de croyance aura une signification inverse:

- arrêt d'un service interne à l'agent
- arrêt d'une tâche ou d'une activité
- suppression d'une information sur l'environnement de l'agent

Une opération de mise à jour d'une croyance, de toute évidence ne concernera que la connaissance de l'agent et sera donc transmise au composant de gestion des croyances, et il en est de même pour les opérations d'abonnements.
Les opérations sur les objectifs (*verify(), achieve()*) seront quant à elles transmises au composant de gestion des compétences, car elles concernent l'utilisation des capacités de l'agent apportées par les modules de compétences.

Mais ce composant a une autre fonction que la distribution des informations, c'est la mise en place de son environnement. Pour cela le cerveau utilise un fichier d'initialisation, qui permet de sauvegarder les croyances, mais aussi les activités, tâches ou objectifs, que l'agent doit connaître dès sa "naissance".

IV.2.5 Fonctionnement de la couche délibérative

La couche délibérative utilise ses composants et leurs services pour remplir les missions suivantes:

1. Compréhension des informations entrantes par analyse des croyances
2. Analyse des demandes de gestion pour les traduire en une suite d'exécution d'opérations
3. Suivi des opérations, contrôle des opérations
4. Recherche, transmission, chargement, déchargement, de modules de compétences
5. Relation de délégation avec les autres agents
6. Acceptation ou refus des opérations de gestion

Pour remplir ces missions la couche délibérative utilise deux types de fonctionnements:

1. **fonctionnement orienté tâches**
2. **fonctionnement orienté objectifs**

Le fonctionnement est dit orienté tâche lorsque l'on spécifie à l'agent la tâche (ou l'activité) que l'on veut lui voir effectuer. Le fonctionnement est dit orienté objectif lorsque l'on veut que l'agent mette en place lui même les préconditions, en déclenchant les opérations correspondantes, pour l'obtention d'un résultat qui est l'objectif. Dans les deux cas l'agent utilisera les connaissances qu'il a sur son environnement, et sur ses compétences.

Grâce à ces deux types de fonctionnement, l'agent offre trois possibilités d'activation des opérations.

Activation d'une opération par demande d'exécution

Pour déclencher l'activation des opérations correspondantes à la tâche à accomplir, une entité (administrateur, agent, module de compétences) doit créer une croyance en utilisant la sémantique présentée précédemment. Cette croyance sera analysée par le cerveau en utilisant l'algorithme qui après avoir vérifié que la croyance lui est bien destinée, et qu'elle ne correspond pas à un service interne à l'agent, la transmettra au gestionnaire de compétences.
Lorsque celui-ci reçoit la demande de création, celui-ci recherche dans sa base de données s'il a la compétence nécessaire pour exécuter la tâche ou l'activité décrite dans la croyance. Dans le cas contraire, un refus motivé par l'absence de compétence est retourné

au demandeur, avec charge à lui de réclamer que l'agent acquière cette compétence (transfert de compétences), ou de trouver un autre agent pour l'exécution de cette opération.
Dans le cas où l'opération est identifiée, ainsi que le module de compétences, le gestionnaire de compétences vérifie auprès du gestionnaire de connaissances, si toutes les préconditions sont satisfaites. Dans la négative la demande d'activation est aussi retournée au demandeur qui doit alors réclamer s'il le désire que les préconditions soient établies (par exemple en utilisant la commande *achieve()*), ou déléguer à un autre agent l'activation de cette opération.

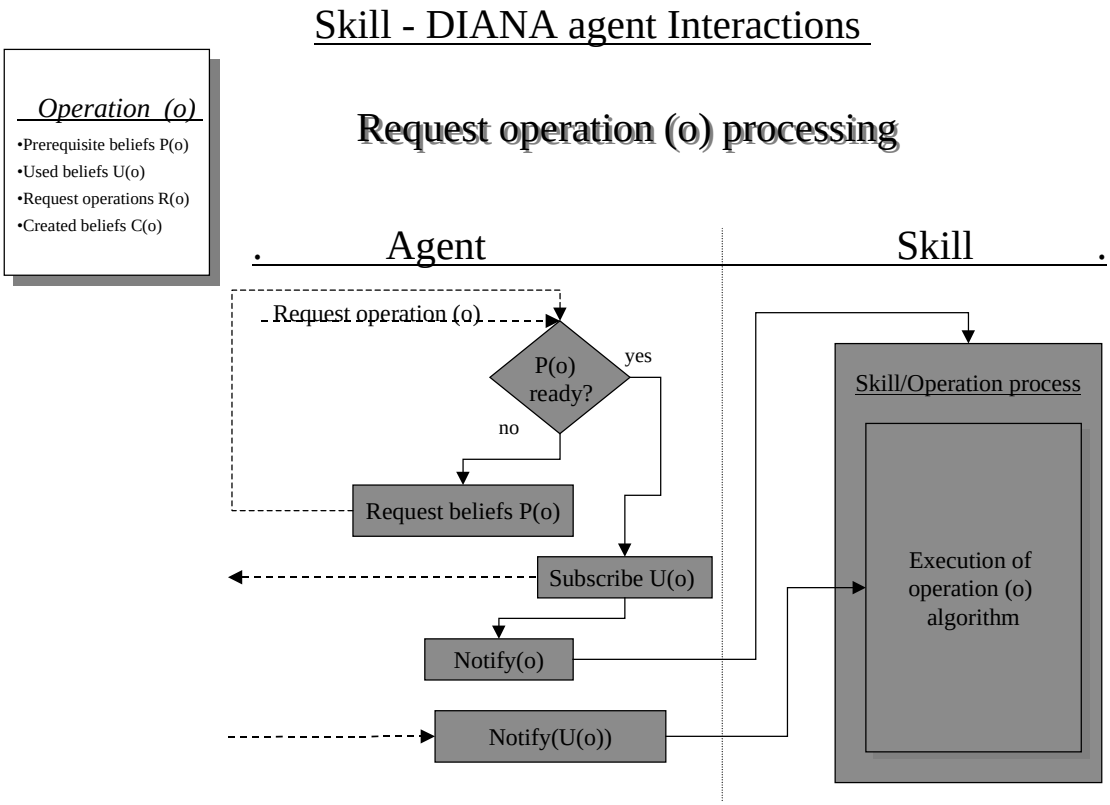


figure 17 Activation d'une opération

Lorsque les préconditions sont satisfaites (voir figure 17) alors le gestionnaire de compétences crée lui-même une croyance pour indiquer que l'opération est prise en compte. Puis il abonne auprès du gestionnaire de connaissances, le module de compétences, sur toutes croyances signalées comme nécessaires à l'opération (les *used beliefs(o)*).

Ainsi le module de compétences sera informé automatiquement de l'existence, de la modification ou de la suppression des paramètres influençant l'exécution de l'opération (tâche ou activité).
La deuxième opération d'abonnement qui doit être effectuée, concerne l'abonnement du requérant (ex: un autre module de compétences) ou délégateur (si c'est un autre agent) toujours auprès du gestionnaire de croyances. Cette abonnement porte sur les croyances

produites en tant que résultats de l'opération. Il peut s'agir de résultat intermédiaire, dans le cas d'une activité (ex: monitoring) ou de résultat définitif dans le cas d'une tâche.

Ensuite le gestionnaire de compétences envoie (via l'interface) une notification au module de compétences concerné, notification qui comprend l'intégralité de la croyance.

A ce stade l'opération est prise en compte par le module de compétences qui peut la démarrer effectivement.
Une fois le module de compétences notifié, le gestionnaire de compétences modifie la croyance concernant l'état de l'opération pour la signaler activée.

Activation d'une opération par demande d'objectif

Par rapport à l'activation par demande d'exécution, l'activation par demande d'objectif demande une plus grande autonomie d'action de la part de l'agent.
Pour déclencher l'activation des opérations correspondantes à un objectif à atteindre, une entité (administrateur, agent, module de compétences) doit utiliser la commande *achieve()* dans laquelle la croyance contenue décrira le résultat attendu c'est-à-dire l'objectif à atteindre.
Après avoir vérifié que cette commande ne doit pas être traitée localement, le cerveau la transmet au gestionnaire de compétences qui va effectuer le travail suivant.
Tout d'abord il recherche parmi l'ensemble des compétences disponibles s'il y en une qui permet d'atteindre cet objectif. Cette recherche utilise la base de données sur les croyances productibles par les modules de compétences connues du gestionnaire de compétences. Dans le cas où l'agent ne connaît pas encore de module de compétences capable d'atteindre cet objectif, il recherche localement s'il peut trouver un module de compétences correspondant, et dans la négative, contacte les autres agents pour démarrer une recherche de compétence.
Il faut signaler ici que les agents possèdent un algorithme de recherche par propagation, dont on peut optimiser l'efficacité en restreignant le domaine des agents impliqués dans la distribution des compétences.
Auparavant le gestionnaire de compétences crée une croyance indiquant qu'une recherche est en cours pour atteindre l'objectif.
Dans le cas où les recherches échouent l'entité qui a émis la demande est contactée pour lui indiquer qu'il n'existe aucune compétence connue permettant d'atteindre son objectif.
Maintenant, une fois le module de compétences trouvé et chargé, le gestionnaire de compétences utilise la commande *verify()* pour vérifier que l'objectif peut être atteint. Cette commande utilise le même principe que la commande *achieve()* en recherchant et en chaînant les préconditions nécessaires à l'exécution des différentes opérations à déclencher jusqu'à trouver des préconditions déjà disponibles, ou de rencontrer un cas où il est impossible d'obtenir une précondition. La commande *verify ()* utilise par contre les informations stockées par le gestionnaire de compétences, sans interroger les modules eux-mêmes, permettant ainsi à l'agent de construire un plan même si les modules de compétences ne sont pas disponibles en mémoire (voir figure 18).

Dans le cas où l'agent a pu construire un plan pour atteindre l'objectif, la croyance correspondante à la demande est modifiée pour indiquer que l'objectif est réalisable. Puis

les croyances sont créées pour déclencher l'activation des opérations (demandes d'exécution) en fonction du plan établi.
Il faut noter ici que l'entité qui a émis la commande *achieve()* n'est informé que de l'état de sa demande (pris en compte, réalisable, en cours, échouée, terminée) et non des résultats intermédiaires, qui sont les croyances produites par les différentes opérations déclenchées pour atteindre l'objectif.

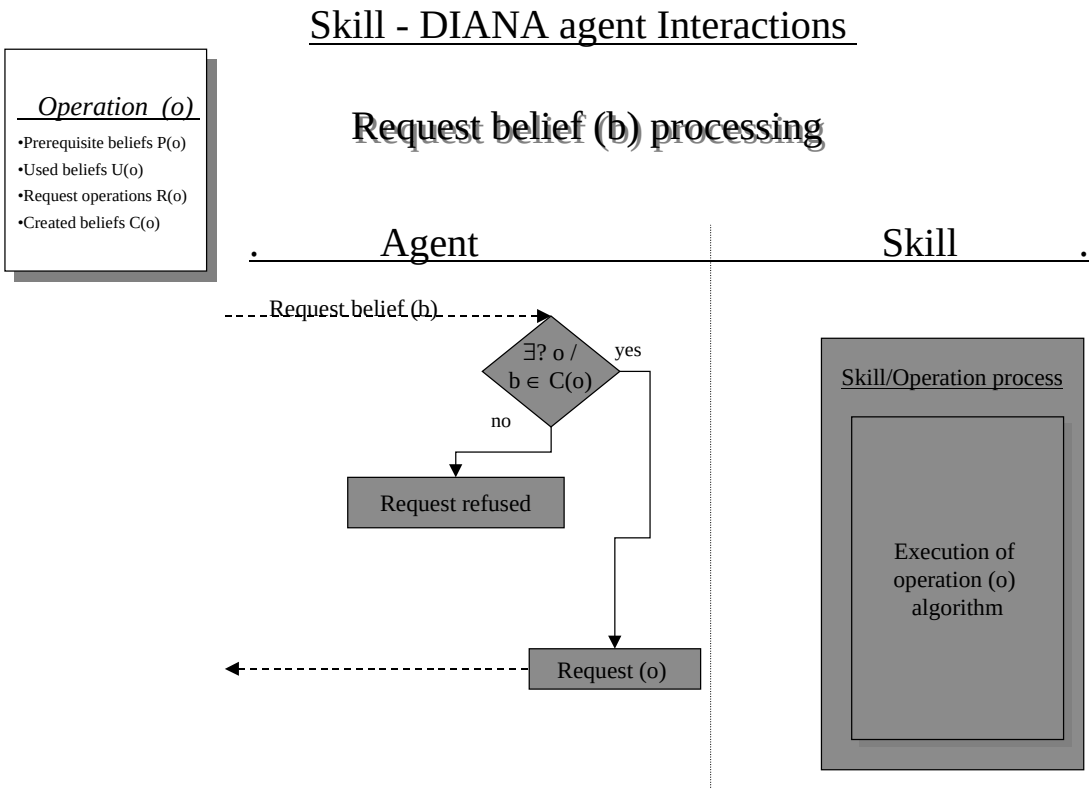


figure 18 Interaction agent - compétences : recherche d'un plan

Cette approche permet la mise en place de la délégation de rôle présentée au chapitre IV.1.5.3.

Activation d'une opération par changement d'état

L'activation par changement d'état ne correspond pas à un fonctionnement différent de ceux présentés précédemment, mais plutôt une conception différente.
Dans les deux approches précédentes nous avons focalisé notre attention sur le déroulement des algorithmes utilisés par l'agent. Nous avons omis de spécifier si les opérations correspondent à des tâches ou à des activités, car cela ne change pas le fonctionnement, et donc le comportement de l'agent. Si l'agent doit démarrer une activité de surveillance, ou modifier des configurations sur un élément du réseau, un observateur verra quasi immédiatement que l'agent agit pour lui en fonction de sa demande.
Maintenant il est possible de considérer la réalisation de modules de compétences dont la mise en fonction ne se fera que sur certaines conditions.

Pour mieux comprendre cette approche, prenons l'exemple d'un agent en charge de la garantie de la qualité de service des connexions de vidéoconférences. Il est en relation avec plusieurs opérateurs qui lui communiquent les tarifs et les qualités disponibles. A un temps *t* donné il existe une connexion entre deux sites et l'agent grâce à qui il est informé régulièrement de l'évolution de la qualité de service concernant cette connexion, ainsi que de la possibilité d'avoir d'autres connexions. Ces informations sont interprétées grâce à une de ses compétences (ex: SkQoS), et transformées en croyances qui indiquent si le niveau de QoS est acceptable ou pas:

SkQoS :connection X :status OK

A *t+1* la croyance sur la QoS de la connexion est modifiée, indiquant que la QoS n'est plus acceptable:

SkQoS :connection X :status KO

La modification de cette croyance indique un changement d'état propre à provoquer l'activation d'une opération, dont l'objectif sera d'établir une connexion de rattrapage offrant la QoS prévue, et de basculer le trafic sur cette nouvelle connexion.
Ce comportement peut être comparé aux comportements proposés par les adeptes des approches BDI (*Belief Desire Intention*) que nous avons abordées chapitre II.2.2, dans lesquelles il correspond à la violation d'une motivation, mais aussi à un comportement réactif que nous verrons plus loin.

Dans tous les cas que nous venons de voir l'activation d'une opération passe obligatoirement par la création d'une croyance correspondante, qui permet aux composants de cette couche délibérative de garantir le suivi et la logique des opérations.

IV.2.6 Conclusion

Lors de la conception de cette couche délibérative, nous avons dû faire certains choix afin de la maintenir la plus ouverte possible. L'objectif étant avant tout comme nous l'avons rappelé, de valider une approche plutôt que de développer encore une nouvelle technologie agent. Si l'intelligence de l'agent est perceptible par le comportement qu'il exhibe, et non pas par la technologie qu'il utilise, nous avons néanmoins dû tenir compte du degré de complexité qui serait nécessaire pour développer ces comportements.
Nous avons donc fait un compromis entre l'approche déclarative largement utilisée pour le développement de systèmes experts, mais lente et rapidement complexe à concevoir, et l'approche procédurale, plus performante mais moins flexible. Nous avons aussi tenu compte des contraintes d'occupation des ressources, l'agent générique devant être particulièrement "léger", et n'utiliser de nouveaux composants que lorsqu'il y en a effectivement besoin. Et enfin nous avons tenu compte des échanges inter-agents en proposant un système simple d'échange d'informations.

IV.3 Couche réactive

IV.3.1 Introduction

La couche réactive est la couche basse d'une architecture hybride, c'est-à-dire celle qui va permettre à l'agent d'agir par réflexe sur son environnement, par opposition à la couche délibérative qui va "réfléchir", c'est-à-dire qui va utiliser des algorithmes plus complexes face à un changement de situation.
Par définition, le fonctionnement de la couche réactive doit être plus simple que celui de la couche délibérative, et ceci pour répondre à plusieurs objectifs.
Le premier objectif de la couche réactive est de fournir à l'agent une **capacité de réaction** rapide, en cas de problèmes sur le réseau ou sur les systèmes, permettant par exemple de reconfigurer les services ou les équipements et de maintenir ainsi le réseau en état de fonctionnement.
Le deuxième objectif de cette couche est de **renseigner la couche délibérative** sur l'évolution de l'état du réseau. Au niveau de la couche réactive il est possible de faire du filtrage ou de la corrélation d'événements, et préparer ainsi des informations synthétiques pour la couche délibérative.
Enfin le troisième objectif est de fournir à l'administrateur de réseaux la possibilité de **programmer des comportements réactifs** pour son système multi-agents, et remplacer des langages comme PERL.

Au chapitre III.2.1 nous avons présenté le principe d'une architecture réactive qui peut se résumer à la séquence:

$$\textit{événement} \rightarrow \textit{modification de la situation} \rightarrow \textit{action} \quad (A)$$

La situation perçue au niveau de la couche réactive est nécessairement différente de celle perçue au niveau de la couche délibérative. Cette couche réactive doit conserver une vision du monde du réseau assez proche de celle fournie par les éléments du réseau lorsqu'on les interroge, afin d'offrir une réactivité réelle.
Malgré tout il est souvent indispensable de combiner des informations entre elles, arithmétiquement ou logiquement, pour avoir une vision utilisable de l'état du réseau (*la situation*), et de son évolution. Par exemple pour savoir s'il y a une détérioration effective de la qualité d'une transmission, il ne suffit pas de savoir si le compteur d'erreurs s'incrémente sur une des interfaces, il faut mettre en rapport la valeur de ce compteur avec le nombre d'octets échangés et avec le temps écoulé entre deux mesures.
Pour cela nous avons utilisé le concept d'indicateur présenté ci-après et défini un concept de réacteur pour décrire la séquence (A).

IV.3.2 Indicateurs

Pour comprendre ce que sont les indicateurs et à quoi ils servent examinons quelques exemples d'indicateurs et leur utilisation:

1) Indicateur de vitesse d’impression: pour détecter l'impression de fichiers postscript en format texte;
2) Indicateur de taux de charge d'un serveur NFS: pour prévoir le changement du serveur ou planifier une meilleur répartition des données;
3) Indicateur du taux de paquets ICMP: pour détecter une tentative de sabotage du réseau;
4) Indicateur du taux d'erreurs pour chaque port d'un commutateur: pour vérifier le fonctionnement normal des équipements connectés;
5) Indicateur de taux d'occupation de la bande passante d'un lien: pour planifier la modification des configurations, et le changement d'équipements.

Les indicateurs sont avant tout des informations synthétiques résultant de la combinaison arithmétique ou logique d'informations de bas niveau, comme celles fournies par les MIBs des équipements de réseaux.

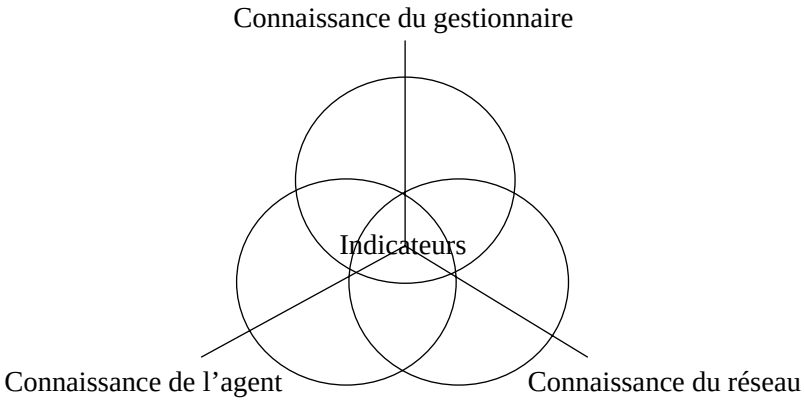


figure 19 Composition des indicateurs

La création d'un indicateur fait donc intervenir des opérateurs, qui vont être reliés à des opérandes exprimant des valeurs provenant de différentes sources. Il existe trois sources de données permettant de définir un indicateur (voir figure 19).

Nous avons indiqué dans les objectifs de la couche réactive, que celle-ci devrait être utilisable par le gestionnaire de réseaux pour programmer des indicateurs et des réacteurs. A ce titre ce dernier va spécifier certaines informations comme la précision de ces indicateurs, c'est-à-dire entre autre la fréquence à laquelle il souhaite que l'indicateur soit recalculé, ou aussi spécifier un paramètre particulier comme la valeur d'un seuil.
La deuxième source d'information est l'agent lui-même, et plus précisément sa partie délibérative. S'il est quelquefois attribué à la couche délibérative le rôle de la programmation des réactions, nous avons dans notre approche laissé cette possibilité à quelques modules de compétences qui pourrait être développés spécifiquement dans cette optique. La couche délibérative intervient néanmoins dans la programmation de la couche réactive en fournissant à celle-ci des données non précisées lors de la conception d'un indicateur. Supposons qu'une entité quelconque conçoive un indicateur pour connaître le taux d'erreurs d'une interface, il est judicieux de laisser à l'agent le soin de spécifier les coordonnées de cette interface (adresse de l'équipement et OID), au moment de l'instanciation de cet indicateur, plutôt que de prévoir autant d'indicateurs que

d'interfaces. La couche délibérative de l'agent va donc fournir des renseignements à la couche réactive pour préciser les paramètres des indicateurs.
Enfin bien entendu ce sont les éléments des réseaux qui au travers de leurs interfaces de gestion vont renseigner la couche réactive sur les valeurs propres aux objets de gestion.

Voici ci-dessous quelques exemples des informations fournies par les entités que nous avons présentées:

Connaissance du gestionnaire:

- ▷ Les seuils de tolérance
- ▷ Les fréquences de rafraîchissement à utiliser
- ▷ Les types d'équipements
- ▷ Les domaines de gestion

Connaissance de l'agent:

- ▷ Les équipements
- ▷ Les MIBs spécifiques

Connaissance du réseau:

- ▷ Les valeurs
- ▷ Les évènements
- ▷ Les protocoles

L'approche qui consiste à instrumenter des indicateurs directement au niveau des agents, c'est-à-dire des entités chargées de collecter les informations sur la santé du réseau, n'est pas partagée par tous. [MH99] propose que tous les événements soient centralisés et d'utiliser des browsers d'événements qui permettraient de filtrer les événements en construisant des sortes d'indicateurs qui seraient ensuite visualisés grâce à une interface spécifique. Malgré tout, la tendance justifiée par une plus grande extensibilité de l'approche distribuée, comme nous l'avons fait remarqué plusieurs fois, est de répartir le traitement en le localisant le plus près possible de la source émettrice d'informations, ce qui se traduit avec les agents en premier lieu par la conception des indicateurs.

Les valeurs de ces indicateurs pourront aussi être conservées s'il est nécessaire d'analyser leur historique, tout en utilisant moins de ressources que ne l'aurait exigé des informations de bas niveau.

Le type des indicateurs

Un indicateur doit être d'un type simple pour être facilement utilisé, mais dans certains cas il est intéressant d'utiliser des types un peu complexes comme les tables ou les vecteurs. Dans le modèle de couche réactive que nous avons défini, nous avons décidé qu'un indicateur pourrait être l'expression directe d'un objet de MIB, et donc d'utiliser les types de données définis par l'IETF pour son SMI. Ce choix nous permet de concevoir

tout indicateur comme un ensemble d'indicateurs reliés par des opérateurs (voir figure 20).

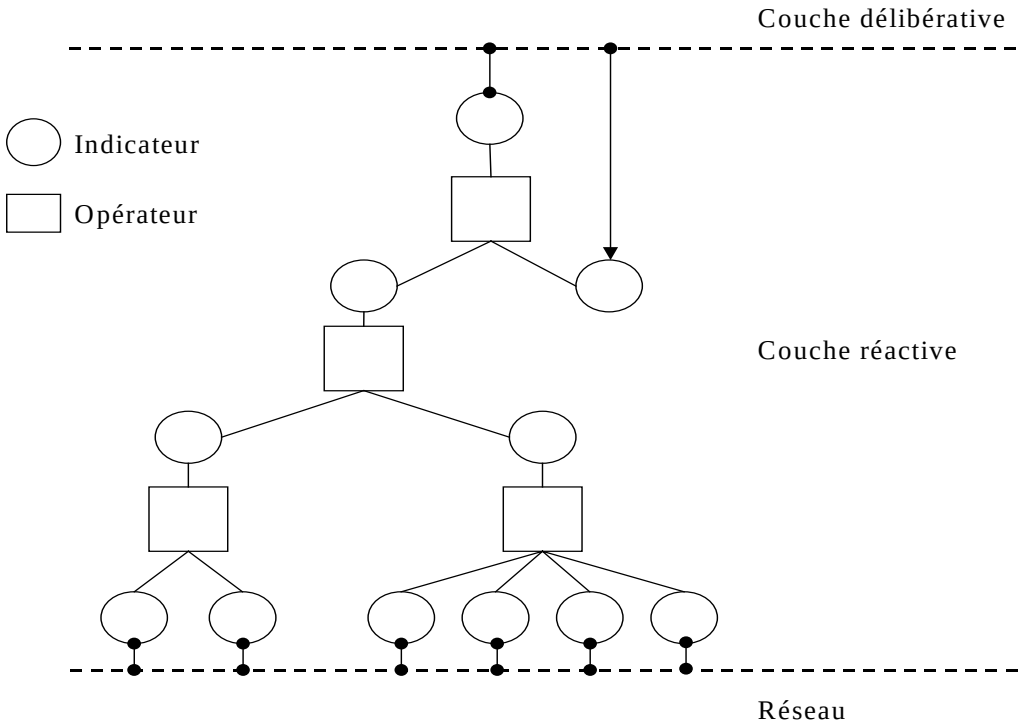


figure 20 Construction d'indicateurs

D'un autre côté nous avons considéré qu'un indicateur pourrait utiliser la description des objets de MIBs standards, l'agent (couche délibérative) devant dans ce cas faire la conversion entre la MIB standard et les MIBs spécifiques en fonction de l'équipement visé.

IV.3.3 Réacteurs

Jusqu'à présent nous avons vu avec les indicateurs comment il était possible de décrire une situation ou un état. Le réacteur va, lui, permettre de définir les conditions dans lesquelles un changement d'état va provoquer une réaction.
Un réacteur décrit une relation de cause à effet:

Si conditions alors actions

que la couche réactive va utiliser pour développer la séquence *(A)*.

Les conditions sont exprimées par des tests de comparaison sur des indicateurs, c'est-à-dire par des relations entre des indicateurs et des opérateurs de comparaison. Les actions, de leur côté, concernent soit le réseau (ex: changement de configuration), soit la couche réactive (ex: prise en compte d'une nouvelle valeur pour un indicateur), soit la couche délibérative (ex: création d'une croyance).

Par l'intermédiaire des indicateurs (actions sur la couche réactive) les réacteurs peuvent être liés entre eux. Il est donc possible de créer des réactions complexes pour faire, par exemple, de la corrélation d'alarmes, ou du diagnostic d'événements.

IV.3.4 Architecture de la couche réactive

La couche réactive est reliée à la couche délibérative au travers d'une interface identique à l'interface utilisée pour la connexion des modules de compétences. Du coté réseau, c'est une sous-couche d'instrumentation qui établit la relation entre les indicateurs et les éléments du réseau à gérer. La sous-couche instrumentation traduit aussi les actions du réacteur en commandes de gestion compréhensibles par les agents de gestion (SNMP, CMIP, etc) situés sur les équipements. Elle utilise pour cela les protocoles correspondants.

La configuration idéale est celle où l'agent est directement situé sur l'équipement, la sous-couche instrumentation agissant alors directement sur celui-ci.

La figure 21 montre un réacteur qui utilise deux indicateurs, l'un étant le résultat d'opérations sur des valeurs provenant du réseau, et l'autre étant renseigné par la couche délibérative. L'action provoquée lorsque les conditions sont remplies est ici la création d'une croyance qui sera utilisée par la couche délibérative (symbolisée par une flèche vers l'interface).

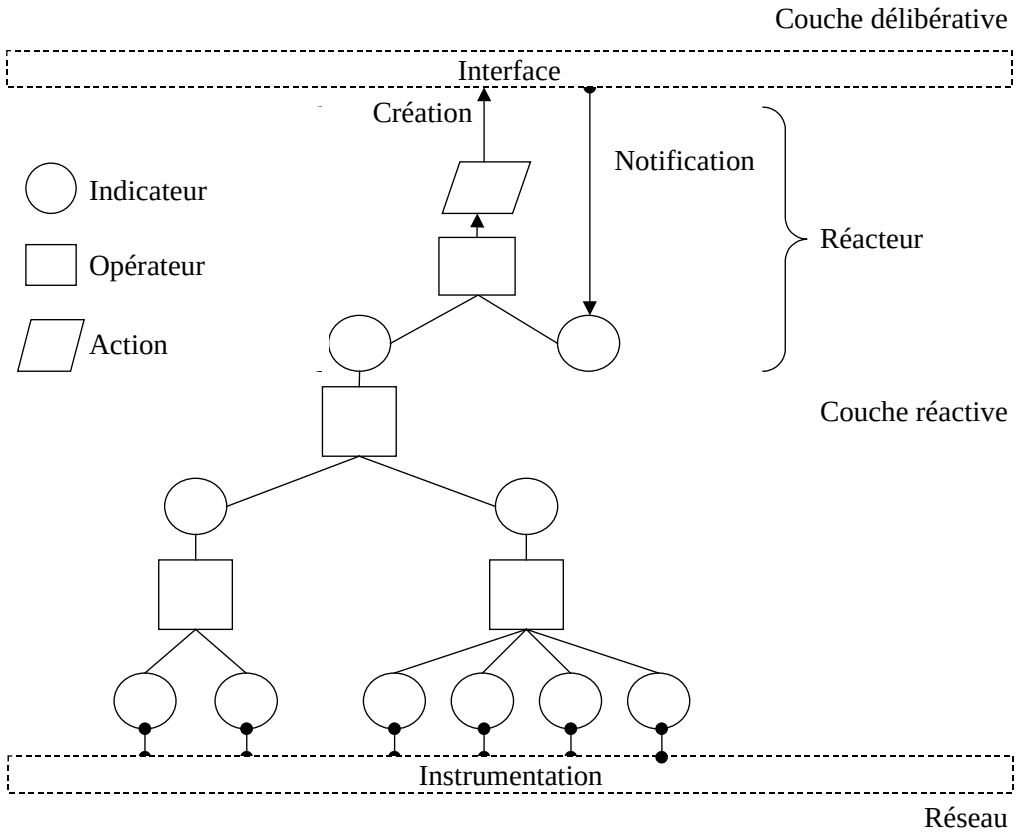


figure 21 Composants de la couche réactive

IV.3.5 Fonctionnement de la couche réactive

Afin de faciliter la programmation de la couche réactive, nous avons conçu un langage de description des réacteurs et indicateurs (voir annexe). Un réacteur est vu par la couche délibérative comme une croyance qui est affectée à la couche réactive. Le contenu de cette croyance décrit le réacteur comme un ensemble d'indicateurs, d'opérateurs, d'actions et éventuellement d'autres réacteurs, avec leurs relations.
La couche délibérative transmet les demandes de création et de suppression, des réacteurs et des indicateurs, en utilisant le mécanisme de notification fourni par l'interface (voir chapitre IV.2.4.3).

Ces notifications sont reçues par l'analyseur (parser) qui va identifier les différentes parties du réacteur et les transmettre au manager (voir figure 22) en charge de la gestion de la couche réactive.

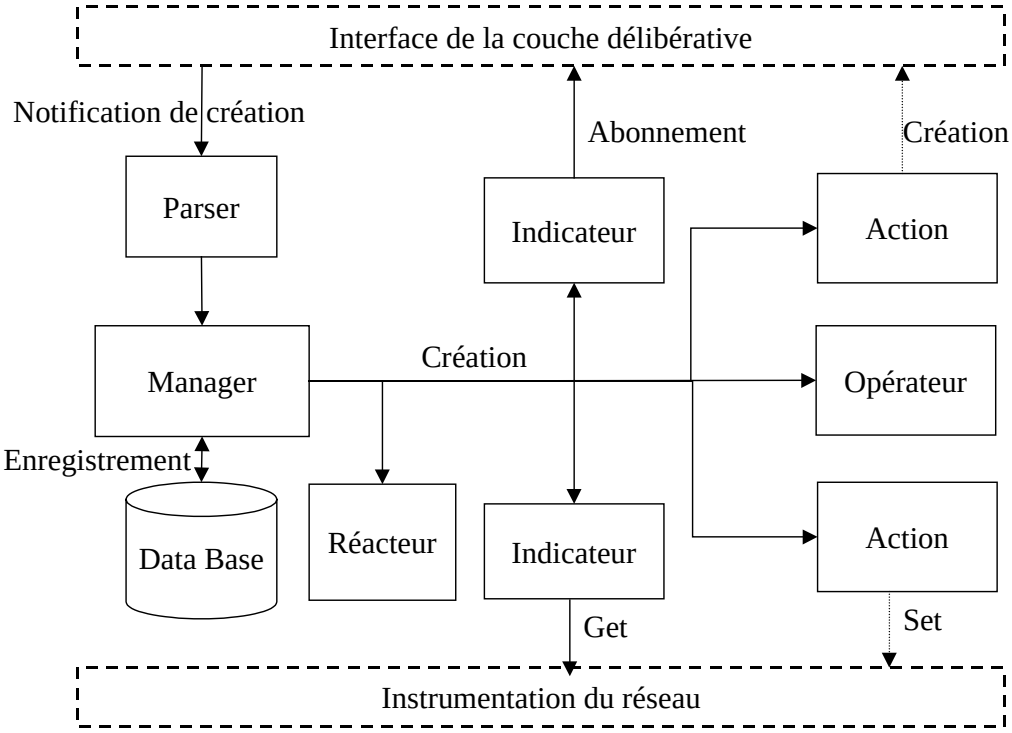


figure 22 Traitement de la demande d'instanciation d'un réacteur

Au cours de la phase d'analyse de la description du réacteur, la couche réactive va devoir résoudre les indéterminés s'il y en a. Ces indéterminés peuvent concerner les éléments suivants:

- ▷ référence d'un indicateur qui n'est pas décrit: dans ce cas le manager vérifie qu'il connaît bien l'indicateur et le précise à l'analyseur
- ▷ OID d'un objet à utiliser: le manager doit vérifier auprès de la couche délibérative de l'agent si cette OID est valable pour le type d'équipement visé, et demander l'OID spécifique si l'équipement en question n'utilise pas la MIB standard
- ▷ adresse d'un équipement: le manager doit interroger la couche délibérative pour obtenir l'adresse d'un équipement
- ▷ valeur d'un indicateur: lors de la conception d'un réacteur, certaines valeurs, comme des seuils par exemple, peuvent être remplacées par des noms logiques qui sont censés être connus de l'agent. Le manager interroge la couche délibérative pour récupérer l'instance de cette valeur.

Si tous ces indéterminés ont été résolus, le manager crée les indicateurs et le réacteur correspondant, et celui-ci devient actif.

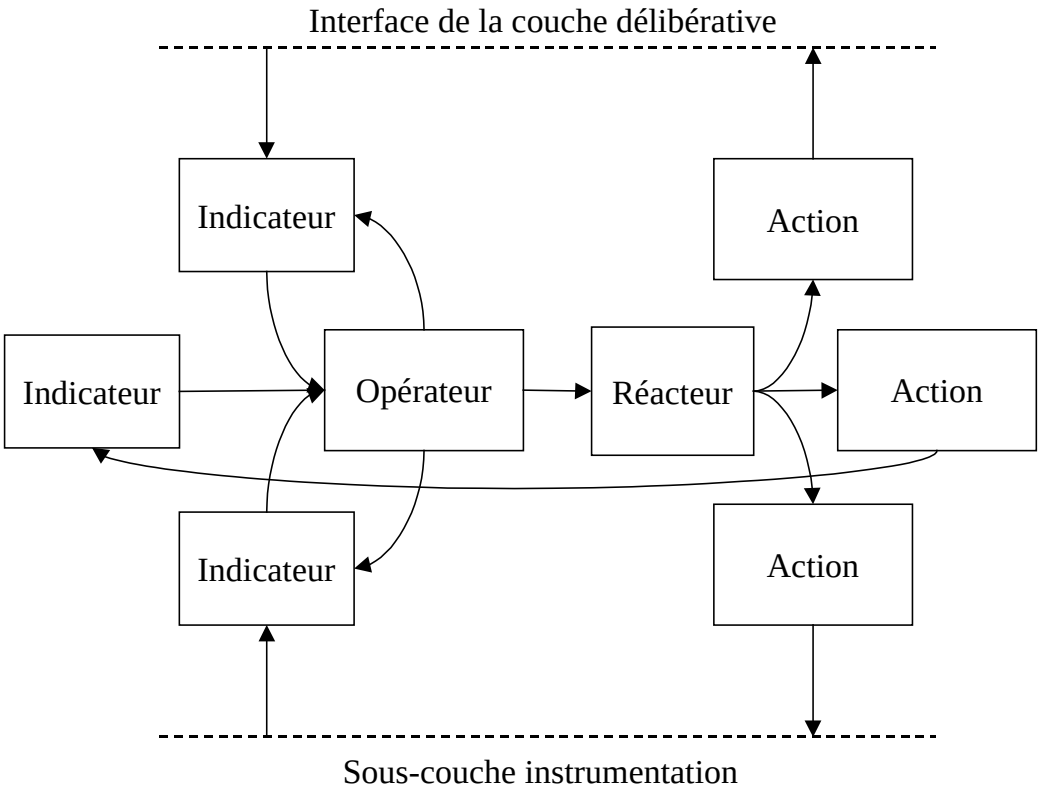


figure 23 Propagation des événements dans la couche réactive

La couche réactive travaille aussi sous un mode événementiel, ainsi lorsqu'un indicateur a sa valeur modifiée, il transmet aux opérateurs une notification de changement de valeur, et ceux-ci refont leurs calculs ou comparaisons et propagent si nécessaire les résultats (voir figure 23).

IV.3.6 Interface de programmation

Pour permettre à l'administrateur de réseaux de programmer facilement un réacteur, nous avons développé une interface graphique présentée ci-dessous.

Sur cette figure l'utilisateur compose un indicateur. Pour cela il peut sélectionner dans une MIB (1) ou spécifier un nom de symbole qui sera traduit par l'agent lorsque l'indicateur sera instancié par la couche réactive. Il dispose d'une liste d'opérateurs (3) et peut choisir d'intégrer des indicateurs préalablement créés en les sélectionnant dans la liste des indicateurs (4) pour composer un nouvel indicateur.
A chaque sélection, les éléments sont rajoutés dans la fenêtre graphique (5) où l'utilisateur va pouvoir les relier entre eux.

Il a également la possibilité de modifier la fréquence à laquelle la valeur des variables de MIB sont mises à jour et le nom des machines destinées à être gérées, ou un nom

symbolique représentant par exemple un type d'équipement, ou un domaine de gestion connu de l'agent qui utilisera l'indicateur.

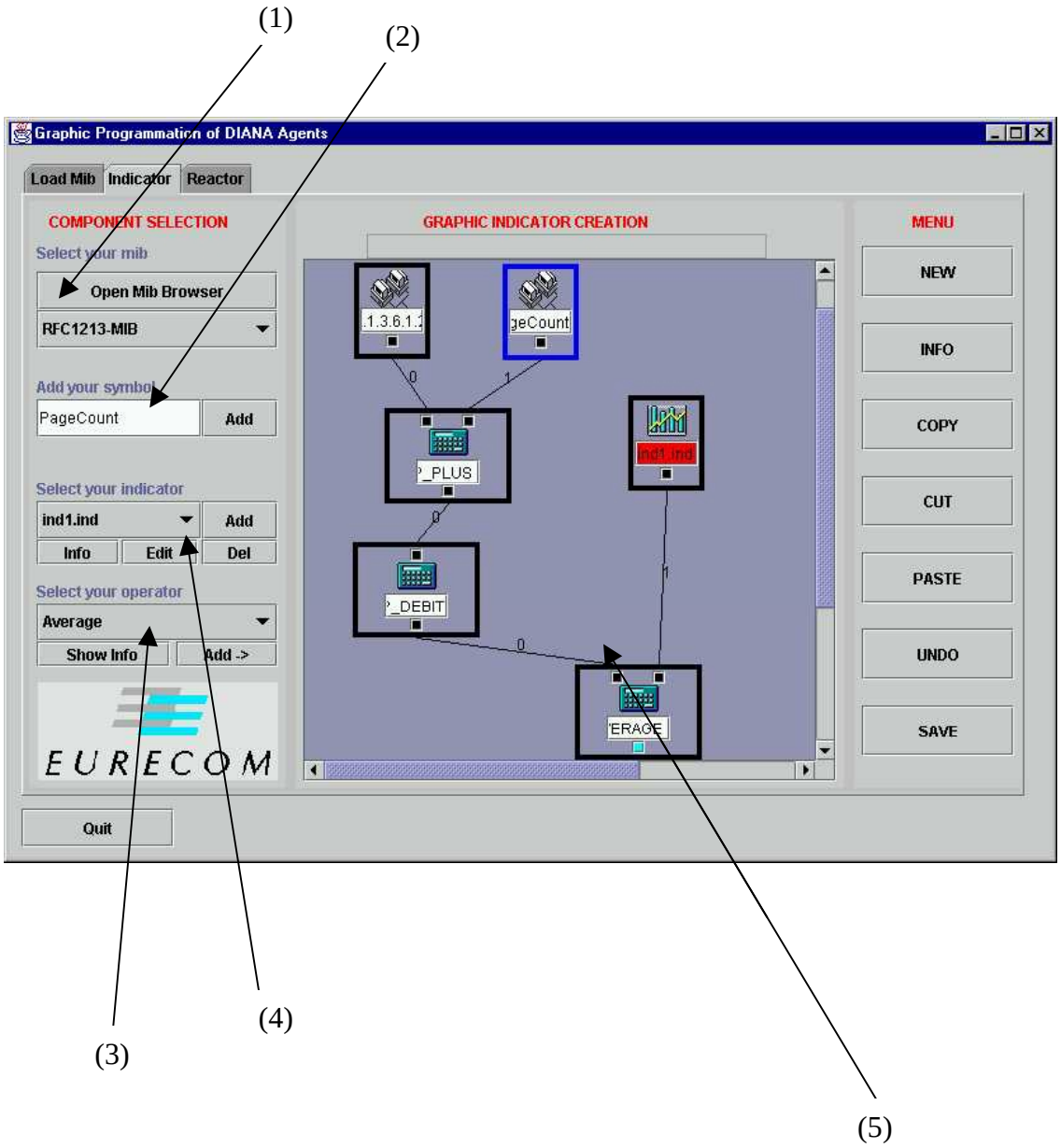


figure 24 Interface de programmation des réacteurs

Une fois composé, l'indicateur est traduit sous une forme textuelle, en utilisant le langage de description qui a été défini, pendant l'opération de sauvegarde.

IV.3.7 Conclusion

La couche réactive est perçue par la couche délibérative exactement comme un module de compétences. En utilisant l'interface, la couche réactive participe à l'instrumentation des croyances concernant l'état du réseau, et en optimisant les requêtes elle permet de diminuer le trafic engendré par la recherche d'information vers les équipements.
Grâce au langage de description utilisé par la couche réactive, il est possible de créer des indicateurs et réacteurs génériques pour un type d'équipement, en utilisant par exemple les MIBs standards ou bien des noms logiques qui seront traduits par l'agent en fonction de ses propres connaissances.
La réactivité des agents peut s'étendre à une réactivité du système multi-agents, car les agents peuvent s'échanger des informations sur la valeur de leurs indicateurs, grâce au mécanisme d'abonnement qui est accessible à la couche réactive. Néanmoins ces échanges diminuent la réactivité de cette couche, et ne doivent donc être utilisés qu'exceptionnellement.
Pour répondre au troisième objectif de la couche réactive qui est d'offrir à l'administrateur de réseaux un moyen de programmer des comportements réactifs pour les agents, une interface graphique de description de réacteurs a été développée.
Ce dernier point est particulièrement important, car contrairement à la couche délibérative qui demande à ce que des modules de compétences soient développés, la couche réactive n'impose aucune manipulation de code à l'administrateur de réseaux. Malgré tout, les réacteurs sont destinés à la programmation d'automatismes simples, et ne démontrent pas en cela l'intérêt de la technologie agent intelligent.
Nous avons donc concentré notre travail sur l'utilisation de la couche délibérative, pour faire ressortir les véritables intérêts de la technologie agent, comme nous allons le voir dans les chapitres suivants.

V Développement de SMA et méthodologie

V.1 Introduction

Une fois décrite l'architecture des agents DIANA, se pose le problème de son utilisation, c'est-à-dire du développement d'un SMA.
Une technologie agent est censée apporter une autre vision du logiciel, des avantages en termes d'utilisation grâce aux propriétés intrinsèques des agents intelligents; mais existe-t-il un apport quelconque en terme de développement?
Dans ce chapitre nous traitons le problème de la conception d'un SMA basé sur l'architecture DIANA et nous proposons une réflexion sur les principaux avantages et inconvénients du paradigme d'agent intelligent pour le développement de logiciel.
Il faut tout d'abord remarquer que très peu d'études ont été faites sur les méthodologies de développement orientées agent. L'essentiel de ces études porte avant tout sur des extensions de méthodologies existantes pour le développement orienté objet, ou provenant du domaine de l'IA [IGG98].
[GF99] fait le point sur ces études et sur les problèmes qui sont soulevés lorsque l'on s'intéresse au développement de systèmes multi-agents. Les auteurs remarquent très justement que les industriels soucieux de la rentabilité des investissements accordés pour le développement de leurs logiciels, veulent avant tout connaître le facteur de réutilisabilité de ces logiciels agent, facteur qui leur permet de réduire les coûts de développement. Il en est de même pour la validation du fonctionnement de tels systèmes, validation qui elle, conditionne les coûts de maintenance.
Les craintes ainsi exprimées sont parfaitement fondées. Il faut se rappeler les efforts consentis par les industriels pour mettre en place l'approche objet lorsque celle-ci a promis de rendre le code développé réutilisable, ou la déception ressentie par tous les acteurs de l'intelligence artificielle quand celle-ci a failli à ses promesses. Si d'un côté le

coût de la ligne de code a justifié l'investissement nécessaire au passage de la programmation structurée à la programmation objet, de l'autre la complexité et l'absence de preuve des systèmes d'IA ont bloqué le développement de projets industriels importants.
Maintenant, en l'absence d'une technologie agent reconnue par tous, certains peuvent se demander s'il n'est pas prématuré de se poser des questions sur les avantages et inconvénients du développement agent, ou bien sur les méthodologies à utiliser telles que celles que nous proposons dans ce chapitre. Cette question est aussi soulevée dans [IGG98], où les auteurs soutiennent en conclusion de leur étude, qu'un niveau conceptuel commun aux différentes approches agent est suffisamment stable pour envisager une ébauche de méthodologie.
De notre côté, pour répondre à cette remarque, nous pouvons argumenter que l'objectif de cette thèse est avant tout de faire le point sur l'approche agent dans le cadre de la gestion de réseaux, afin d'évaluer le potentiel de cette approche. C'est pourquoi nous ne prétendons pas ici proposer une méthodologie définitive, mais plutôt une démarche qui nous semble originale et appropriée.
Dans le même esprit [WJ98] propose une analyse critique des pièges que pose le développement orienté agent dans l'objectif de faire comprendre comment passer des études et prototypes en laboratoire à l'ingénierie dans les entreprises.

Dans une première partie nous montrons qu'une telle étude ne peut être complètement dissociée du système multi-agents sous-jacent, puis après avoir fait ressortir la distinction entre l'approche objet et l'approche agent dans le développement d'applications, nous proposerons une méthodologie de développement.

V.2 Modèles et domaines d'application

Après avoir développé notre architecture d'agent générique, une des premières questions que nous avons dû aborder pour traiter nos études de cas, a été la question du choix de la méthodologie à adopter.
D'après [WJK99] il n'existe actuellement aucune méthodologie générale adaptée au développement orienté agent. Leur argument, que nous partageons entièrement, est que l'approche agent introduit des concepts de comportement et d'autonomie qui sont impossibles ou tout au moins très difficiles à capturer par les méthodologies existantes, y compris les méthodologies objet. Dans ces dernières viennent aussi se greffer les risques de confusion entre les différents sens qui sont donnés aux mêmes mots comme *rôle* ou *comportement*.

De plus, du fait même de la diversité des technologies agent, il est impossible de prétendre proposer une méthodologie unique pour le développement orienté agent. C'est pour cette raison que la méthodologie que nous introduisons dans ce chapitre n'a pas une vocation universelle, même si elle serait applicable à d'autres technologies agent en cours d'élaboration. C'est aussi pour cette raison qu'il n'existe encore que très peu d'articles traitant de méthodologies de développement orientées agent.

[WJK99] nous propose une méthodologie basée sur les systèmes ADEPT et ARCHON [CJ96] pour développer des applications ayant certaines restrictions comme :

- l'impossibilité de prendre en compte d'éventuels conflits entre agents ou objectifs
- un nombre maximum d'agents relativement faible (inférieur à 100).

Parmi les quelques publications concernant des propositions de méthodologies de développement de SMA, [CN99] propose une approche pragmatique basée sur leur expérience engrangée lors de développements utilisant leur outil ZEUS. Si leur méthodologie utilise, tout comme celle présentée par [WJK99], un découpage de l'application par identification des rôles, les auteurs laissent par contre ouvert le choix de la méthode à utiliser pour analyser le système.

Nous trouvons aussi [GF99] qui propose une méthodologie volontairement restreinte pour pouvoir s'intégrer avec des méthodologies centrées sur des modèles spécifiques d'agents. La seule contrainte étant alors pour ces modèles de supporter une implémentation de l'approche « agent-groupe-rôle ». Bien qu'intéressante, cette méthodologie sous-entend qu'un groupe est constitué d'un ensemble d'agents ayant le même rôle alors que dans notre modèle un groupe ou domaine peut être un ensemble d'agents collaborant à un objectif commun.

[IGG99] présente un tour d'horizon assez exhaustif des études sur les méthodologies applicables ou conçues pour le développement agent, dans lequel sont citées aussi bien les méthodologies issues de l'approche objet que celles provenant de l'IA. Les auteurs concluent leur étude en remarquant que si actuellement le manque de standards dans le développement agent est une des raisons de l'absence de méthodologies dans ce domaine, un niveau conceptuel pour l'analyse des SMA décrivant les:

- ▷ modèles d'agents: les caractéristiques de chaque agent doivent être décrites incluant les compétences, les capacités de raisonnement, et les tâches;
- ▷ modèles de sociétés: les relations de dépendance et les interactions entre les agents

peut être défini indépendamment des architectures agent sous-jacentes. Nous sommes assez sceptiques à ce propos, ne serait-ce que parce que nos agents étant génériques et pouvant endosser plusieurs rôles, il est inapproprié de définir des modèles d'agents comme il est suggéré par les auteurs.

Grâce à cet aperçu nous remarquons à la fois l'importance de la terminologie dans un domaine encore flou, et la difficulté voire l'impossibilité de faire abstraction des architectures sous-jacentes aux technologies pour établir une méthodologie utilisable.
En raison du caractère encore exploratoire de la technologie agent, il serait non seulement prétentieux de proposer une méthodologie formelle, mais de plus celle-ci serait vaine, car il faudrait revoir ce formalisme en fonction de l'évolution des concepts qui sont au fur et

à mesure intégrés dans l'approche agent. C'est pourquoi la méthodologie que nous proposons dans ce chapitre est avant tout une ébauche de procédure, dont la formalisation ne pourra être entreprise qu'une fois les concepts renforcés et acceptés par l'ensemble de la communauté.
Pour les raisons évoquées dans cette section, cette méthodologie se fonde bien évidemment avant tout sur les concepts implémentés dans l'architecture générique DIANA, et a pour objectif de permettre le développement d'applications distribuées utilisables par un SMA DIANA.
Cette précision est importante pour deux raisons. La première est que certaines approches agent imposent qu'à un agent corresponde un rôle ou une entité fonctionnelle du système, ce qui fige en cela l'application dans la structure (ou l'organisation des agents). Avec l'approche DIANA, l'organisation des agents est faite dynamiquement, tout comme dans une entreprise l'organisation est faite en fonction des besoins de celle-ci. L'entreprise étant alors le système ou l'application. Il est donc possible de changer les affectations des rôles, en fonction des besoins comme nous le verrons plus tard. Enfin la deuxième raison est que nous ne proposons pas ici une méthodologie pour développer un système multi-agents particulier et que l'on pourrait donc coder avec n'importe quelle technologie, mais bien pour développer une application sur un SMA DIANA. Il faut noter en effet que des partisans de méthodologies comme celle introduite avec UML, prétendent que celles-ci sont indépendantes de la technologie sous-jacente. Ces prétentions sont l'objet de discussions acharnées sur les mailinglists spécialisées, aussi notre prétention est ici modeste, et nous nous focalisons avant tout sur l'utilisation de la technologie des agents DIANA.

V.3 Approche objet vs approche agent

Si nous faisons le tour des méthodologies existantes dans le monde du développement logiciel, il est immédiatement évident que les seules méthodologies qui ont su s'imposer sont les méthodologies de développement objet, notamment grâce aux techniques de modélisation comme OMT (Object Modeling Technique), et plus récemment UML (Unified Modeling Language) qui semble faire l'unanimité.
Mais pour savoir si ces méthodologies objet sont applicables dans le domaine du développement de SMA, il faut avant tout vérifier la compatibilité des concepts et faire ressortir les différences entre l'approche objet et l'approche agent. Pour cela nous pouvons comparer les notions de comportement, le comportement étant une notion décrite et utilisée par les deux approches.

La notion de comportement dans le processus de développement du logiciel n'est pas une découverte induite par l'approche agent, ni par l'approche objet, mais une réalité que les concepteurs ont toujours prise en compte. Avant l'apparition des langages objet, le comportement était avant tout le comportement des fonctionnalités du logiciel. Le cahier des charges puis les spécifications fonctionnelles déterminaient les comportements que devaient avoir l'application, et tout comportement anormal était considéré comme un «bug ».

Bien que n'ayant pas supprimé la présence de bugs, l'AO (approche objet) a démontré son efficacité en simplifiant leur correction, diminuant ainsi le coût de maintenance des applications.
Toujours en conception objet, le *comportement* de l'application est décrit au moyen des modèles fonctionnels, dynamiques, et statiques. Ceux-ci indiquent respectivement ce qui se passe, quand cela se passe, sur quoi cela se passe.
Lors de la conception d'une application, les classes d'objets sont tout d'abord identifiées, puis sont établies les associations entre ces classes pour déterminer la structure statique de l'application, déterminant ainsi des rôles attribués aux tuples (classe, association, classe).
L'OMT définit un rôle comme étant "*l'identification d'une extrémité de l'association de deux classes"*, et précise que lorsqu'il n'y a qu'une association entre deux classes distinctes, le nom des classes sert souvent de nom de rôle [RBE95].
Nous voyons donc que dans le monde objet, le rôle décrit soit une classe d'objet (ex : client), soit une relation ordonnée entre deux classes (ex : "responsable clientèle" entre la classe employée et elle-même).
Alors que dans le monde agent le rôle est associé à la notion de comportement, au travers des capacités et des compétences, dans le monde objet le rôle et le comportement sont fortement dissociés, toute interaction entre objets étant considérée comme un comportement.

Le comportement global est vérifié grâce aux cas d'utilisation ("use cases" en anglais) qui vont se traduire par des diagrammes de séquences (temporel), de collaboration (spatial) ou par des diagrammes d'états-transitions, et d'activités si l'on utilise la notation UML.
D'une manière pratique, dans l'approche objet, le comportement est décrit d'une manière textuelle lors de la rédaction du cahier des charges et des documents de conception, puis ceux-ci sont utilisés pour décrire les scénarios de tests validant le «bon» comportement du système. Enfin lors du développement, les interactions (et donc les comportements) sont figées à la compilation et lors de l'édition des liens du code produit. Ceci est dû au mécanisme d'échange de messages utilisé par l'AO. Si dans l'AO les objets communiquent en s'échangeant des messages tout comme dans l'AA (Approche Agent), ces messages sont de nature différente et utilisés différemment. Dans l'AO, les messages sont des données typées transmises à l'occasion de l'invocation de méthodes, tandis que dans l'AA, les messages contiennent des informations quelconques, et sont échangés en utilisant des protocoles plus ou moins complexes (ex: KQML). Dans l'AO, la méthode invoquée d'un objet destinataire du message est obligatoirement exécutée, tandis qu'en AA, les messages sont mis à la disposition de l'agent qui va les traiter de manière différente en fonction de leur contenu et de ses propres objectifs (voir chapitre IV.2.4.6).

L'utilisation de l'approche agent amène donc un niveau d'abstraction supplémentaire par rapport à l'AO, les comportements pouvant être soit intégrés pendant la phase de conception du SMA, soit déduits par le système lui-même pendant la phase opérationnelle.

Une autre différence entre ces approches est que si l'approche objet permet de décrire ce qui se passe « à l'intérieur », l'approche agent décrit ce qui se passe à l'extérieur. Par exemple pour une application qui doit s'occuper de l'éclairage d'une pièce, on pourra

avoir un objet interrupteur dont une méthode sera "basculer la position des contacteurs". A l'opposé un agent aura une compétence qui lui permettra d'agir pour déterminer les actions à mettre en place pour que la pièce soit éclairée, comme par exemple utiliser la méthode "basculer de l'objet interrupteur" ou "changer l'ampoule" s'il s'aperçoit que celle-ci est défectueuse.
Enfin il est naturel de penser aux comportements d'un objet en fonction de son environnement, cela indépendamment de l'objet lui-même. C'est-à-dire sans que ce comportement soit nécessairement intégré ou pris en compte pendant la conception de sa classe.
Par exemple, le comportement d'une voiture sur route mouillée n'est pas uniquement lié à la voiture elle-même mais à un ensemble de paramètres comme l'état des suspensions, l'état des pneus, la nature du revêtement de la route. Aucun de ces objets, bien qu'ils participent tous au comportement global de la voiture sur la route mouillée, ne peut être associé directement à ce comportement.
A l'opposé l'agent conducteur pourra apprendre à utiliser l'objet voiture, en observant son comportement, et en agissant sur les objets associés. En effet la conception agent sous-entend que les objets ne sont visibles aux agents qu'au travers de leurs sens (sensors), et sont utilisables grâce à leurs actionneurs (effectors).
Le comportement de l'agent, du fait de l'utilisation de ses facultés d'apprentissage, ou de sa capacité à construire dynamiquement des plans d'actions, n'est définitivement pas modélisable par les méthodes objet.

V.4 Principes et définitions

La modélisation des SMA et la définition d'une méthodologie agent nécessitent comme nous l'avons vu, de préciser les termes utilisés. Si certaines définitions ont déjà permis de clarifier l'architecture DIANA, la conception d'une application introduit d'autres niveaux d'abstraction que nous précisons ci-après.

Termes et abstraction directement traduisibles et utilisables dans l'architecture DIANA:

Situation
Une *situation* est un ensemble d'informations (croyances) connues de l'agent à un instant *t*. Une situation peut être réduite à l'ensemble des croyances utilisables par une compétence particulière de l'agent. C'est-à-dire que la situation correspond à une vision plus ou moins large des croyances de l'agent à un instant donné.

Objectif
L'objectif est concrétisé soit par une ou des croyances produites comme résultat de l'activation d'une opération, soit par une croyance contenant une demande d'activation d'opération (voir chapitre IV.2.4.4)

Motivation

Une *motivation* est un objectif associé à une activité, c'est-à-dire à l'exécution d'une opération continue, par opposition à une opération ponctuelle.

Compétence
Une *compétence* est un ensemble de capacités et donc d'opérations, permettant à un agent d'assurer un rôle. La compétence est traduite par un module de compétences dans l'architecture DIANA.

Opération
C'est une entité algorithmique exécutable, appartenant à une compétence et connue de l'agent par sa description (voir chapitre architecture). Une *opération* permet de modifier une situation.

Tâche
Une *tâche* est l'instance d'une opération (voir activité).

Activité
L'*activité* d'un agent est l'ensemble de ses opérations dont le statut est «en cours », donc l'ensemble des tâches. Une activité est aussi une instance d'une seule opération lorsqu'elle désigne une tâche permanente.

Plan
Un *plan* est le résultat de la construction dynamique d'un comportement pour atteindre un but. Un plan est produit suite à la création d'un objectif portant sur un résultat.

Termes et abstractions utilisables pour l'analyse et le design du SMA

Organisation
Une *organisation* décrit une application agent intelligent par identification des rôles et des relations qui existent entre ces rôles.

Domaine
C'est une croyance particulière décrivant un ensemble dont chaque élément est concerné par un même objectif (ex: un domaine de commutateurs) ou participe à un objectif commun (ex: un domaine d'agents dans le cadre d'une coopération).

Responsable ou Maître
C'est un rôle particulier dans une organisation qu'endosse un agent qui doit diriger plusieurs autres agents.

Comportement
C'est l'ensemble des opérations qu'exécute l'agent face à une situation. Un *comportement* est un ensemble ordonné d'opérations, dont on connaît les causes et les effets en terme de croyances. Un comportement induit est un comportement qui découle de l'exécution d'un autre comportement.

Mission
Une mission représente un ensemble d'objectifs dont la réalisation est perçue par l'agent au travers de la production d'informations. Une *mission* décrit une tâche

Le rôle
Le rôle est l'identification d'un ensemble de missions. Un *rôle* se traduit par des compétences en terme d'implémentation agent.

En fonction des définitions ci-dessus, et de l'architecture de l'agent que nous avons créée, nous pouvons définir certaines propriétés et relations entre les différents concepts utilisés.

Posons :

Soit les ensembles suivants

$C = \{ c_1, ..., c_n\}$ capacités (compétences)
$O = \{ o_1,..,o_n\}$ opérations
$B = \{b_1,...bm\}$ buts (croyances produites comme résultats)
$P = \{ p1,...,pn\}$ préconditions
$U = \{u1,..,un\}$ croyances utilisées en cours d'opération
$Q = \{q1,...,qs\}$ croyances (intermédiaires ou transitoires) produites en cours d'opération
S l'ensemble des croyances connues de C
$S = B \cup P \cup U \cup Q$

et soit les opérations suivantes

$\mathbf{P}(o)$ = ensemble des croyances préconditions pour l'opération o
$\mathbf{S}(o)$ = {inconnu, prêt, en cours, suspendu, terminé}statut de l'opération o
$\mathbf{R}(o)$ = croyances résultats de l'opération o

Règles de dépendances utilisées par l'agent:

1) entre opérations O et compétences C:
   $\forall o_i \in O, \exists c_x \in C / o_i \in c_x$
   Attention! O est une ensemble non nécessairement déterminé, car o_i peut être un modèle dont certains éléments ne sont connus qu'à l'exécution (voir chapitre IV.2.4.1)
2) entre buts et opérations:
   $\forall b_i \in B \exists o_j \in O / \mathbf{R}(o_{j)} = b_i$
3) entre buts et préconditions:
   si pour bi il existe oi / P(oi)<>{0} alors il existe {oj,..ok} / {R(oj),…R(ok)} = P(oi)

Vérification de la possibilité d'atteindre un but

L'agent utilise l'algorithme suivant pour vérifier qu'un but ou objectif est atteignable :

Soit bi le but à atteindre, E l'ensemble des croyances de l'agent, A(bi) la fonction recursive de recherche se définit comme suit :

$$A(b_i) = \exists\, o_i \in O \,/\, \mathbf{R}(o_i) = b_i \text{ et}$$
$$\forall\, \alpha_i \in \mathbf{P}(o_i)\,,\, \alpha_i \in E \mid A(\alpha_i)$$

V.5 Méthodologie

V.5.1 Analyse et design

Le processus de conception passe par la phase d'analyse du système à développer, pendant laquelle il faut décomposer ce système à l'aide de modèles. Le premier niveau d'analyse permet de modéliser les rôles, puis de modéliser les relations entre ces rôles, c'est-à-dire l'organisation.
Nous rappelons ici que l'abstraction et l'organisation sont avec la modularité, la distribution et l'intelligence, des principes reconnus comme permettant de maîtriser la complexité d'un système. Les rôles, comme nous le verrons par la suite, présentent un premier niveau d'abstraction, celui-ci étant par la suite décomposé en compétences (voir figure 25).

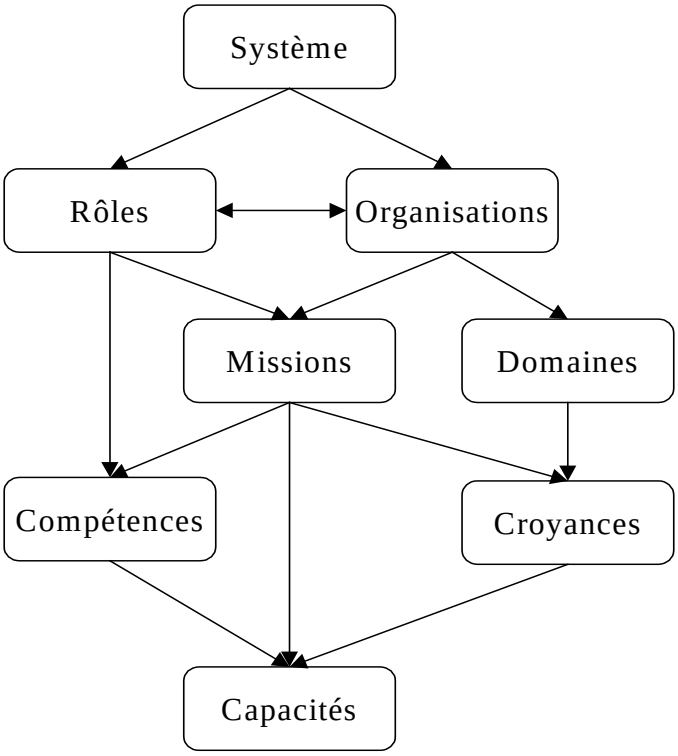


figure 25 Modélisation d'un système multi-agents

Dans un premier temps le système est décomposé en **rôles** et en **structure organisationnelle**.

Les rôles sont des abstractions qui définissent des fonctions (ex: président, technicien de surface) dont l'exercice nécessite des compétences (vision statique) et des missions (vision dynamique ou opérationnelle)

Il existe deux types d'organisation:

1. l'organisation des rôles qui définit les relations qui existent entre les rôles. Ces relations définissent des dépendances hiérarchiques ou fonctionnelles.
2. l'organisation des agents où l'on peut retrouver les notions de délégateur et de délégué, ainsi que les notions de coopération.

Une fois les rôles et les organisations définis, les cas d'études décrits par le cahier des charges vont être traduits sous forme de missions en précisant le cadre de l'utilisation des rôles, c'est-à-dire les compétences précises, les capacités et les domaines, relatifs à ces rôles. La description de missions va être traduite en tâches ou en activités, et en croyances, précisant ainsi les limites ou les champs d'utilisation des compétences.

Ces missions permettent de décrire la dynamique de la vie de l'organisation, sous forme d'enchaînements ou de dépendances actives. Par rapport à l'organisation qui définit des dépendances statiques.

Afin de clarifier notre présentation de cette méthodologie, nous prendrons un exemple simple pour illustrer notre démarche. Cet exemple est celui du développement d'une application de monitoring, qui est une activité essentielle des plates-formes de gestion de réseaux.

V.5.2 Identification des rôles

Dans [Wei99] le rôle est défini comme : "la part fonctionnelle ou sociale qui dans un agent appartenant à un système multi-agents, intervient dans le processus de résolution de problème, de planification, ou d'apprentissage". Pour l'auteur, les rôles sont décrits sous forme de permissions et responsabilités, et sont associés à des modèles d'interactions. Les rôles sont souvent vus comme étant définis à travers des lois sociales ou des stratégies qui permettent de représenter l'organisation des entités.

Dans le contexte de systèmes multi-agents, le rôle est associé à un agent contrairement au domaine réel où un rôle peut être une notion abstraite associée à un groupe, comme par exemple la famille, dont l'un des rôles est un rôle éducatif. Le groupe et le rôle son néanmoins étroitement reliés pour [FG98] qui présentent le rôle comme « une représentation abstraite d'une fonction ou d'un service délivré par un agent au sein d'un groupe ». Leur approche les amène à considérer un rôle particulier celui de « gestionnaire de groupe » qui est obligatoirement attribué au créateur du groupe. Plus généralement un

rôle va être *joué* ou *pris en charge* par une seule entité, mais sous la responsabilité d'une autre entité (agent ou humain).

Un rôle va pouvoir être précisé par des missions, explicites ou implicites. Pour remplir les missions correspondant à son rôle, un agent va avoir besoin de ressources (moyens), de compétences (savoir), de pouvoir (autorité).
La mission va permettre de préciser les compétences liées et nécessaires à l'exécution des rôles, ainsi que les relations avec la société d'agents.

Dans notre approche, l'identification des rôles se fait en répondant aux questions suivantes:

1. Quelles fonctions distinctes peuvent être identifiées?
2. Est-ce que ces rôles peuvent être divisés en sous-rôles ou complétés?
3. Est-ce que ces rôles peuvent être délégués?

La réponse à la première question doit se faire en considérant d'un côté les compétences supposées nécessaires à la réalisation du rôle, compétences qui doivent pouvoir être prises en compte par un seul agent, et de l'autre côté la distribution possible ou souhaitable des rôles qui pourrait permettre une décomposition des rôles.

En utilisant l'exemple de l'application de monitoring des équipements d'un réseau, nous pouvons répondre à la première question, en identifiant dans un premier temps:

- ▷ un rôle d'interface utilisateur: pour l'affichage de l'état du réseau, et la prise en compte des contraintes de monitoring demandées par l'utilisateur
- ▷ un rôle de moniteur: pour formuler les requêtes vers les équipements et traiter les réponses
- ▷ un rôle de spécialiste équipement: pour transformer ces requêtes générales en requêtes spécifiques à l'équipement (ex: en fonction des protocoles utilisés par les équipements)

La deuxième question concerne essentiellement les rôles abstraits ou génériques comme le rôle de la famille, le rôle de l'entreprise, qui nécessitent une décomposition en fonction des objectifs de l'application. Cette décomposition n'entraîne pas forcément la suppression du rôle initialement choisi. C'est le cas par exemple où le rôle peut être caractérisé par une compétence particulière commune aux sous-rôles.
Les trois rôles de l'application de monitoring n'ont pas besoin d'être divisés, mais ils pourront être complétés par des rôles annexes, comme par exemple un rôle de gestionnaire d'historique.

La troisième question permet de prévoir les possibilités de délégation pour chaque rôle, c'est-à-dire l'organisation possible des agents en fonction de leurs rôles. C'est une question dont la réponse va influencer les capacités d'extensibilité de l'application, la rendant plus ou moins sensible au facteur d'échelle. Pour bien répondre à cette question il faut donc envisager le cas où l'application va être utilisée intensivement et extensivement.

Dans notre exemple, il faut donc se demander si notre application est utilisable dans le cas où le réseau est important, et de plus composé de plusieurs sous-réseaux.
Il est alors envisageable d'avoir un nouveau rôle : celui de responsable (master) chargé de la répartition du monitoring pour une partie du réseau, et qui va être en relation avec d'autres masters et des agents ayant le rôle de moniteurs.

Ce nouveau rôle peut sembler inutile car les agents DIANA peuvent facilement déléguer des rôles ou des activités, et un agent en charge du monitoring peut aussi demander à d'autres agents de prendre en charge ce rôle. L'intérêt de ce nouveau rôle est en fait relatif à l'organisation et au suivi des activités de monitoring.
En effet il semble judicieux d'avoir une entité responsable de la répartition des activités plutôt que de laisser à chaque agent le soin de se faire aider par d'autres.
Ce nouveau rôle correspond à un rôle d'organisateur, et s'il ne participe pas directement au monitoring, il peut par contre posséder des compétences particulières pour optimiser la répartition des activités (load balancing).
A la fin de cette étape nous avons donc identifié quatre rôles (voir figure 26):

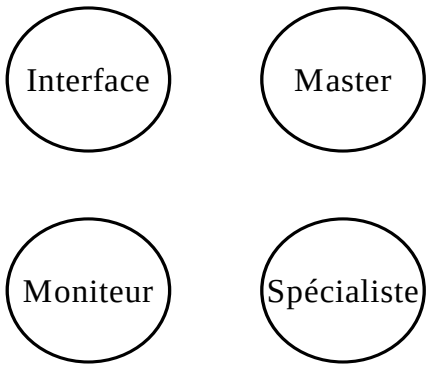


figure 26 Identification des rôles

V.5.3 Organisation

L'organisation définit les dépendances qui existent entre les rôles, c'est-à-dire qu'elle permet de déterminer les dépendances hiérarchiques ou fonctionnelles des rôles. Ces relations vont donc permettre de déterminer les délégateurs, les délégués, les coopérateurs, et préciser le type de délégation (ex: de un à plusieurs).
Une question qui doit être résolue lors de la détermination de l'organisation est la question des échanges, ou interactions, entre les rôles. Cette question doit être traitée en considérant que les rôles peuvent être joués ou exercés par des agents différents, et que ces échanges vont entraîner une consommation des ressources de communication, ainsi que certains délais relatifs au transfert des messages.
Les questions auxquelles il faut répondre pour déterminer l'organisation des rôles sont les suivantes:

1. Quelle est la dépendance d'un rôle?
2. Quelles sont les interactions entre les rôles?
3. Quelles sont les priorités à attribuer aux interactions?

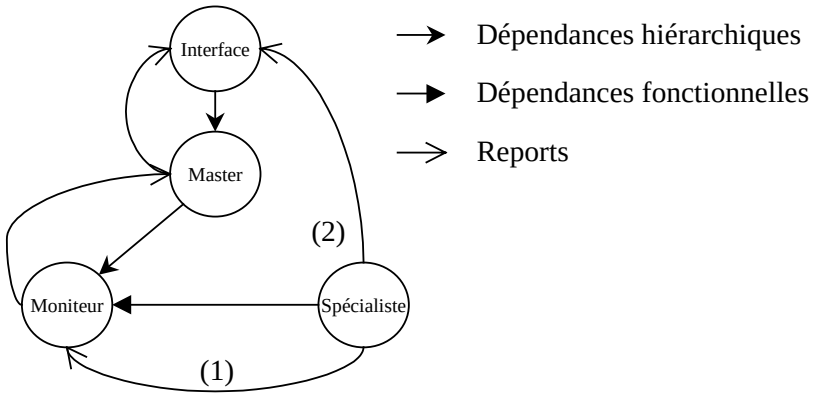


figure 27 Organisation des rôles

La dépendance des rôles permet de déterminer le flot de traitement des informations nécessaires à l'exécution d'une activité. Dans notre exemple, le rôle de master est dépendant de l'interface dans le sens où c'est à partir de l'interface que se fera la commande de mise en place du monitoring. Cette dépendance est *hiérarchique*, car le rôle de master ne nécessite pas d'interface pour être réalisé, tandis que l'interface nécessite l'existence d'un master. De même pour le moniteur: celui-ci peut démarrer une activité de monitoring sans l'intermédiaire du master, qui n'est là que pour gérer la répartition des activités de monitoring. Le moniteur est donc fonctionnellement indépendant du master, et le master va exercer un contrôle hiérarchique sur le moniteur. Il n'en est pas de même pour la relation entre le spécialiste et le moniteur, car c'est le moniteur qui transmet au spécialiste toutes les informations relatives aux activités de monitoring. Le moniteur ne peut donc monitorer sans l'aide du spécialiste, c'est une dépendance *fonctionnelle*.

Un autre type de relation peut être défini, il s'agit du *report* qui indique que l'exercice du rôle produit des informations qui sont utilisées par le destinataire.

Dans la figure 27, nous indiquons qu'il existe deux choix possibles d'interactions avec le rôle de spécialiste. Par défaut, si c'est le moniteur qui transmet des requêtes de monitoring à l'agent en charge du rôle de spécialiste local, c'est lui qui sera averti des résultats (1), mais il peut demander que les informations soient directement transmises à l'interface (2).
De plus le master peut lui-même gérer plusieurs masters, rendant l'application extensible.

La détermination de l'organisation des agents consiste à répondre à des contraintes du cahier des charges, ou à conseiller des organisations en fonction de critères particuliers comme des critères de performance ou d'économie de ressources.

En raison de la souplesse de l'architecture des agents DIANA, cette organisation peut être changée pendant la phase opérationnelle, comme nous le verrons dans les études de cas. Un des avantages de la réorganisation dynamique est de pouvoir fiabiliser les systèmes. A ce niveau, il faut vérifier que les rôles soient bien déterminés et cohérents pour qu'ils aient un maximum d'indépendance les uns par rapport aux autres.
La figure 28 montre une possibilité d'organiser les agents participant au système applicatif, et les relations de délégation qui existent entre les agents.
Un des éléments influençant le regroupement des rôles au sein d'un même agent concerne la quantité d'informations qui sont échangées entre les rôles, en tenant compte de l'utilisation réelle de ceux-ci. Un autre élément permettant d'organiser le regroupement des rôles est la qualité des connexions entre les agents. Ce dernier élément peut être inconnu pendant les phases de conception de l'application, d'où l'avantage de pouvoir déléguer des rôles en cours de fonctionnement de l'application.

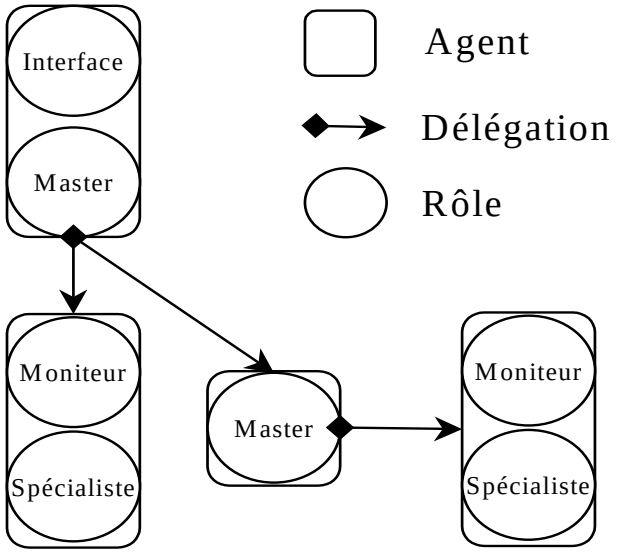


figure 28 Organisation des agents

V.5.4 Domaines

Il est maintenant possible de préciser des domaines d'attribution, domaines qui représentent des abstractions utilisées pour préciser les modèles de relation entre les rôles et les agents. Nous verrons par la suite que tout comme il existe deux types d'organisation, il existe deux types de domaine. Les domaines de travail, ou de mission, et les domaines d'agent. Ce sont ces derniers que nous discutons dans ce chapitre.

Lorsqu'un agent peut déléguer une tâche ou une activité à un rôle, alors il est utile que cette délégation se fasse par l'intermédiaire d'un domaine. Le domaine est en ce sens comparable à la notion d'agent-groupe-rôle proposée par [GF99]. Pour les auteurs, le groupe est lié à un rôle et chaque agent membre du groupe assure le rôle correspondant. La différence ici avec notre modèle est que ce ne sont pas les agents qui décident de faire

partie ou non d'un domaine, ceux-ci étant réquisitionnés par un agent ayant un rôle d'organisateur, ou par un responsable de la gestion du domaine. La responsabilité de la composition des domaines est naturellement donnée au gestionnaire de réseaux pendant le déploiement de l'application, puis transmise aux agents ayant un rôle d'organisateur.
C'est pourquoi les domaines sont à ce niveau de l'analyse des noms symboliques qui seront exploitables par les compétences développées autour des rôles, sous forme de croyances.
Les questions qui permettent de déterminer les domaines à prendre en compte pour le design de l'application sont les suivantes:

1. Est-ce que les délégations concernent des relations de 1 à 1 ou de 1à n ?
2. Est-ce que la gestion du domaine est prise en charge par un rôle ?

En répondant à ces questions, il est possible qu'apparaisse la nécessité de rajouter un rôle qui n'était pas prévu dans l'organisation. Il s'agit le plus souvent de rôles administratifs qui n'ont pas d'impact direct avec les fonctionnalités de l'application, mais qui ont un impact sur les propriétés du système comme par exemple la flexibilité de celui-ci.
L'architecture des agents DIANA n'offre que les services de gestion de croyances comme support pour la gestion de domaines, contrairement à d'autres approches qui sont orientées sur les domaines. Ce choix est justifié par la diversité des politiques de gestion de domaines qui peuvent exister en gestion de réseaux. Nous verrons par exemple dans la première étude de cas un exemple de gestion de domaines permettant de fiabiliser le monitoring, cette gestion n'étant pas applicable à tous les problèmes. Il est donc à notre avis impossible de proposer un rôle unique de gestionnaire de domaines, celui-ci devant être adapté à l'application, ou aux propriétés recherchées.

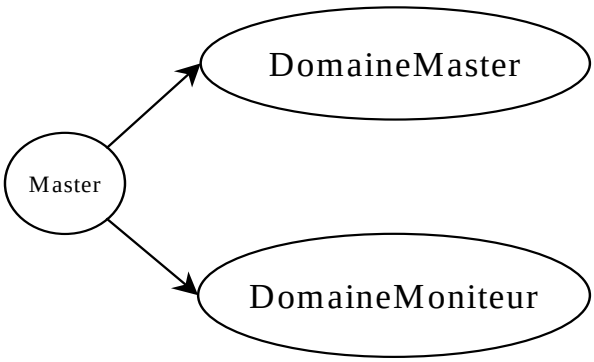


figure 29 Relations rôles-domaines

Dans notre exemple de monitoring deux domaines peuvent être créés, un pour les agents ayant le rôle de master, un pour les agents directement en charge du monitoring (figure 29).

V.5.5 Missions

Comme nous pouvons le constater, la définition des rôles et leur organisation reste d'un niveau d'abstraction relativement élevé. Pour préciser ces rôles et par la suite déterminer les compétences nécessaires à leur exercice, nous introduisons la notion de missions. Ces missions correspondent à une re-formulation des scénarios développés dans le cahier des charges, ou à la décomposition des fonctionnalités recherchées. Cette décomposition va spécifier les responsabilités attribuées à chaque rôle pour le traitement des informations, et préciser les interactions et relations inter-rôles.
Ces missions seront ensuite traduites sous forme de tâches, de croyances, précisant les limites ou les champs d'utilisation des compétences. Cette décomposition sous forme de tâches a déjà été étudiée par [SV98][Gla96] et donné lieu à des essais de formalisation [BTWW95]. Nous rappelons que les missions permettent de décrire la dynamique de la vie de l'organisation, sous forme d'enchaînements ou de dépendances actives, tandis que l'organisation des rôles définit les dépendances statiques.
Dans le cadre de leur méthodologie, [CN99] rapproche cette notion de mission de celle de responsabilité et de collaborateur. Directement incluse dans la description du rôle, la responsabilité décrit les différentes activités liées au rôle ainsi que les collaborateurs (autres rôles) qui participent aux activités. Pour les auteurs, à chaque rôle correspond un ensemble de responsabilités, elles-mêmes rattachées à des rôles, l'ensemble étant décrit dans des feuilles de rôle ou formulaires.
L'utilisation de formulaires de description de missions est compatible avec notre approche, un rôle étant exercé au moyen de compétences en fonction de missions.

Les missions sont déterminées par les réponses aux questions suivantes :

1. quelle est l'implication de chaque rôle dans le déroulement des scénarios précisés dans le cahier des charges, c'est-à-dire quelles sont les tâches relatives à l'exécution du rôle ?
2. quelles sont les informations (ex : croyances, domaines) nécessaires à l'accomplissement de la mission ?
3. qui les fournit ?
4. quelles sont les informations produites par la mission ?
5. à qui vont-elles servir ?
6. est-ce que les échanges sont optimisés ?

Dans l'exemple de l'application de monitoring que nous avons utilisé jusqu'à présent, et comme nous n'avons pas précisé de cahier des charges, nous pouvons imaginer qu'un scénario d'utilisation est :

> (1)L'utilisateur veut avoir en permanence un affichage qui permette de connaître les éléments hors service du réseau, et à la demande l'ensemble des éléments en fonctionnement avec la durée depuis leur mise en marche.

La première question nous oblige à nous intéresser à la répartition des activités entre les rôles. La détermination des éléments, actifs ou non, est raisonnablement une activité liée

au rôle du moniteur. La sélection du type d'affichage est bien une tâche concernant le rôle de l'interface.
La deuxième question concerne les interactions entre les rôles : est-ce le moniteur ou bien le master qui doit remonter les informations de monitoring en permanence à l'interface ? C'est-à-dire, est-ce que le moniteur doit avoir comme mission de filtrer des informations ?
Pour répondre à cette question il faut considérer d'un côté la logique de cette tâche, et de l'autre les interactions entre les rôles en fonction de l'organisation des agents.
Par exemple si nous considérons que cela fait partie du rôle de moniteur de filtrer les informations, car il possède à son niveau la connaissance de l'état des équipements, nous devons aussi considérer le cas où deux utilisateurs veulent afficher des informations différentes en même temps. Le moniteur n'est pas directement en relation avec l'interface, et c'est par l'intermédiaire du master que transitent les informations. Le master a donc la connaissance de l'origine des requêtes qu'il transmet aux moniteurs, et puisque plusieurs interfaces peuvent demander des informations de monitoring concernant le même équipement, il est plus intéressant que ce soit le master qui prenne en charge le filtrage.

Nous voyons ici que la détermination des missions soulève des problèmes relatifs à l'organisation, et dans certains cas peut amener à spécifier de nouveaux rôles.

Une fois les missions précisées, il est possible de penser aux détails comme les opérations et les croyances qui vont être utilisées.

V.5.6 Compétences

Les compétences sont le résultat de la décomposition d'un rôle en fonction des missions qui lui sont attribuées, comme par exemple le rôle de secrétaire peut être décomposé en compétence de gestion d'agenda ou d'utilisation de traitement de texte.

Cette décomposition est donc dictée par les missions liées aux agents, c'est-à-dire par les tâches identifiées et liées aux rôles. Par exemple une secrétaire n'aura pas forcément à gérer un agenda, mais pourra avoir comme mission de prendre des notes en sténographie. Elle doit donc avoir une compétence particulière de sténographie, et l'utilisera pour les tâches de sténographie qui lui seront attribuées.

Chaque compétence comprend donc des capacités (opérations) nécessaires à la réalisation des tâches décrites par les missions, et dont le regroupement en modules de compétences est dicté par la logique des opérations.

La conception des modules de compétences doit aussi tenir compte du facteur de réutilisabilité, en considérant qu'une compétence peut être utilisée dans d'autres contextes. La détermination des compétences doit donc tenir compte de l'indépendance possible des tâches par rapport à la mission.

Dans l'exemple de l'application de monitoring, le rôle d'interface exige une compétence d'affichage, dont une des tâches peut se traduire en opération de lancement d'applets java, et une autre en communication entre les applets et l'agent en charge de l'interface.
Il est tout à fait possible de construire cette compétence de façon à ce qu'elle soit indépendante du type d'affichage, et des commandes que peut rentrer l'utilisateur, laissant ainsi la possibilité de réutiliser cette compétence pour d'autres rôles.
Cette modularité qui existe et qui est recherchée dans l'approche objet est donc tout à fait disponible dans l'approche agent.
La différence que l'on trouve ici est à l'avantage de l'approche agent, et en particulier à l'architecture DIANA que nous proposons, car elle permet de changer, par exemple pour une mise à jour, ce module de compétences d'une manière dynamique sans avoir à stopper l'application.
Pour revenir à notre exemple, le rôle d'interface va donc nécessiter (1) une compétence pour la gestion des applets dont nous venons de parler, (2) une compétence de présentation des informations, et (3) une compétence pour traduire les requêtes des utilisateurs vers le master, et du master vers la compétence de présentation.
En considérant que les compétences (2) et (3) sont suffisament liées par le fait que les requêtes de monitoring et les informations qui sont produites à la suite de ces requêtes sont indissociables, il est tout naturel de les regrouper en un seul module.

La conception des compétences doit aussi passer par :

- la description des opérations qui doivent être réalisées
- les références aux autres compétences dont dépend la compétence à créer (hiérarchie de compétences)
- la description des types d'informations échangées entre les opérations

Une fois que les compétences sont décrites, il est possible de valider l'ensemble du système en utilisant les scénarios d'utilisation décrits par le cahier des charges, avant de passer au design des opérations et à la réalisation.

Le lecteur pourra remarquer que si les compétences offrent les capacités opérationnelles pour l'exécution des tâches, ces tâches ne sont pas décrites par les compétences à ce niveau de la conception.

Pour comprendre cette remarque reprenons l'exemple simple de l'interrupteur que nous avons cité chapitre V.3. Un agent a la tâche d'éclairer une pièce, sa compétence sera par exemple limitée à l'utilisation de l'interrupteur, et sa tâche sera d'utiliser cette compétence pour qu'il y ait de la lumière dans la pièce.
En ce qui concerne le monitoring, une des tâches du master est de répartir les équipements à monitorer sur les agents de son domaine (DomainMoniteur).

La question que l'on peut soulever ici est: qui s'occupe d'attribuer les tâches, si celles-ci ne sont pas déterminées par une compétence particulière. C'est ce que nous allons voir en traitant les opérations et les croyances.

V.5.7 Opérations et Croyances

Après avoir décrit les modules de compétences et les capacités qui doivent être offertes par ces modules, il reste au concepteur à spécifier les détails de ces opérations en termes fonctionnels. Nous rappelons qu'une tâche correspond à une instance d'une opération, c'est pourquoi les deux termes seront utilisés dans le même sens ici.

Pour chaque module de compétences, le concepteur doit donc se pencher sur les questions suivantes :

- Est-ce que le module aura besoin d'un temps de préchauffage et d'une initialisation particulière avant d'être utilisable ?
- Quelles tâches sont réalisables avec cette compétence?
- Quelles relations y a-t-il entre les paramètres utiles à l'exécution d'une tâche, et les croyances produites comme résultat de l'opération?
- Quelles sont les croyances (pré-conditions) qui permettent de vérifier que l'opération peut se réaliser?
- Quelles sont les croyances qu'utilisera l'opération pendant son exécution?
- Quelles sont les informations (croyances) qui ne peuvent être connues des agents et doivent donc être fournies par l'utilisateur?

Les croyances qui seront spécifiées en réponse à ces questions vont permettre de préciser les modèles d'information qui seront utilisables par les agents.
Dans l'exemple de l'agent responsable de l'éclairage (compétence éclairagiste), l'opération "basculer l'interrupteur du salon" doit produire une croyance "interrupteur activé", et si le senseur visuel de l'agent ne détecte pas de lumière, une croyance de type "problème électrique sur la lumière du salon" pourra être générée. Cette croyance sera adressée à un agent ayant les compétences nécessaires pour comprendre cette croyance et traiter le problème par exemple en fournissant un diagnostic sur l'état de l'ampoule.

V.5.8 Conclusion

Ce chapitre propose un cadre méthodologique pour la conception d'un SMA basé sur les agents DIANA. Comme le fait remarquer [Wag00] les techniques de modélisation orientées agent sont encore en balbutiement, et nous sommes convaincus qu'elles resteront encore très dépendantes des architectures sous-jacentes tant qu'une uniformisation des concepts agent ne sera pas menée et approuvée par la communauté scientifique et industrielle.

Actuellement la majorité des travaux sur la modélisation orientée agent s'est portée sur les échanges ou interactions entre agents dans des contextes spécifiques [Jen00], et les travaux sur la modélisation et les méthodologies de conception de SMA ne sont applicables qu'avec les architectures agent ayant servi comme base aux recherches. Quelques rares approches plus généralistes comme [GF99] définissent un cadre utilisable pour la description de l'organisation des agents.
Le cadre méthodologique que nous avons exposé dans ce chapitre n'échappe pas à cette remarque, car il n'est utilisable que si les architectures agent offrent la souplesse de fonctionnement des agents DIANA.
Une autre remarque importante que nous devons faire ici concerne l'approche comportementale des agents.
Il a été précisé en introduction de ce chapitre que les agents avaient un comportement difficilement modélisable par les méthodes objet. Puis nous avons défini le comportement comme étant l'ensemble des opérations qu'exécute l'agent face à une situation. Un *comportement* est un ensemble ordonné d'opérations, dont ont connaît les causes et les effets en termes de croyances. Un comportement induit est un comportement qui découle de l'exécution d'un autre comportement. Le comportement peut se rapporter à une application, à un agent, à un groupe d'agents.
La question que l'on doit soulever ici est: l'approche agent permet-elle de modéliser les comportements de ses agents?
Des tentatives comme [JC97] ont été menées avec quelques succès, mais les auteurs précisent les nombreuses limitations de leurs approches, dues à la complexité des échanges qui s'accroît rapidement avec le nombre d'agents.
Cette complexité n'est pas toujours ressentie pour peu que les agents soient en nombre relativement limités, ou que chaque agent soit considéré comme travaillant indépendamment [SPDW97]. Il n'est alors plus question de collaboration ou de délégation mais uniquement de fonctionnalités traduites sous forme de comportements.

Au cours de nos travaux ces questions sur le comportement des agents et des applications ont souvent été abordées. Mais du fait même de l'adaptabilité recherchée, de la haute dynamicité des systèmes multi-agents, une autre question a été soulevée: faut-il figer le comportement de l'agent sous forme de modèles?

En effet, un des avantages attendu et recherché de la technologie agent est la manifestation d'un comportement adaptatif pour atteindre ses objectifs dans un environnement hautement dynamique. Le comportement adaptatif de l'agent se manifestant par sa capacité à rechercher comment atteindre un de ses objectifs, par l'intermédiaire de plans. L'établissement d'un plan conditionne le comportement que va manifester l'agent pour passer d'une situation à une autre.

Ce comportement peut être figé à la conception, comme dans l'approche objet où l'algorithme des méthodes est décidé à la conception, ou construit par l'agent en fonction des compétences disponibles et de son "méta-comportement". Nous appelons ici méta-comportement, le comportement que montre l'agent pour construire les comportements (enchaînement d'opérations pour atteindre un objectif).

La conclusion que nous retrouverons en analysant le résultat des études de cas que nous présentons le chapitre suivant, est que les agents présentent des comportements incertains et par nature non modélisables, du fait même de leur autonomie et de la nature hautement dynamique de leur environnement. Ce constat qui peut paraître négatif lorsque l'on est habitué à contrôler et à traquer le moindre écart de comportement de nos objets, car ils sont la manifestation de bugs, est au contraire encourageant dans l'approche agent car il est la manifestation de la capacité des agents de s'adapter à de nouvelles situations. Nous rejoignons ici [Jen00] pour qui dans des environnements très dynamiques l'émergence de comportements d'un SMA est souvent le modèle le plus approprié pour trouver une solution aux problèmes complexes.

VI Études de cas

Nous revenons dans ce chapitre sur les intérêts que présente l'approche agent pour résoudre les problèmes rencontrés par les systèmes de gestion de réseaux traditionnels. Nous avons vu par exemple dans les chapitres précédents que cette approche est particulièrement intéressante pour le développement d'applications distribuées, notamment parce qu'elle offre les propriétés de délégation et de coopération.
Mais le plus souvent ces propriétés sont développées et étudiées séparément. Par exemple si la distribution de l'intelligence vers les équipements, ou plutôt vers les agents en charge de la gestion des équipements, a déjà fait l'objet de propositions [GA97][KSSZ97][SKN97], l'approche généralement utilisée consiste à transmettre des scripts à des agents intermédiaires (proxy), agents qui ne possèdent ni propriété de délibération, ni même de propriété de coopération.
Cette limitation est également dénoncée par d'autres chercheurs comme [LNNW98], qui constate aussi que si de nombreuses publications ont été faites autour des propriétés fonctionnelles des systèmes multi-agents, leurs propriétés non fonctionnelles comme la stabilité ou l'extensibilité ont été insuffisamment traitées.

Dans ce chapitre, nous proposons d'étudier l'utilisation conjointe des multi-propriétés de systèmes multi-agents DIANA pour valider l'approche agent dans le cadre de développement d'applications de gestion de réseaux.
L'objectif que nous nous fixons ici est de démontrer que cette approche n'est pas confinée à la résolution de quelques applications particulières, mais que le potentiel qu'elle possède va même au delà des propriétés annoncées. Pour cela nous avons défini deux études de cas.

La première étude permet de montrer la facilité avec laquelle on peut développer un système multi-agents dynamique et résistant aux défaillances.

La deuxième met en avant la flexibilité des applications développées avec un SMA utilisant les propriétés de délégation et de coopération.

VI.1 Fiabilité des systèmes

Nous avons vu lors de l'étude des problèmes des systèmes de gestion que la fiabilité est d'une importance primordiale aussi bien pour les opérateurs ou les administrateurs que pour les utilisateurs. Dans la présentation de ses services réseaux [http://www.att.com/network/standard.html], AT&T argumente que «le premier objectif des services réseaux et informatiques est la fiabilité des réseaux, parce que c'est la clef du succès à l'approche du prochain millénaire où l'on attend que les communications soient rapides, efficaces, et disponibles n'importe où, n'importe quand ».

Dans les approches centralisées traditionnelles, la fiabilité du système de gestion en phase opérationnelle est assez peu étudiée. On considère en effet que la fiabilité d'un NMS doit être traitée pendant les phases de développement, où les tests unitaires et d'intégration sont les moyens utilisés pour vérifier la solidité de l'application finale. Au niveau télécom la conséquence de cette démarche est qu'une fois mis en opération, le système de gestion est figé pour éviter qu'il ne «plante».
Et en cas de nécessité, sur les réseaux particulièrement critiques, les solutions utilisées pour fiabiliser les systèmes sont basées sur la redondances des équipements participant à la gestion, et sur des réseaux fortement maillés pour assurer le transport des informations d'un système à un autre.
Une première remarque que nous pouvons faire ici est qu'en dehors du coût élevé de telles solutions, cette approche n'est pas compatible avec l'évolution rapide du marché et des technologies, ni même vraiment adaptée à la gestion de réseaux locaux dont la configuration est souvent modifiée pour s'adapter aux nouvelles exigences de ses utilisateurs.
Par défaut ou par obligation, les administrateurs doivent faire confiance au NMS qu'ils utilisent, et donc à tous les éléments (ex : processus de monitoring) qui le composent. Dans le cas où la défaillance se manifeste par l'arrêt pur et simple du processus de gestion ou par l'indisponibilité d'une des fonctionnalités, celle-ci sera détectée rapidement, par l'administrateur.
Les agents de gestion qui, quant à eux, sont placés sur les équipements, sont considérés comme fiables s'ils répondent aux sollicitations du système de gestion. Grâce aux protocoles de gestion utilisés (SNMP ou CMIP), un agent de gestion qui ne répond pas à une opération SET ou M_SET, qui sont des opérations confirmées, sera repéré et considéré comme défaillant.
Par contre si à l'origine de la défaillance il y a une instabilité d'un composant, celle-ci sera très difficile à détecter. La détection de ce type d'erreur réclame en effet une expertise importante de la part de l'administrateur afin qu'il puisse déduire de quelques événements, d'apparence anodins, que le système de gestion ne se comporte pas correctement.

Jusqu'à présent, ces problèmes au niveau des agents de gestion n'étaient pas considérés comme majeurs, d'un côté en raison de la difficulté de détecter un comportement anormal d'un agent de gestion, et de l'autre parce qu'un agent de gestion est avant tout un élément à faible responsabilité. Il n'y a donc pas de danger qu'un agent prenne de mauvaises décisions, et entraîne un disfonctionnement de l'ensemble du réseau.
Malgré tout, le comportement défaillant d'un agent de gestion aura quand même comme conséquence de risquer de masquer, ou de retarder, la détection et l'identification d'une panne d'un élément du réseau.
Dans le cas d'une approche distribuée comme celle proposée par les SMA, la confiance que doit accorder l'administrateur à son NMS est de tout autre nature.
Contrairement aux systèmes centralisés où l'administrateur a la possibilité de vérifier l'état des processus participant au traitement des informations de gestion, dans un système distribué, ce contrôle est plus difficile. Il ne peut raisonnablement pas faire manuellement une vérification exhaustive de l'ensemble des systèmes pour vérifier qu'ils fonctionnent tous, et il doit donc pouvoir faire globalement confiance au système pour détecter la défaillance d'un de ses éléments.
Concrètement cela veut dire que l'administrateur doit pouvoir se fier non seulement aux informations qui lui sont remontées par ses agents, mais aussi à la capacité du système à pouvoir poursuivre ses activités en cas de défaillance.

Dans un système distribué la fiabilité du système est fonction de la fiabilité de chacun de ses composants. La défaillance d'un composant affecte donc moins le comportement global du système. Cette estimation ne tient pas compte du découpage fonctionnel de l'application, ni de l'organisation du système c'est-à-dire des relations entre les composants, qui ont bien sûr une influence sur la fiabilité totale du système. D'un côté, avec l'approche centralisée, la défaillance d'un processus de gestion entraîne l'arrêt de la fonctionnalité correspondante, et sa détection est assez immédiate. De l'autre côté, l'approche distribuée permet de répartir les fonctionnalités du système de gestion, avec comme avantage le fait que l'arrêt d'un processus de gestion n'a qu'un impact réduit sur la fonctionnalité correspondante, et comme inconvénient, la difficulté de détecter la défaillance de ce processus.

Nous montrons dans cette étude qu'il est possible de détecter et de suppléer un élément défaillant du système distribué, en l'occurrence un agent, et donc de renforcer la fiabilité et la stabilité globale du système de gestion.
L'originalité du travail qui est présenté dans ce chapitre est de fournir une approche sur la fiabilité du système de gestion indépendamment des tâches de gestion qui lui sont dévolues.
Nous définissons ici la stabilité comme étant la capacité du système à retrouver un état normal à la suite d'une perturbation. Quant à la fiabilité, c'est la capacité du système à remplir ses fonctions malgré les perturbations.

Dans ce chapitre, après avoir discuté du problème de fiabilité des SMA, nous allons décrire l'étude de cas proprement dite, nous expliquerons ensuite l'algorithme SLD qui servira à fiabiliser le système, puis nous détaillerons les spécifications et l'implémentation, et enfin avant de discuter des résultats en conclusion, nous décrirons le déroulement de l'expérimentation.

VI.1.1 Fiabilité d'un agent

Puisque la fiabilité d'un système distribué passe par la fiabilité de ses composants, nous devons donc nous préoccuper de la fiabilité des agents intelligents.

Il existe plusieurs raisons pour lesquelles un agent ne pourra pas fonctionner normalement. Parmi ces raisons nous avons des raisons impliquant :

- le système sur lequel est installé l'agent : surcharge du CPU, problème de mémoire, problème d'alimentation en puissance
- les connexions avec l'agent : défaillance d'un lien, d'une interface, du protocole
- l'agent lui-même : bug dans un des modules de compétences, surcharge d'activité

Comme dans certains cas, il est très difficile d'obtenir de l'agent qu'il constate lui-même un éventuel problème (auto diagnostic) (par exemple en cas de coupure d'alimentation, l'agent ne pourra rien faire), nous avons choisi d'utiliser une approche coopérative de la détection de défaillance.
Nous considérons par conséquent que la défaillance de l'agent se manifeste soit par son absence de réponse aux sollicitations, soit par son silence, soit par la détection d'incohérence dans les informations qu'il communique.
Afin de détecter ce dernier type de défaillance, nous avons utilisé l'algorithme connu sous le nom de SLD (System Level Diagnosis) que nous détaillerons plus loin, et dont la propriété est de diagnostiquer des composants défaillants dans un système distribué.

L'objectif final de notre système de gestion est qu'une fois la détection d'une défaillance effectuée, le SMA puisse réagir sans intervention humaine et redistribuer les tâches de gestion aux agents considérés comme fiables, afin de poursuivre ses activités.

VI.1.2 Description de l'étude de cas

Pour étudier la fiabilité d'un SMA, nous avons choisi de traiter une fonctionnalité indispensable à la gestion de réseaux, le monitoring.

Nous considérons au départ un NMS à base d'agents DIANA en charge de la gestion d'un réseau (voir figure 30). Ce NMS a une activité de gestion de fautes dont l'organisation est basée sur la répartition topologique des équipements, ce qui en soi est assez classique.
Pour détecter les fautes du réseau, le système monitore le réseau et fournit à l'administrateur du réseau une présentation de l'état de celui-ci, grâce à une interface sur laquelle sont affichés les équipements sensibles du réseau (routeurs, commutateurs, concentrateurs, serveurs,...). Tout équipement défaillant est indiqué comme KO, tandis que sont affichés les temps de fonctionnement des équipements valides.

La responsabilité de la surveillance de ces équipements est distribuée aux agents DIANA, chacun ayant un domaine d'équipements à monitorer (en fonction de leur répartition topologique). L'interface indique pour chaque agent le domaine des équipements à gérer.

Cette expérimentation doit permettre de vérifier que le SMA est capable de détecter la défaillance d'un des agents en charge du monitoring, grâce à l'implémentation d'un algorithme de diagnostic de systèmes que nous décrirons plus loin. L'interface doit donc permettre l'activation et la désactivation de cet algorithme, ainsi que le pilotage de la simulation de la défaillance d'un agent et d'un équipement.

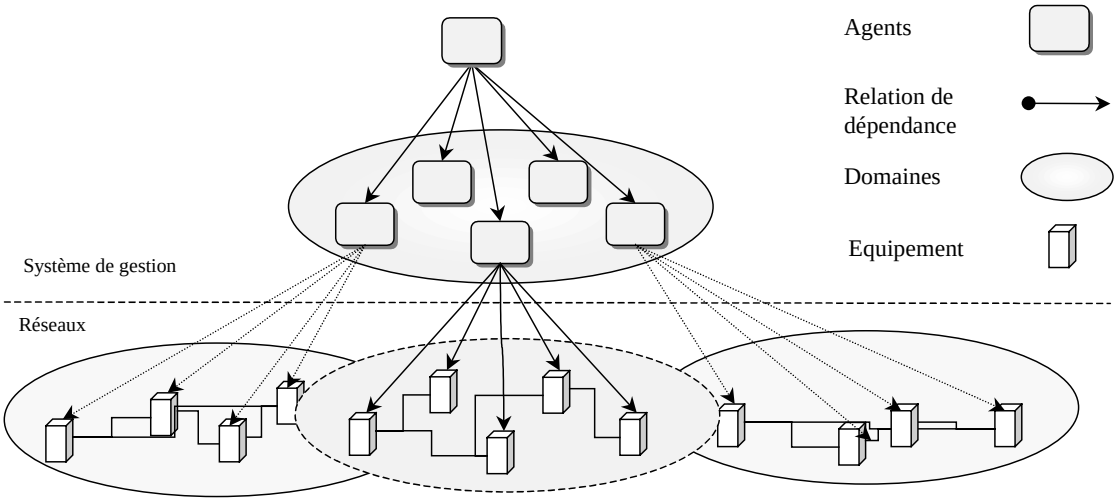


figure 30 Organisation hiérarchique de la gestion de réseaux

Nous mettrons aussi en valeur les propriétés agent comme la délégation et la coopération qui sont explicitement ou implicitement utilisées dans l'implémentation de cette étude.

VI.1.3 Algorithme SLD

Etudié depuis plus de trente ans, le modèle de diagnostic SLD est basé sur des tests mutuels qu'exerce chaque élément d'un système sur les autres [PMC67]. Dans le système tous les éléments sont à tour de rôle testeur et testé, ce qui présuppose qu'ils peuvent communiquer entre eux, chacun déterminant si les autres ont OK ou KO.

Issue de la théorie des graphes, SLD a été adapté aux problèmes de diagnostic de fautes dans les systèmes distribués par Chutani [Chu95]. Chutani considère un réseau comme un graphe, où chaque élément participant est un nœud, et les liens de communication les arêtes du graphe. L'ensemble des résultats des tests effectués par les éléments testeurs sur les éléments testés est appelé syndrome et sert à faire le diagnostic final, c'est-à-dire établir l'ensemble des éléments non fiables du système.

Cet algorithme a été décliné sous plusieurs variantes répondant à des contraintes différentes :

1) centralisé
2) distribué
3) adaptatif
4) à une passe

Nous avons choisi d'utiliser l'algorithme proposé par [CN94] qui est du type adaptatif, en le simplifiant pour qu'il donne un résultat en une seule passe. Le corollaire de ce choix est, comme nous le verrons, que le système n'est diagnostiquable que si au moins la moitié des agents est fiable. Dans le cas peu probable où plus de la moitié des éléments sont défaillants (KO), le système ne pourra être plus diagnostiqué et l'administrateur sera informé que le système de gestion ne peut plus s'auto-stabiliser.
L'algorithme utilisé pour déterminer si le système peut être diagnostiqué est présenté dans le cadre précédent [Mar98]

Soit : S le graphe de test,
V(S) l'ensemble des nœuds,
U l'ensemble des nœuds OK ,
C l'ensemble des candidats,
$T(u_i, u_k)$ le résultat du test de u_i sur u_k
$U = V(S)$, $C = \{\varnothing\}$
Tant que $U \neq \{\varnothing\}$ et $|C| < (|V(S)| +1) /2$
Si $C = \{\varnothing\}$ alors
Soit $u \in U$
$C = C + \{u\}$
$U = U - \{u\}$
Fin si
Si $U \neq \{\varnothing\}$ alors
Soit $u_j \in C$
Si $\exists u_i \in U / T(u_i, u_j) = OK$ alors
$C = C + \{ u_i \}$
$U = U - \{ u_i \}$
Sinon
$C = C - \{ u_j \}$
Fin si
Fin si
Fin tant que
Si $|C| \geq (|V(S)| +1) /2$ alors le système est diagnostiquable

Chaque agent du système teste la fiabilité apparente de certains de ses voisins et transmet le résultat de ses hypothèses à un agent chargé de l'exécution de cet algorithme.
Prenons l'exemple d'un ensemble de cinq agents (Agent1, Agent2, Agent3, Agent4, Agent5). Chaque agent surveille trois de ces voisins comme illustré sur la figure 31, et

reporte à un agent responsable que nous appellerons MasterSLD le résultat de ses constatations.

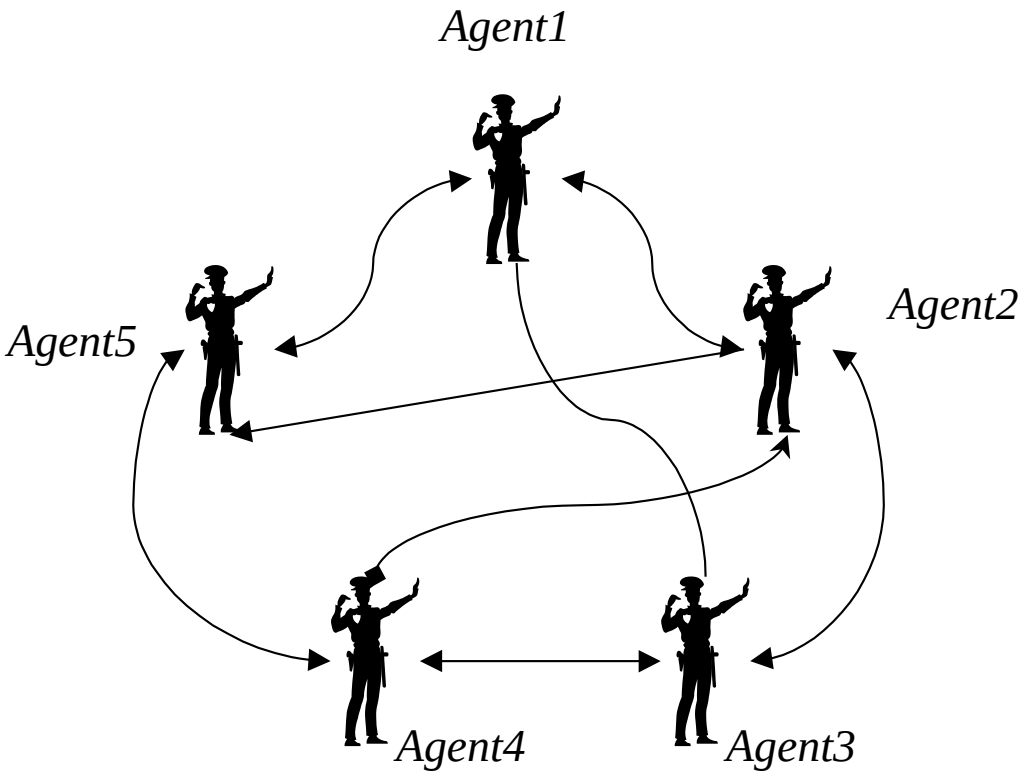


figure 31 Surveillance mutuelle dans un système SLD

L'agent MasterSLD reçoit donc de chaque agent une liste de couples (agent, état) qui lui permet de construire la matrice suivante:

Matrice S	Agent1	Agent2	Agent3	Agent4	Agent5
Agent1	-	Ok	Ok	-	Ok
Agent2	Ok	-	Ok	Ok	Ok
Agent3	Ok	Ok	-	Ok	-
Agent4	-	Ok	Ok	-	NotOK
Agent5	Ok	Ok	-	Ok	-

Le graphe *S* est construit à partir de ces informations, et l'utilisation de l'algorithme SLD permet de déduire les agents que l'on peut considérer comme fiables. Dans l'exemple ci-dessus, l'Agent5 est considéré comme non fiable par l'Agent4, mais le résultat donnera quand même l'Agent5 comme fiable en raison du témoignage favorable apporté par les agents Agent1 et Agent2.

Bien entendu l'algorithme SLD est surtout important dans les cas plus compliqués où plusieurs agents sont considérés comme non fiables par un ou plusieurs de leurs voisins,

car il permet de déterminer si ces voisins sont eux-mêmes fiables, et donc de considérer leur jugement comme objectif ou pas.
Cet algorithme produit en tant que résultat une liste *V(S)* des agents considérés comme fiables.

VI.1.4 SLD et DIA NA

Nous avons considéré dans cette implémentation qu'un agent était fiable si les données qu'il détient et transmet sont correctes. Les données que nous utiliserons sont issues de la perception qu'a l'agent de son environnement, et plus particulièrement celles issues de la surveillance de son domaine de gestion. Chaque agent, comme nous l'avons spécifié dans la description de l'étude de cas, est en charge du monitoring d'un certain nombre d'équipements, dont l'ensemble est appelé domaine.

Pour satisfaire leur rôle de moniteur, les agents vont régulièrement interroger les équipements de leur domaine en utilisant les protocoles SNMP ou CMIP, et transmettre le résultat aux agents qui en font la demande.
Pour pouvoir vérifier la validité des données transmises par un agent A, et tester ainsi sa fiabilité, un agent B interrogera lui aussi un équipement appartenant au domaine de

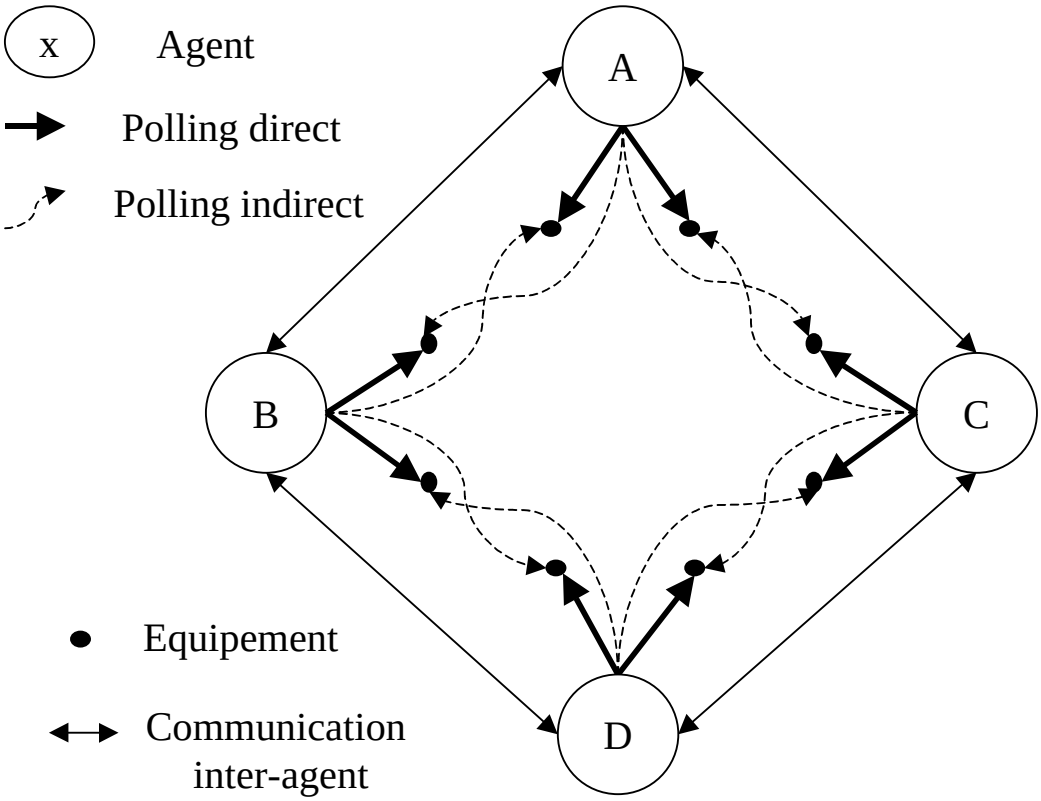


figure 32 Relations inter - agents

l'agent A. L'agent B pourra ainsi comparer les valeurs qu'il a lui-même obtenues par rapport aux valeurs correspondantes qu'il a reçues de l'agent A.

La figure 32 montre les relations qu'il y a entre les agents SLD. Ceux-ci monitorent les équipements sous leur responsabilité (polling direct) et un équipement de leur voisin (polling indirect) et s'échangent les informations sur les valeurs obtenues.

Comme les agents ne vont pas forcément interroger les équipements au même moment, il est probable qu'il existe une différence sur la valeur des données à comparer.
Il est donc indispensable de considérer la fraîcheur des informations échangées et en fonction du type de données, d'admettre une marge de tolérance sur leurs écarts.
Par exemple, si nous choisissons uniquement de tester si les équipements sont accessibles ou non en utilisant une requête de type PING (Packet InterNet Groper), la valeur en retour aura peu de chance de varier, et aucune marge de tolérance n'est possible (accessible, non-accessible). Ce type d'information n'offrira qu'une garantie très limitée sur le fonctionnement d'un agent. En effet la valeur variant peu, il est impossible de garantir qu'elle est issue d'une opération récente de monitoring, ou d'une plus ancienne.
D'un autre côté il est assez difficile d'appréhender la variation de la valeur d'un compteur d'interface entre deux instants, et donc de fixer une marge de tolérance.
Nous avons donc décidé d'utiliser une donnée que l'on sait pouvoir être retournée par tous les équipements et dont la variation est régulière et donc prévisible, le SysUpTime.

Plus précisément, les agents devront renvoyer à leurs collègues qui en font la demande, la valeur du SysUpTime des équipements que nous appellerons éléments représentatifs. Ces éléments seront surveillés par les agents et leurs voisins et les valeurs de leur SysUpTime recoupées. Dans le cas où un agent constate que la valeur donnée par son voisin ne correspond pas à celle qu'il a observée en tenant compte de la marge de tolérance, il considérera cet agent comme défaillant.

VI.1.5 Rôles, organisation, domaines

Le cahier des charges que nous utiliserons ici pour définir les rôles et leur organisation, est composé du chapitre donnant la description de l'étude de cas (VI.1.2) et des chapitres sur la présentation de SLD (VI.1.3, VI.1.4).

Dans un premier temps nous pouvons identifier le rôle de *gestionnaire de fautes* qui est en charge de la supervision des rôles de *moniteur* qui, eux, doivent collecter les informations sur les équipements qui leur sont attribués. Pour cette étude, le monitoring des équipements ne fait pas appel à des connaissances particulières sur le type même des équipements, nous ne définirons donc pas de rôle spécifique pour ceux-ci.

Pour s'occuper de l'attribution des domaines de gestion, nous définissons un rôle de *gestionnaire de domaines*. Ce dernier sera également sous la responsabilité du gestionnaire de fautes afin de lui préparer la liste des agents, et attribuer à ces derniers un domaine particulier de gestion. Enfin nous avons un rôle d'*interface* pour permettre l'affichage des informations résultantes du monitoring.

Pour la mise en place de l'algorithme SLD, dans le chapitre (VI.1.3) nous avons considéré l'existence d'un agent MasterSLD, ce qui correspond ici à un rôle particulier que nous nommerons aussi *MasterSLD*. Un deuxième rôle *SLD* est aussi nécessaire pour la surveillance des voisins et l'établissement des diagnostics.

Ces rôles sont détaillés plus loin mais nous pouvons déjà préciser les relations qu'ils entretiennent avec le graphe suivant.

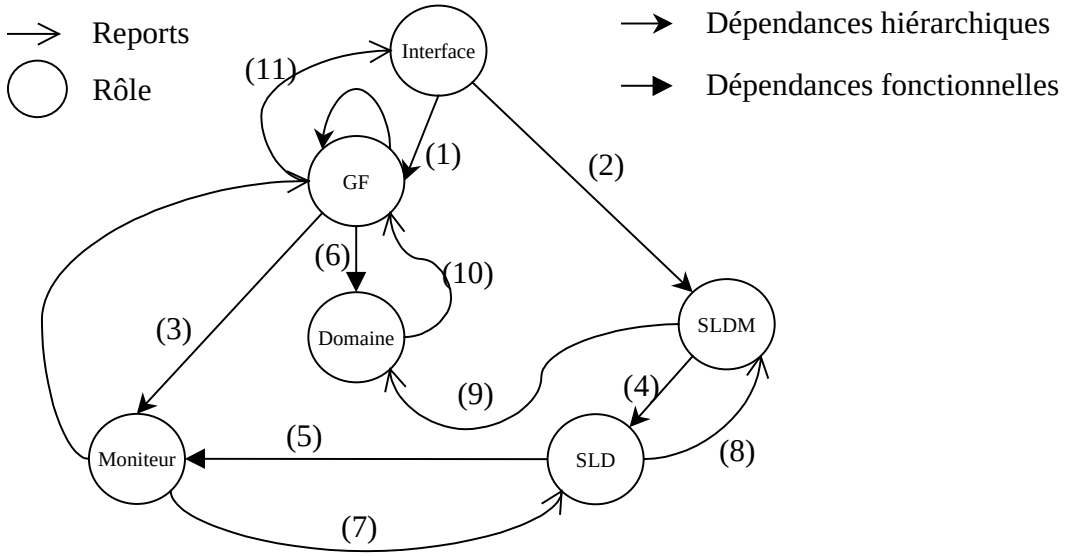


figure 33 Relations de dépendance entre les rôles

C'est à partir de l'interface qu'il est demandé l'activation du rôle de gestionnaire de fautes (1) et celle de l'algorithme SLD (2). L'interface reçoit en retour (11) les informations traitées par le gestionnaire de fautes, pour les afficher ensuite.
Le gestionnaire de fautes ne peut exercer son rôle que si un gestionnaire de domaines est présent (6), de même pour le rôle SLD qui doit demander au moniteur (5) des informations sur l'état des équipements (7).
Le rôle de maître SLDM nécessite l'utilisation de celui de SLD (4), car les informations retournées (8), lui permettent de calculer la liste des agents fiables. Cette liste permettra la modification des informations sur les agents utilisés pour jouer le rôle de gestionnaire de domaines, d'où la relation (9).

Définition des domaines

Fonctionnellement le rôle de gestionnaire de domaines nécessite une liste (ou domaine) d'agents qui est normalement fourni par l'administrateur du réseau lorsqu'il déploie son application. Nous appellerons cette liste *Agentlist*. Une autre liste lui est nécessaire, il s'agit de la liste des équipements que doivent gérer l'ensemble des agents, que nous nommerons *EquipmentList*. A partir de ces données, le gestionnaire de domaines créera et maintiendra le domaine *AgentDomains* qui précise la répartition des équipements par agent.

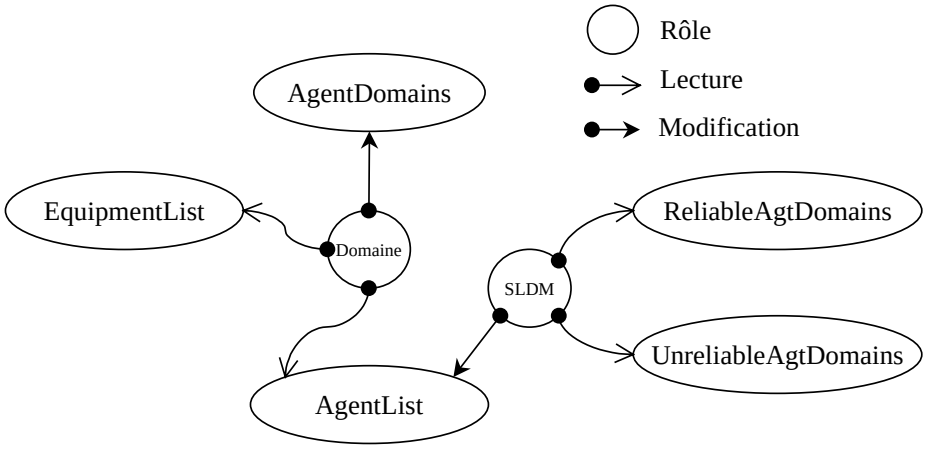


figure 34 Relations entre domaines et rôles

De son côté le SLDM, lui, s'occupe des domaines des agents fiables *ReliableAgtDomain* et des agents non fiables *UnreliableAgtDomain* dont l'union correspond à *Agentlist*. La figure 34 représente ces domaines et leurs relations avec les rôles. Le rôle de l'administrateur qui doit communiquer le domaine AgentList, c'est-à-dire les agents qu'il souhaite mettre à la disposition du gestionnaire de domaines, n'est pas représenté.

Organisation des agents

Pour simplifier la structure organisationnelle des agents, nous déléguerons les rôles de gestionnaire de domaines, de gestionnaire de fautes, de SLDM, et d'interface à un seul agent que nous appellerons *Master*. Les autres agents que nous appellerons *agents domaines* seront quant à eux chargés du monitoring et du rôle SLD de base (surveillance des voisins).

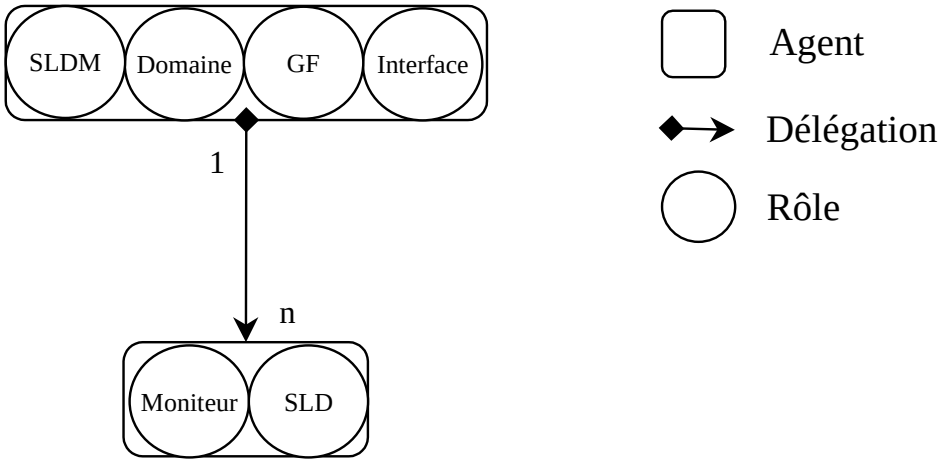


figure 35 Organisation des agents

La figure 35 indique qu'un agent ayant en charge des rôles de responsable SLD (le SLDM), de la gestion de domaines (Domaine), de la gestion de fautes (GF) et de l'interface (Interface), délègue le rôle de monitoring et de SLD à un ensemble n d'agents.
Dans cette figure la relation entre l'utilisateur et l'agent principal n'est pas représentée.

Scénario

Afin de préciser ce que l'on entend démontrer dans cette étude de cas, et par conséquent les missions et les interactions entre les rôles, nous proposons d'utiliser le scénario suivant. Dans un premier temps on vérifie qu'un agent peu fiable ne renvoyant pas des informations correctes sur son environnement passe inaperçu. Pour cela:

1) les agents sont mis en fonction, et l'activité de gestion de fautes est mise automatiquement en route pas l'agent Master, ainsi que l'interface
2) on récupère le panneau de contrôle de l'expérimentation en se connectant à partir d'un navigateur sur l'interface de l'agent Master
3) on vérifie que le monitoring est effectif, c'est-à-dire que tous les agents renvoient des informations sur les éléments de réseau qu'ils ont en charge
4) on active le bouton (1) mettant un équipement hors service et l'on vérifie que cet équipement est bien détecté hors service
5) on réactive l'équipement en utilisant le bouton
6) on utilise le bouton (2) de défaillance d'agent pour qu'un agent devienne non fiable (simulé)
7) on active de nouveau le bouton (1) et l'on constate que l'équipement n'est pas détecté hors service

Dans un deuxième temps, on vérifie le comportement de nos agents lorsqu'ils sont capables de détecter la défaillance d'un des leurs. Pour cela on remet en fonction l'équipement de test et l'agent défaillant (simulé).

1. on active le mode SLD grâce au troisième bouton
2. on utilise le bouton (2) pour simuler la défaillance de l'agent
3. on attend quelques instants pour observer le résultat de cette défaillance
4. on utilise le bouton (1) pour mettre l'équipement hors service et l'on vérifie qu'il apparaît effectivement HS.

Nous pouvons maintenant attribuer des missions aux rôles que nous avons identifiés.

VI.1.5.1 Rôle gestionnaire de fautes

Le gestionnaire de fautes a pour mission de faire effectuer (délégation) les opérations de monitoring par les agents du domaine *AgentDomains*.
Comme c'est lui qui dirige les agents effectuant du monitoring, c'est lui qui est chargé, à la demande de l'interface, d'indiquer à un agent qu'il doit simuler la non fiabilité. Il est en effet peu intéressant de stopper directement un agent, car celui-ci sera détecté directement

comme défaillant et l'administrateur en aura connaissance au travers de l'interface. De plus, il est très difficile de provoquer artificiellement une défaillance de l'agent par ralentissement du système sur lequel se trouve l'agent, celui-ci pouvant aussi bien être un système Unix, Windows, VMS, ou autre. Il a donc été choisi de simuler à la demande cette défaillance au niveau du rôle de moniteur.
De même, comme nous utiliserons le réseau opérationnel d'Eurécom pour l'expérimentation, il nous est impossible de déconnecter physiquement ou de provoquer une panne de l'un de ses équipements.
A la demande de l'interface, le gestionnaire de fautes enverra à tous les agents une demande pour que le monitoring d'un des éléments gérés, désigné au préalable par l'administrateur, ait comme résultat HS (hors service).
Ces demandes concernant la simulation de la défaillance, ainsi que la mise hors service d'un équipement, seront annulables aussi sur demande de l'interface par le gestionnaire de fautes.

VI.1.5.2 Rôle moniteur

Le rôle de moniteur consiste à effectuer un polling, en fonction des paramètres spécifiés par le gestionnaire de fautes, sur les équipements appartenant au domaine de l'*agent domaine*. Nous venons également de voir que le rôle de moniteur consiste aussi à simuler la défaillance de l'agent, en retardant la remise des informations résultantes du polling des équipements.
Le moniteur doit aussi simuler à la demande l'arrêt d'un équipement qui lui est désigné.
Il doit alors produire un résultat identique à l'arrêt réel de cet équipement.

VI.1.5.3 Rôle SLDM

Les missions attribuées au rôle du SLDM sont, à l'activation de ce rôle:

1. désigner les voisins de chaque agent SLD, pour que ceux-ci sachent qui surveiller
2. demander à chaque agent de prendre le rôle SLD et leur donner la liste des voisins qu'ils devront surveiller

En cours de mission:

1. construire et maintenir la liste des agents fiables au fur et à mesure de la réception des informations renvoyées par chaque agent,
2. modifier le domaine AgentList, pour que celui-ci ne contienne que des agents fiables.

Lorsque la mission est terminée:

1. rétablir la liste AgentList originale en cas d'arrêt d'activité SLD

2. informer tous les agents que leur mission SLD est terminée

VI.1.5.4 Rôle SLD

Un agent SLD doit évaluer l'état des agents qu'on lui a demandé de surveiller. Pour ce faire, il vérifie que les informations renvoyées par ceux-ci sont valides en les comparant à celles qu'il possède. Pour cela, dans un premier temps, il demande à ses voisins la liste des équipements (NEs) que ceux-ci surveillent (les éléments représentatifs), puis en choisit un par agent, et demande aux mêmes agents d'être informé (abonnement) sur la valeur du SysUptime de l'équipement choisi.
Il démarre ensuite de son côté une tâche de monitoring sur ces mêmes équipements pour pouvoir comparer les valeurs renvoyées par ses voisins. Ces tâches sont donc les suivantes :

1. demander la liste des NEs monitorés par ses voisins
2. choisir un NE par voisin
3. demander à chaque voisin de le renseigner sur la valeur du SysUpTime du NE choisi
4. monitorer directement les NEs choisis pour récupérer la valeur du SysUpTime
5. comparer les valeurs au fur et à mesure de leur arrivée
6. renvoyer le résultat à l'agent Maître

Lorsque la mission est terminée, l'agent doit de désabonner auprès de ses voisins, et demander l'arrêt des opérations de monitoring qu'il a mis en place pour surveiller les équipements de ses voisins.

VI.1.5.5 Rôle d'interface

Chaque agent possède une interface de communication standard qui lui permet de dialoguer avec un opérateur. Mais afin de rendre l'expérimentation utilisable par une personne n'ayant aucune connaissance de la gestion de réseaux ni d'un système multi-agents, nous avons décidé de rajouter une interface spécifique de test composée de deux parties:

- une applet java avec boutons de commande et tableaux de résultats, que l'on peut lancer de n'importe quel navigateur internet
- un module de compétences qui exprime directement le rôle d'interface en faisant la traduction entre les ordres envoyés par le testeur lorsqu'il presse un bouton et leur traduction sous forme d'objectifs de gestion au niveau de l'agent. Et dans le sens agent → testeur, de préparer les informations de gestion pour l'affichage de celles-ci par l'applet.

La première mission de l'interface est donc de transmettre les classes de cette applet sur demande exprimée par le testeur (plusieurs applets peuvent être utilisées en parallèle).

Puis en fonction des actions du testeur, sa deuxième mission est de traduire les demandes vers le gestionnaire de fautes (simulation de la défaillance d'un agent, ou d'un équipement), ou bien d'activer ou désactiver le rôle de SLDM.

VI.1.5.6 Relation entre les capacités et les croyances

Une fois déterminés les rôles, l'organisation et les missions, nous pouvons spécifier plus finement les relations entre les capacités exigées pour la réalisation des missions, et les croyances nécessaires et produites par ces capacités. Les capacités sont comme nous l'avons vu dans le chapitre Méthodologie les opérations que l'agent doit être capable d'exécuter pour atteindre un objectif, ou pour remplir une tâche.

Nous pouvons représenter les relations qui existent entre les croyances et les capacités avec un graphe où les objectifs expriment l'invocation des opérations. Le graphe suivant (figure 36) exprime les relations qui existent dans l'application de monitoring, dans le cas où SLD n'est pas utilisé.

La relation *implique* indique qu'un objectif ne peut être atteint que si l'opération destination de l'implication est activée. Tandis que la relation *prévenue de* indique que toute modification de la croyance sera communiquée à la compétence responsable de l'opération.

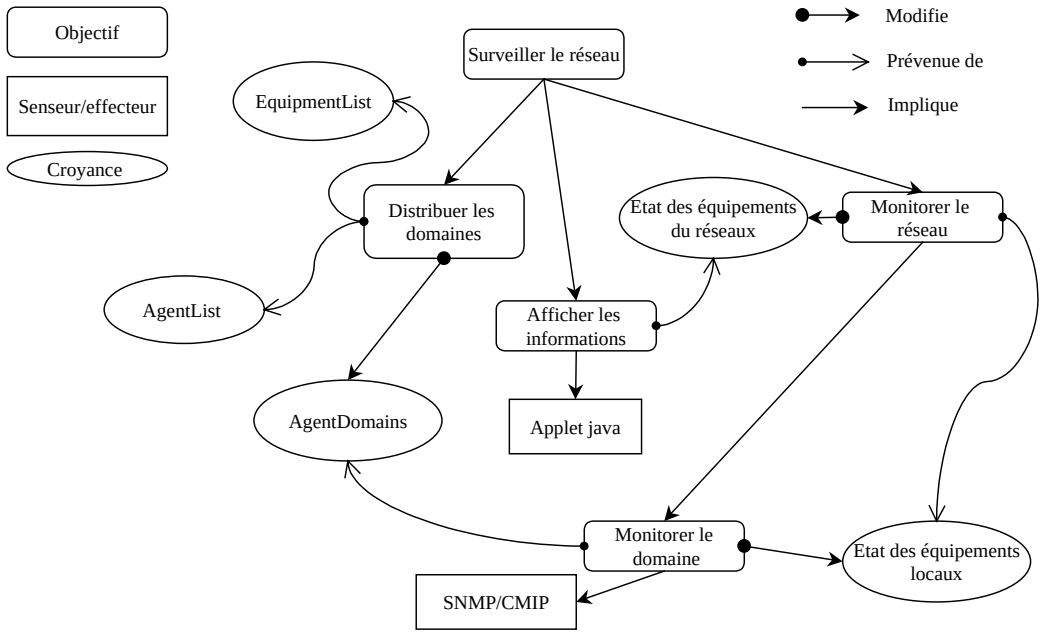


figure 36 Organisation du système

VI.1.6 Description des modules de compétences

Le développement de cette étude de cas a donné lieu à la réalisation de 6 modules de compétences correspondant aux rôles identifiés:

1. SkFaultMgt pour la gestion de fautes
2. SkAgentMgt pour la gestion des domaines affectés aux agents
3. SkSLDM pour le diagnostic du système de gestion SLDM
4. SkSLD pour le diagnostic de l'état des agents voisins
5. SkMonitor pour le monitoring des NEs
6. SkInterface pour la gestion de l'interface

La définition des modules de compétences offre une vue détaillée du design de l'application.

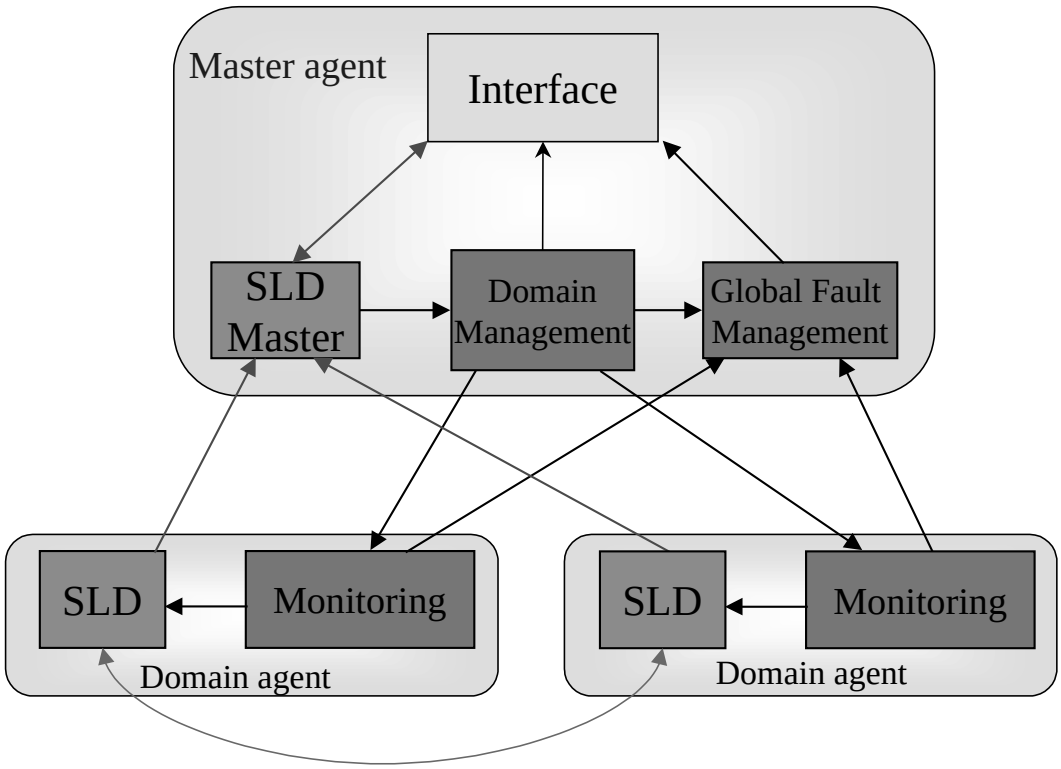


figure 37 Répartition des compétemces

Pour ne pas surcharger ce document nous ne donnons ici que quelques informations sur chacun des modules, et plus particulièrement quelques exemples de croyances produites ou utilisées par chacun d'eux.

La figure 37 montre les relations inter-modules de l'application, les modules étant décrits dans les sous-sections suivantes.

VI.1.6.1 Module gestion de domaines

Ce module utilise la liste des agents disponibles et des NEs connus de l'agent et répartit ces NEs sur l'ensemble des agents.
Plusieurs algorithmes de distribution sont envisageables, comme par exemple la proximité des agents et des NEs, ou la charge des agents. L'objectif de cette étude étant avant tout de démontrer le comportement fiable du système de gestion, nous avons choisi de faire une répartition aléatoire des domaines.
Lorsqu'il est en fonction, le module de gestion de domaines est abonné aux informations concernant l'ensemble des NEs (*EquipmentList*) et l'ensemble des agents (*AgentList*) sous la responsabilité de l'agent qui joue ce rôle. De ce fait, toute modification d'une de ces informations va entraîner la redistribution des domaines.
Voici ci-dessous un exemple de la croyance produite comme résultat de l'affectation des domaines:

(AgentsDomains :agent Ironman :domain (hub101 hub102 hub103)
:agent Hulk :domain (sw101)
:agent Batman :domain (hub301 hub 303 sw202))

VI.1.6.2 Module gestion de fautes

Ce module est responsable de la délégation des activités de monitoring. Il utilise pour cela le module de gestion de domaines qui affecte un domaine de monitoring par agent. Une fois ces informations d'affectation disponibles, le module de gestion de fautes envoie à chaque agent ayant un domaine la description de ce domaine, et la demande de prise en charge de celui-ci pour effectuer le monitoring.
Voici ci-dessous deux exemples de croyances produites par ce module de compétences:

(sendTo Ironman :domain (hub101 hub102 hub103))
(sendTo AgentList start monitoring)

La première croyance correspond à une demande de création de croyance envoyée à l'agent Ironman, concernant son domaine de gestion.
La deuxième croyance concerne l'envoi à tous les agents appartenant au domaine AgentList d'une demande d'activation de leur rôle de moniteur.

Comme nous l'avons vu lors de la description des principes de délégation mis en place pour les agents DIANA, cette délégation implique que les résultats des opérations de monitoring seront envoyés automatiquement à l'agent délégateur. C'est-à-dire dans notre étude de cas l'agent Master.
Le module de gestion de fautes peut donc traiter les informations qui lui sont renvoyées par tous les agents et en faire une synthèse qu'il rend disponible sous forme de croyances pour le module interface.

Ce module est abonné à l'information *AgentDomains* fourni par le module de gestion de domaines, ce qui implique que lorsque cette information est modifiée, le module de gestion de fautes modifie à son tour les informations des agents concernés.

VI.1.6.3 Module monitoring

Ce module est responsable de l'instrumentation des croyances portant sur les NEs qu'il a dans son domaine. L'instrumentation se fait simplement en utilisant la couche réactive comme nous l'avons vu lors de la description de l'architecture DIANA. Cette architecture offre en alternative, la possibilité de déléguer l'instrumentation aux modules eux-mêmes ou à des modules spécialisés. Ces alternatives sont particulièrement intéressantes lorsque l'agent doit communiquer avec des équipements pour lesquels il ne possède pas de senseurs particuliers. Nous avons expérimenté cette possibilité en incluant dans ce module de monitoring le code nécessaire à la communication des opérations de gestion (SNMP).
Comme les autres modules, le module de monitoring est abonné lors de son activation à une information, qui est le domaine de monitoring de l'agent. Il sera donc notifié de toute mise à jour de cette information afin d'adapter son monitoring en conséquence.

VI.1.6.4 Module SLDM

Le module SkSLDM est responsable de la gestion des opérations prévues par l'algorithme SLD. C'est-à-dire la sélection des voisins que chaque agent SLD devra surveiller, le contrôle de l'activité SLD sur ces agents et l'interprétation des résultats.
SkSLDM utilise l'information sur la liste des agents disponibles, et un algorithme de répartition simple, chaque agent devant être surveillé au moins par trois autres.
Il envoie ensuite à chaque agent la demande d'activation du mode SLD, par exemple:

(sendTo Batman start SLD :master Master :neighborList Hulk Ironman)

Cette croyance qui sera transmise par l'agent Master à l'agent Batman, indique que celui-ci a pour mission de surveiller ces voisins Hulk, et Ironman.

VI.1.6.5 Module SLD

Le module SkSLD doit quant à lui, dans un premier temps, demander aux agents inscrits dans sa liste de voisins quels sont leurs éléments représentatifs. Puis il démarre localement le monitoring d'un de ces éléments et demande à être informé des valeurs du monitoring trouvées par les voisins correspondants.

A la réception de ces valeurs, il fait la comparaison par rapport à celle qu'il a lui-même obtenue pour déterminer si ses voisins sont OK ou KO, et il crée alors une croyance contenant le résultat de son algorithme qui sera envoyé à l'agent Master. Par exemple:

(SLD :analyse OK agent: Hulk :status OK agent: Ironman :status OK)

Nous pouvons remarquer à cette occasion qu'il n'y a aucun envoi explicite vers l'agent Master, car celui-ci en tant que délégateur est abonné automatiquement aux résultats produits par les agents délégués.

VI.1.6.6 Module Interface

L'applet développée pour contrôler l'exécution de l'expérimentation comprend trois boutons et deux panneaux d'affichage. Cette applet est représentée sur la figure 38.

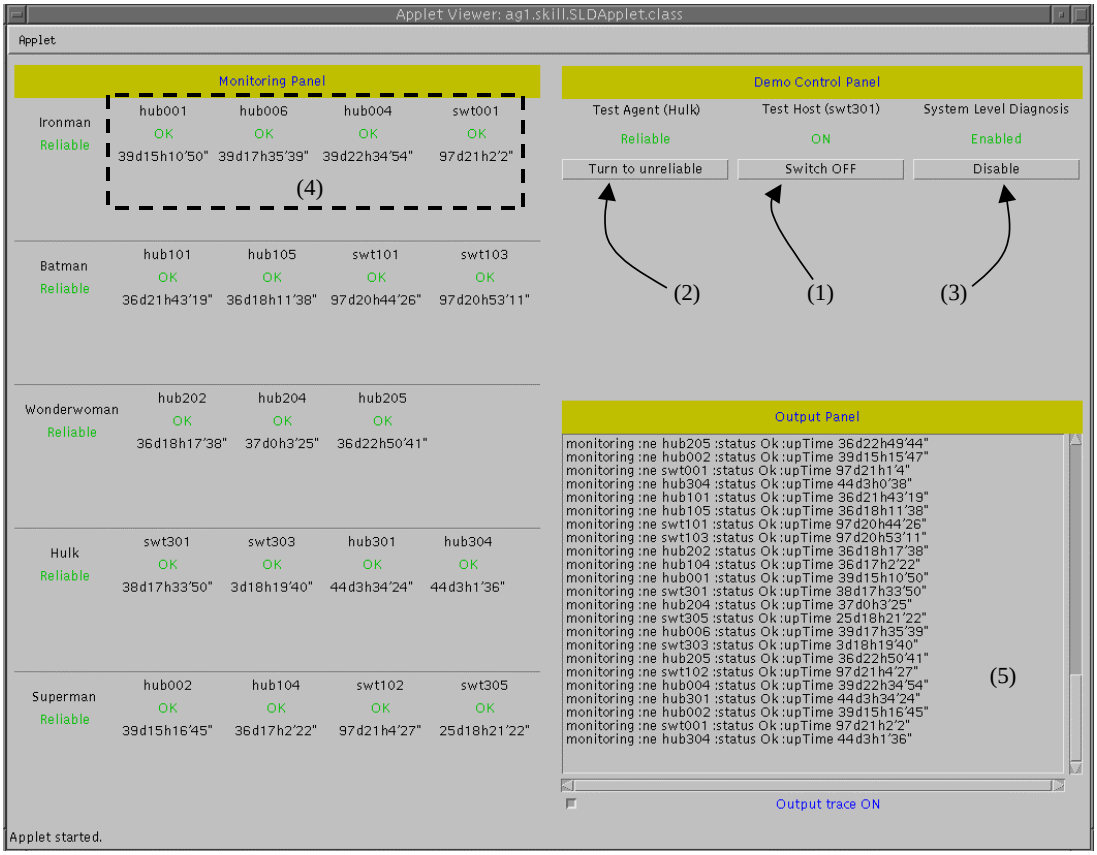


figure 38 Interface de démonstration

Le premier bouton permet de demander à un agent du domaine de simuler une panne en ne renvoyant que des informations erronées (2). Pour cela l'agent choisi possède aussi un module ad hoc pour réagir correctement lorsqu'il reçoit la demande de panne (voir chapitre VI.1.5.2). Le deuxième bouton permet l'envoi d'une information indiquant que le commutateur «switch301» doit être mis hors fonction ou rétabli (1). Cette information

est envoyée aux agents ayant ce commutateur dans leur domaine de gestion. Ceux-ci possèdent un module de compétences leur permettant de gérer cette simulation et ne renvoient plus de données de monitoring sur ce commutateur lorsqu'il est hors service.
Enfin le troisième bouton permet de démarrer ou de stopper l'activité de contrôle de la fiabilité c'est-à-dire SLD (3).

Le premier panneau d'affichage permet de visualiser les agents, les NEs de leur domaine respectif (4) et l'état de ces NEs. L'interface utilise aussi les domaines ReliableAgtDomain, et UnreliableAgtDomain pour afficher sous le nom des agents l'état de ceux-ci (fiable, non fiable).

Le deuxième panneau permet de tracer toutes les informations de gestion arrivant à l'agent Master (5).

VI.1.7 Déroulement de l'expérimentation

Une fois programmés, les modules de compétences sont mis à la disposition des agents DIANA. Nous expliquons et commentons ici le déroulement de cette expérimentation.
Dans un premier temps les agents sont lancés sur plusieurs machines d'Eurécom (un agent par machine) au moyen d'un script.
Lors de leur mise en route, les agents lisent leur fichier de configuration dans lequel ils trouvent les informations sur leur identité, sur le répertoire dans lequel il peuvent rechercher ou stocker les modules de compétences, et enfin d'autres croyances concernant les activités qu'ils doivent entreprendre.

Par exemple, le Master connaît ainsi la liste des agents qu'il a sous son autorité (agentList), et la liste des éléments de réseaux qu'il doit gérer EquipmentList. Il démarre aussi l'activité de gestion de fautes en demandant tout d'abord l'activation du gestionnaire de domaines, afin que le domaine *AgentDomains* soit créé, ou maintenu s'il existe déjà.
En parallèle l'agent active aussi le rôle d'interface pour être prêt à transmettre le panneau de contrôle au testeur.

En ce qui concerne les agents en charge du monitoring, nous attirons l'attention sur le fait qu'il n'est pas nécessaire que ceux-ci aient chargé un module de compétences particulier avant la demande d'activation qui leur sera transmise par l'agent Master. En effet si un module de compétences n'est pas chargé lorsque la demande arrive, l'agent le recherchera localement, puis, en cas d'échec, le réclamera auprès des agents qu'il connaît. Néanmoins, pour diminuer les temps de réponse à la suite de la demande d'activation d'un rôle, il est conseillé d'inclure cette demande de chargement dans les fichiers de configuration de ces agents.

Une fois les agents en fonction, à partir d'un navigateur internet lancé sur une machine quelconque connectée au réseau interne d'Eurécom, nous envoyons une requête HTML à l'agent Master. Si l'agent Master est sur la machine Yasamin, la requête est par exemple:

http://yasamin.eurecom.fr:6622/launch.html

où *6622* désigne le numéro du port sur lequel on peut connecter l'agent, tandis que *launch.html* est la page HTML que fournit l'interface pour le chargement de l'applet interface.

A la réception de cette requête l'agent Master transmet alors les fichiers de l'applet java (voir figure 38 pour la représentation de cette applet) qui une fois chargés et exécutés par le navigateur vont nous permettre de suivre l'expérimentation.
En suivant le scénario décrit au chapitre VI.5.1, nous pouvons ainsi vérifier le bon fonctionnement des simulations de défaillance de l'agent, et de la panne de l'équipement (switch 301).

Enfin, après avoir activé le rôle SLDM en utilisant le bouton correspondant sur l'interface (bouton 3), nous pouvons observer le comportement de notre système en déclenchant la défaillance de l'agent (bouton 2).
Nous pouvons observer sur le panneau de contrôle :

1. que l'agent défaillant est identifié comme non fiable au bout d'un temps donné
2. que le domaine de cet agent est vide
3. qu'il a une redistribution de ce domaine sur les autres agents fiables
4. que l'état de tous les équipements sous surveillance est de nouveau affiché au bout de quelques instants

Il faut aussi noter au sujet du point (1) que pour surveiller leurs voisins, les agents domaines utilisent des informations produites à la fréquence paramétrée pour le monitoring des équipements. Le temps nécessaire à la détection d'un agent est donc proportionnel à la valeur de cette fréquence.

VI.1.8 Discussion

Cette étude de cas nous a permis à la fois de valider l'architecture des agents DIANA et de démontrer les propriétés dynamiques des systèmes agent.
La propriété de coopération des agents a été démontrée avec SLD, du fait que tous les agents se surveillent mutuellement pour déterminer la santé du système. Chaque agent dialogue avec ses voisins pour connaître les équipements qu'ils ont sous leur responsabilité, et reçoit de leur part les mises à jour des croyances issues des opérations de monitoring.

La propriété de délégation a été utilisée à la fois pour le rôle de moniteur et pour celui de SLD. L'agent Master envoie aux agents de son domaine de gestion une requête pour que ceux-ci effectuent le monitoring sur un domaine qui leur est propre. Il obtient en retour le résultat de ce monitoring.

Mais au delà de ces aspects agent, cette étude de cas nous montre :

- **le comportement adaptatif d'un système qu'il est possible de développer avec une application orientée agent**
- **la fiabilité que l'on peut attendre d'une application distribuée à base d'agents pour peu que l'on utilise l'algorithme SLD**

Comportement adaptatif

Il est en effet intéressant de voir comment le gestionnaire de fautes peut continuer à recevoir des informations de monitoring fiables sur l'ensemble du domaine de gestion des équipements. Le rôle et l'activité du gestionnaire de fautes sont en effet indépendants de la fiabilité des agents.
Cette notion de fiabilité est introduite avec les rôles SLDM et SLD, et le gestionnaire de fautes va bénéficier de cette fiabilisation du fait même de l'enchaînement suivant:

1. SLDM retire de la liste des agents (*AgentList*) les agents non fiables
2. le gestionnaire des domaines, averti par le cerveau, exécute une nouvelle opération de distribution
3. le gestionnaire de fautes, averti par le cerveau d'une mise à jour de la croyance *AgentDomains,* relance le monitoring des agents concernés

Ce comportement non programmé montre la capacité des agents à utiliser les informations (croyances) qui sont mises à leur disposition.

Certains peuvent objecter que dans un système objet, en cas de défaillance, il suffit de recréer l'objet. Dans cette étude il n'est pas spécifié si c'est l'agent en tant qu'application qui est défaillant ou si c'est le l'équipement sur lequel s'exécute l'agent.
Ce qui est constaté par les autres agents, c'est le comportement anormal de l'agent défaillant.
Ce comportement anormal pouvant être soit l'absence de communication avec cet agent, soit la réception d'informations erronées.
Si c'est l'équipement qui est défaillant, il faut qu'il soit remplacé, ce qui demande une intervention manuelle et rend impossible la création d'une nouvelle instance sur cet équipement.
D'autres peuvent aussi penser que l'application doit avoir été prévue pour prendre en compte ces risques.
C'est sur ce point que la technologie agent présente un avantage essentiel sur la technologie objet en proposant l'adaptation de l'application sans a priori sur sa conception.

Fiabilité d'un système multi-agents

Nous avons montré ici qu'il était possible avec l'algorithme SLD de facilement fiabiliser un système multi-agents, et ceci sans augmenter fortement sa charge de travail, puisque SLD utilise l'activité de monitoring poursuivie par les agents.

Si cet algorithme peut être adapté pour utiliser d'autres données, le choix est ici justifié par le cadre de la gestion de réseaux, où le monitoring en tant qu'activité essentielle sera exécuté par une majorité d'agents.
Dans les cas où l'utilisation d'autres types de données est justifiée, par exemple pour des agents ayant un rôle administratif ou d'organisation, comme l'agent Master dans cette étude, il ne sera pas nécessaire de développer à nouveau le rôle SLDM.
En effet, le rôle SLDM demande uniquement à ses agents leur point de vue sur l'état de leurs voisins, et ce, indépendamment des moyens que ceux-ci utilisent pour estimer l'état de leurs voisins. Seuls les opérations du rôle SLD devront être revues.

VI.2 PVC sur réseaux ATM

Cette deuxième étude de cas a pour objectif de mettre en avant la flexibilité qu'apporte l'approche agent intelligent, aussi bien pour le déploiement que pour l'utilisation des applications. Cette flexibilité est le résultat de l'utilisation conjointe des propriétés d'apprentissage par transfert de compétences et transfert de connaissances, de délégation, de coopération.
Comme support, nous avons choisi de traiter le problème de la gestion des configurations de circuits virtuels permanents (PVC) sur ATM (Asynchronous Transfer Mode). L'intérêt que présente ce sujet est qu'il n'existe actuellement aucun protocole standard pour automatiser la création de circuits PVC pour les réseaux ATM [PLBS98].
La configuration d'un PVC se fait actuellement manuellement, commutateur après commutateur, avec comme difficulté supplémentaire l'hétérogénéité des interfaces de gestion, chaque constructeur proposant une interface propriétaire.
Pour minimiser ce problème d'hétérogénéité, une interface web a été étudiée dans le cadre du projet Utopia [Uto98]. Cette interface propose des fenêtres de dialogue pour définir les paramètres de QoS et de connexion, et utilise le protocole SNMP pour accéder ensuite aux commutateurs ATM et réaliser l'opération de configuration. Tous les inconvénients ne sont pas écartés car la configuration est encore réalisée manuellement, et il est impossible d'avoir une vue de bout en bout d'un circuit ainsi configuré.
Un projet d'automatisation a été proposé par [PLBS98], basé sur l'approche agent mobile. Nous discuterons brièvement des inconvénients comparés de cette approche à l'occasion de la conclusion de cette étude de cas.
Ce chapitre est organisé comme suit : nous commencerons tout d'abord par expliquer la difficulté que représente la configuration de VPC. Puis nous détaillerons le scénario utilisé pour le déroulement de l'expérimentation.
Nous identifierons ensuite les rôles, l'organisation, et les missions nécessaires, avant de détailler les modules de compétences qui ont été développés.
Nous terminerons par la description du déroulement de l'expérimentation et une discussion en forme de conclusion.

VI.2.1 Configuration de PVC

Les réseaux ATM offrent deux types de connexion: les connexions commutées SVC (Switched Virtual Circuit) et les connexions permanentes PVC (Permanent Virtual Circuit). Ces deux types de connexion correspondent à des utilisations différentes. Les SVC sont destinés à des connexions courtes que l'on peut comparer à des connexions téléphoniques, sur lesquelles les contraintes de QoS sont généralement faibles. Tandis que les PVC correspondent plus souvent à des connexions inter-entreprises, sur lesquelles on veut avoir une qualité de service garantie, et un temps d'établissement de connexion le plus faible possible.
S'il existe les protocoles standardisés de signalisation UNI (User-Network Interface) et PNNI (Private Network-to-Network Interface) qui permettent de configurer automatiquement les SVC, il n'existe par contre aucun protocole standard de signalisation pour la configuration de PVC [Uyl98].
Malgré l'existence des standards UNI et PNNI, il est très souvent impossible d'établir une connexion SVC dans un réseau ATM hétérogène en raison d'une part, des différences qui existent entre les implémentations réalisées par les constructeurs, et d'autre part, du fait que ces protocoles sont rarement complètement implémentés.
La configuration d'un circuit virtuel permanent est par contre adapté aux réseaux hétérogènes, ne serait-ce que parce que se faisant manuellement, il n'y a pas de problèmes d'incompatibilité de signalisation.
La démarche nécessaire à l'établissement d'un PVC est la suivante :
Premièrement une route physique doit être déterminée entre les deux points à connecter. Il peut y avoir plusieurs routes possibles, la sélection se faisant alors en fonction de critères comme l'optimisation des ressources. Sur chaque commutateur de la route il faut noter le port d'entrée et le port de sortie.
Ensuite il faut choisir le chemin virtuel (VP) qui sera utilisé pour le PVC. La solution la plus simple, mais pas la meilleure, consiste à utiliser le VP ayant l'identification 0 (VP0), car ce VP est créé par défaut lors de la mise en route du commutateur. L'inconvénient est que chaque VP a une capacité limitée et ne supporte qu'un nombre restreint de VCI (Virtual Channel Identifier).
Classiquement, les opérateurs créent un VP par lien physique entre deux commutateurs, différent du VP0, qui reste disponible en cas de besoin. Si ce choix simplifie la gestion des configurations, l'inconvénient est que le nombre de circuits (VCC) configurables sur ce VP est limité aux valeurs de VCI disponibles pour ce VP.
La dernière étape consiste à sélectionner sur chaque commutateur les VCI en entrée et en sortie et créer une entrée dans la table de routage du commutateur. Au final, entre deux commutateurs, le couple (VPI,VCI) doit être identique pour une même route. Dans le cas contraire une cellule ATM n'atteindra jamais sa destination.
A ce stade, lorsque tous les commutateurs ont été configurés de la sorte, nous pouvons dire qu'il existe un PVC entre les deux points. Mais tout n'est pas fini pour autant, il reste à spécifier la QoS que l'on désire avoir entre ces deux points.
Cette qualité de service est traduite pour ATM en UPC (Usage Parameter Control). Un UPC correspond à un contrat, de réservation de ressources et de gestion de trafic, délivré par le commutateur pour un PVC particulier.
Il faut savoir que sur la route d'un PVC, si un UPC est improprement configuré sur un commutateur, le résultat peut aussi bien ne pas affecter la communication que la rendre impossible. Certaines applications comme la vidéo sont en effet très sensibles à une mauvaise configuration des paramètres comme le taux de perte de cellules. Au niveau du

commutateur chaque UPC créé possède un identifieur (généralement une chaîne de caractères), et il est très difficile de découvrir l'origine d'un problème de communication, car tous les paramètres doivent être revus les uns après les autres sur la route du PVC.

Pour résumer nous pouvons constater qu'il est particulièrement fastidieux et délicat d'établir un PVC entre deux points, car cela exige la prise en compte de nombreux paramètres, et la vérification que les contraintes de qualité puissent être satisfaites de bout en bout. Nous allons donc voir comment une application d'établissement de PVC peut être développée en utilisant les agents DIANA.

VI.2.2 Description de l'étude de cas

Considérons qu'un opérateur utilisant des réseaux ATM, veuille offrir à ses clients la possibilité d'établir automatiquement des PVC. Ses clients ou utilisateurs pourront alors établir des PVC entre leurs succursales, ou avec des clients, sans avoir à contacter au préalable l'opérateur.
L'opérateur peut leur proposer des contrats en fonction du type de service et donc du type d'application comme la vidéoconférence, l'audioconférence, ou même simplement le transfert de fichiers importants à haut débit.
Pour chaque type d'application, il est possible de spécifier des paramètres de qualité de service standards. Par exemple pour une conférence de haute qualité, nous pouvons estimer que le contrat doit spécifier que la classe de service soit de type Constant Bit Rate, avec Peak Cell Rate de 13020 et une tolérance pour la variation de délai de 0,1 ms, et un temps maximum de transfert (Cell Transfert Delay) de 1 seconde [PM97].

Un tableau spécifiant les contrats UPC comme le suivant peut être fourni au client:

PVC **Type de service**	**PVC** **Classe de service**	**ATM**	**ATM** **classe de service**	**Code**
Video on demand	High quality	PCR = 13020	CBR	VDHQ
Video on demand	Medium quality	PCR = 3906	CBR	VDMQ
Video on demand	Low quality	PCR = 15625 SCR = 3438	VBR	VDVQ
Teleconferencing	High quality	PCR = 1000	CBR	THQ
Teleconferencing	Medium quality	PCR = 333	CBR	TMQ
Voice Transfer	High quality	PCR = 1000	CBR	VTHQ
Voice Transfer	Medium quality	PCR = 166	CBR	VTMQ
Voice Transfer	Low quality	PCR = 21	CBR	VTLQ
Data Transfer	Best effort	----	UBR	DTBE

Grâce à une application agent, jouant le rôle d'interface et fournie par l'opérateur, le client peut quand il le souhaite établir lui-même les connexions PVC.
De son côté l'opérateur doit développer et mettre en place un ensemble d'agents pour gérer les demandes client et configurer les commutateurs.

Comme nous avons déjà discuté de l'aspect modélisation, afin de ne pas alourdir ce document nous passerons rapidement sur les différentes étapes d'analyse et de conception.

VI.2.3 Rôles, Organisations, Missions

A partir de cette description succincte de l'étude de cas, nous pouvons déjà définir un certain nombre de rôles, établir leur organisation, et étudier les missions qu'ils doivent remplir.
Tout d'abord du côté utilisateur, un agent doit prendre en charge le rôle d'interface de connexion.
Du côté de l'opérateur, pour gérer les demandes de ses clients, un rôle de responsable utilisateur peut être défini. Ce rôle serait surtout intéressant si une composante commerciale était donnée à cette étude, mais comme nous avons décidé de considérer avant tout le côté fonctionnel de cette application, nous ne définirons pas de rôle de responsable clientèle.
Un agent devant quand même recevoir les requêtes émises par les clients, nous considérerons que c'est l'agent en charge de l'établissement de bout en bout du PVC qui s'occupera aussi des relations avec les clients. Nous nommerons ce rôle MasterPVC.
Comme l'établissement du PVC commence par la détermination d'une route utilisable, nous aurons besoin d'un agent ayant le rôle de topologue, capable de renseigner le MasterPVC sur la ou les routes possibles.
Pour simplifier le design de l'application nous allons considérer qu'un agent est en charge d'un commutateur et d'un seul. Nous nommerons ce rôle SlavePVC.

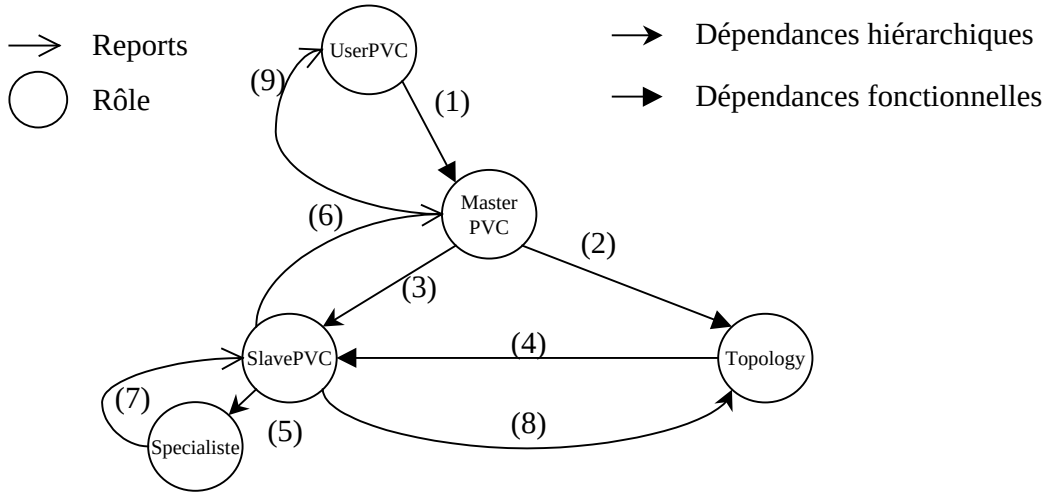


figure 39 Relation de dépendances entre les rôles

L'agent ayant le rôle de MasterPVC va donc contacter chaque agent des commutateurs sur la route du PVC à établir, pour leur demander de configurer localement leur commutateur.

Afin de garder une certaine indépendance par rapport aux équipements, nous pouvons considérer que chaque commutateur possède un spécialiste capable de traduire les demandes standard de configuration en demandes spécifiques à l'équipement. Ce spécialiste ayant le nom du type de l'équipement correspondant (par exemple ForeATM, IBMATM, CiscoATM,...).

Comme le montre l'organigramme utilisé pour la réalisation de cette application (voir figure 39) plusieurs simplifications ont été choisies, comme par exemple l'absence de fournisseur (opérateur ou autre fournisseur de service), et il n'a été pris en compte qu'une seule unité englobant l'ensemble des services, le MasterPVC.

Une fois l'organisation précisée, la deuxième étape consiste à préciser les rôles en leur attribuant des missions. Les sections suivantes présentent ces rôles et ces missions en donnant quelques précisions sur les modules de compétences correspondants qui ont été développés.

VI.2.3.1 Rôle UserPVC

Les missions principales de l'interface utilisateur pour l'établissement de PVC sont les suivantes:

1) demander l'établissement d'une connexion, la suppression d'une connexion auprès du responsable utilisateur du fournisseur de service
2) présenter une interface utilisateur simple pour sélectionner la station à connecter et l'application devant utiliser le PVC.

La figure 40 présente l'interface utilisateur qui a été développée pour établir les connections.

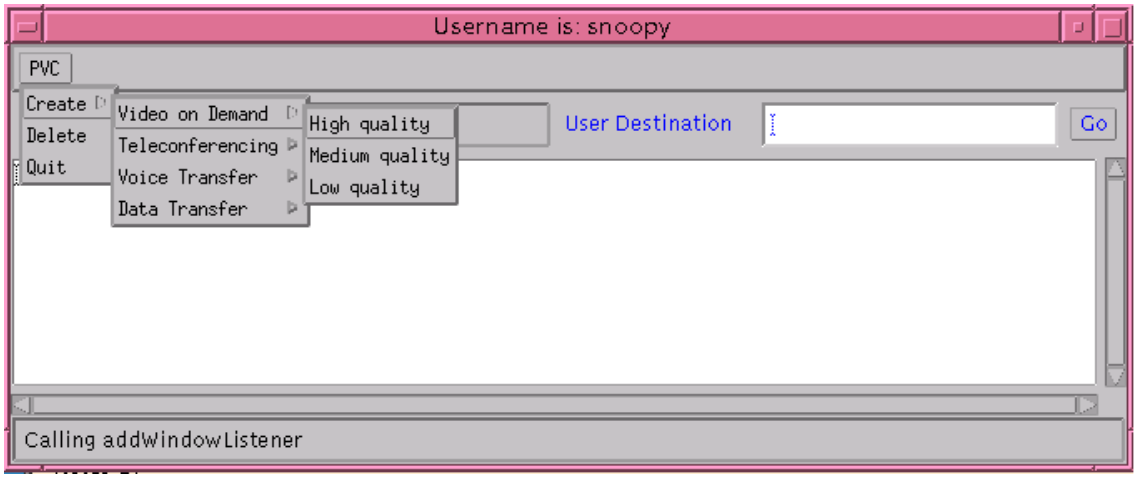


figure 40 Interface utilisateur

Lors de son initialisation, l'agent de l'utilisateur prend en charge le rôle d'interface, ce qui provoque le lancement de l'interface.

VI.2.3.2 Rôle Topology

Le rôle de "topologiste" consiste dans un premier temps à obtenir des agents en charge des commutateurs la description de leurs raccordements physiques pour construire la topologie du réseau ATM.
Nous avons simplifié ce rôle en lui donnant directement cette description sous forme de croyance, le rôle se réduisant ainsi à:

1) fournir les chemins possibles entre une source et une destination à la demande des MasterPVC

VI.2.3.3 Rôle MasterPVC

Comme nous n'avons pas de garantie au préalable que tous les commutateurs sur une route disposent de ressources suffisantes pour établir le PVC, nous avons choisi de procéder en deux étapes. La première consiste à réserver les ressources si elles sont disponibles, et la deuxième d'effectuer proprement dit la configuration.
En procédant de la sorte, nous offrons la possibilité d'optimiser l'utilisation des ressources en cas de fortes demandes concurrentes. La raison principale est que les agents, ne pouvant être pour l'instant directement installés sur les commutateurs, communiquent avec les commutateurs au travers du protocole SNMP. Les temps de modification de la configuration d'un commutateur sont donc loin d'être négligeables, et dans le cas où sur le chemin un commutateur ne dispose pas de ressources suffisantes pour l'établissement du PVC, les ressources déjà allouées sur les autres commutateurs doivent être libérées. Entre temps, si une demande de PVC est émise pour un autre client, celle-ci risque de ne pas pouvoir être satisfaite si la demande arrive avant la fin de la libération des ressources. Alors que dans le cas où la réservation est annulée, l'agent du commutateur prend en compte cette libération immédiatement.

Pour répondre à une demande d'établissement de PVC le MasterPVC va donc:

1) vérifier qu'il existe un chemin pour établir une connexion entre le site du demandeur et le site du destinataire.
2) s'il existe un chemin, demander une réservation de connexion sur tous les éléments du chemin (agents en charge).
3) demander la mise en place de la configuration si toutes les réservations ont été acceptées, et sauvegarder les références de la connexion.
4) reporter à l'utilisateur les résultats de sa demande.

Pour la suppression d'un PVC il devra:

1. vérifier que la connexion est référencée
2. transmettre à tous les responsables sur le chemin les demandes d'annulation.
3. supprimer la référence de la connexion
4. reporter à l'utilisateur le résultat de sa demande d'annulation.

VI.2.3.4 Rôle SlavePVC

Le rôle attribué aux agents en charge d'un commutateur est le rôle de SlavePVC. Pour remplir son rôle celui-ci doit:
Pour une demande de réservation de PVC:

1) vérifier qu'il existe un VP remplissant les conditions de l'UPC, ou qu'un tel VP peut être créé.
2) renvoyer le résultat au MasterPVC

Pour une demande de création:

3) envoyer aux switchs voisins sur le chemin la proposition d'une liste de couples VPI/VCI, et à la réception des listes voisines, sélectionner la plus basse commune.
4) contacter le spécialiste local du commutateur (ex: ForeATM) et lui donner les paramètres de configuration pour les VPI/VCI retenus, ainsi que l'UPC

Pour une demande d'annulation de PVC:

5) transmettre la demande d'annulation au responsable de la configuration locale.

VI.2.3.5 Rôle Spécialiste

Le spécialiste sert d'intermédiaire entre le SlavePVC qui a la connaissance logique des commutateurs, et le commutateur lui-même qui est spécifique suivant les constructeurs.
Le module de compétences développé pour assumer le rôle de spécialiste fournit les services pour créer et supprimer les VP et les VC ; il se sert du protocole SNMP pour gérer les commutateurs.
De tout le système développé pour cette étude de cas, c'est le seul module qui nécessite une adaptation en fonction du type d'équipement. Ne disposant que de commutateurs du constructeur Fore, nous n'avons développé qu'un seul module de compétences spécialisé pour les commutateurs Fore.

VI.2.3.6 Croyances du MasterPVC

Pour donner au lecteur un aperçu de ce que représente la conception du modèle d'information utilisé par un module de compétences, nous donnons en exemple dans cette section les croyances utilisées par l'activité de création de PVC.
Le lecteur pourra se reporter au chapitre IV pour plus de détails sur l'utilisation des croyances par le cerveau de l'agent.

L'encadré suivant contient les différentes croyances que nous commentons. La première croyance correspond à la désignation de l'opération de création proprement dite.
Dans le cas de la création de PVC, la seule croyance prérequise au niveau du MasterPVC est celle contenant le nom de l'agent et de son système, informations qui lui serviront pour créer un identifiant unique à chaque PVC .

```
(  (SkMasterPVC createPVC :source * :dest * :upcNr * :userName $U :reqId $R)
// *****************   prerequisite beliefs ************************
( (me :name * :host *) )
// ***************  created beliefs  ****************************

((SkMasterPVC :result * :pvcId * :reqId $R)
(SkMasterPVC :pvcId * :source * :dest * :userName $U :upcNr * :status *)
(SkMasterPVC :result * :reqId $R :reason *))

// ****************** created goals  ******************************

((SkTopology findSP :source * :dest *)
(SkSlavePVC reservePVC :iport * oport * :upc * :resId *)
(SkSlavePVC createPVC :resId * :pvcId *)
(SkSlavePVC cancelRes :resId *)
(sendTo * SkSlavePVC reservePVC :iport * oport * :upc * :resId *)
(sendTo * SkSlavePVC createPVC :resId * :pvcId *)
(sendTo * SkSlavePVC cancelRes :resId *))

//******************** used beliefs ******************************

( (SkTopology :nodeList *)
( * SkSlavePVC :desc 'reservationDone' :resId *)
( * SkSlavePVC :desc 'reservationError' :resId * :reason *)
( * SkSlavePVC :desc 'creationDone' :resId *)
( * SkSlavePVC :desc 'creationError' :resId * :reason *) ) )
```

Puis sont données les croyances créées par l'exécution de l'opération. Elles sont utilisées par l'agent lorsqu'il doit verifier la possibilité d'atteindre un but corrspondant à la création de beliefs. Il essaie alors de chaîner les opérations en fonction des valeurs des attributs et en cas de réussite, active les opérations correspondantes. Les *goals* créés ne sont actuellement pas utilisés par l'agent, et donnés uniquement à titre indicatif. Par

contre, lorsque l'agent activera l'opération, il abonnera automatiquement le module de compétences sur les *used beliefs* qui sont les dernières informations de la spécification d'une opération. Grâce à cela, le module de compétences sera notifié des opérations effectuées par le cerveau sur les croyances qui l'intéressent, et ceci tant que l'opération est en activité.

VI.2.4 Déploiement du système

En marge de l'étude de cas, nous proposons ici de montrer comment un administateur peut déployer une application, c'est-à-dire comment les agents peuvent collaborer et s'échanger les modules de compétences afin de mettre en place un nouveau service comme celui que nous venons de développer. Nous extrapolons ci-dessous les résultats obtenus à partir du déploiement réel de notre étude de cas sur deux commutateurs FORE ATM, à un ensemble simulé de cinq commutateurs (figure 41).

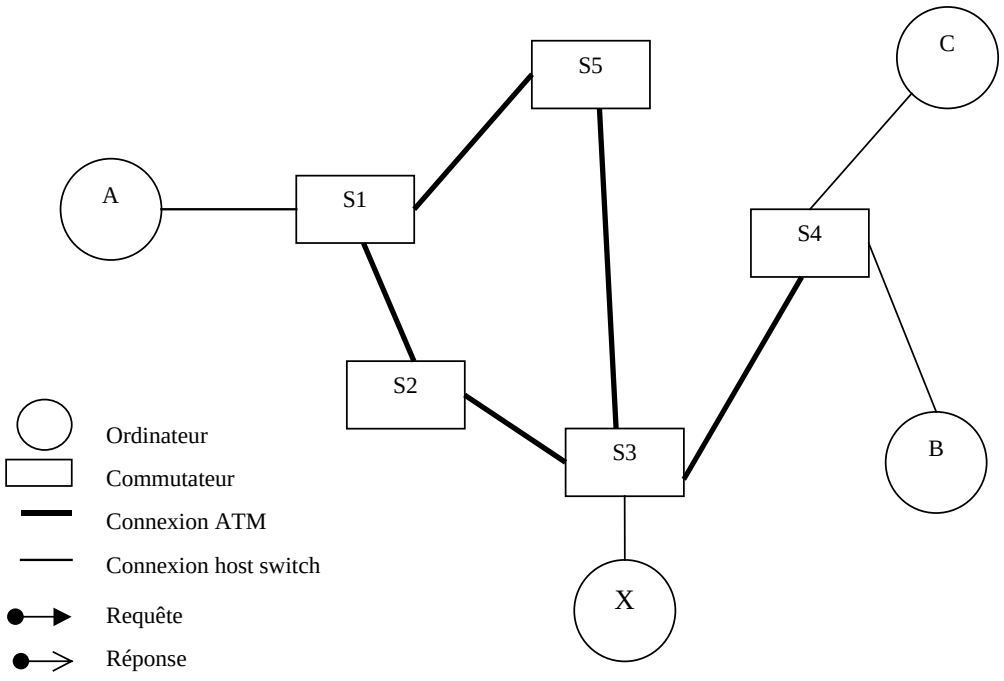


figure 41 Topologie du réseau ATM

N'ayant pas encore la possibilité d'utiliser des commutateurs intégrant des machines virtuelles Java, nous avons lancé nos agents sur des systèmes connectés aux différents commutateurs en affectant un agent par commutateur. Nous rappelons aussi que la couche instrumentation des agents DIANA n'était pas développée au moment de l'expérimentation, aussi une couche de protocole SNMP a-t-elle été intégrée au module de compétences SkForeATM pour permettre aux agents de dialoguer avec les équipements.

Nous nous plaçons dans le contexte où un administrateur de réseaux a à sa disposition un système de gestion à base d'agents DIANA. Ce système est en fonctionnement, et l'administrateur va ajouter un nouveau service de création de PVC. Les conditions sont les suivantes :

- ▷ l'administrateur est connecté sur la station X
- ▷ les agents S1,S2,S3,S4,S5 des commutateurs sont tous actifs
- ▷ les agents S1 à S5 ne connaissent que les agents directement voisins excepté S3
- ▷ les modules de compétences sont tous placés dans l'environnement de l'agent S3, qui est le seul à avoir la liste complète des autres agents.
- ▷ tous les commutateurs ATM sont des FORE

A partir de l'interface utilisateur de l'agent S3 l'administrateur demande à chaque agent de prendre le rôle de SlavePVC. Pour cela il demande à l'agent S3 la création de la croyance suivante:

1. *"sendTo S1 :me SkSlavePVC :switchType ForeATM :name S1"*
2. *"sendTo S2 :me SkSlavePVC :switchType ForeATM :name S2"*
3. *"sendTo S3 :me SkSlavePVC :switchType ForeATM :name S3"*
4. *"sendTo S4 :me SkSlavePVC :switchType ForeATM :name S4"*
5. *"sendTo S5 :me SkSlavePVC :switchType ForeATM :name S5"*
6. *"sendTo AgentSwitchList :me loadSkill SkSlavePVC"*

Les premières croyances (1,2,3,4,5) sont automatiquement envoyées par l'agent S3[5] aux agents correspondants, qui enregistrent alors la croyance (*SkSlavePVC :switchType ForeATM :name* *).
La deuxième croyance créée sur l'agent S3 (6) est de la même manière automatiquement envoyée à tous les agents du domaine *AgentSwitchList* (domaine contenant la liste des agents S1,S2,S3,S4,S5). Cette croyance leur demande de prendre en charge le rôle de SlavePVC.

A la fin de cette opération, l'administrateur peut observer que tous les agents ont non seulement pris en compte le rôle de SlavePVC, mais ont aussi chargé le module de compétences SkForeATM.

L'explication est la suivante : à la demande de chargement (6), les agents n'ayant pas la compétence réclamée ont utilisé leur algorithme de recherche de compétence, et l'agent S3 leur a communiqué les modules SkSlavePVC.
Au chargement, le module de compétences pendant la phase dite de warmup, recherche le type de commutateur qu'il doit gérer ainsi que son adresse.

[5] L'administrateur aurait aussi pu utiliser le module de compétences de gestion de domaine pour attribuer automatiquement les commutateurs aux agents

Ces renseignements sont utilisés par le module pour SkSlavePVC pour réclamer à l'agent le module spécialisé correspondant, c'est-à-dire ici SkForeATM.
Les agents recherchent tout comme ils l'avaient fait pour SkSlavePVC, le module de compétences SkForeATM, puis le chargent.
Tous les agents du domaine *AgentSwitchList* sont ainsi prêts à recevoir des demandes de configurations de PVC.

Remarque:

Cette simulation nous permet d'apprécier les propriétés des agents pour le déploiement des applications. Principalement ici, la propriété de collaboration, qui permet aux agents (S1,S2,S3,S4,S5) de mettre à niveau leurs compétences. Mais aussi la propriété d'apprentissage par transfert de compétences, utilisé par l'ensemble des agents qui vont s'adresser à l'agent S3, ce dernier mettant à leur disposition ses compétences.

VI.2.5 Etapes de création d'un PVC

Cette section illustre les étapes nécessaires à la création d'un PVC. Nous utilisons pour cette description la configuration représentée sur la figure 42.

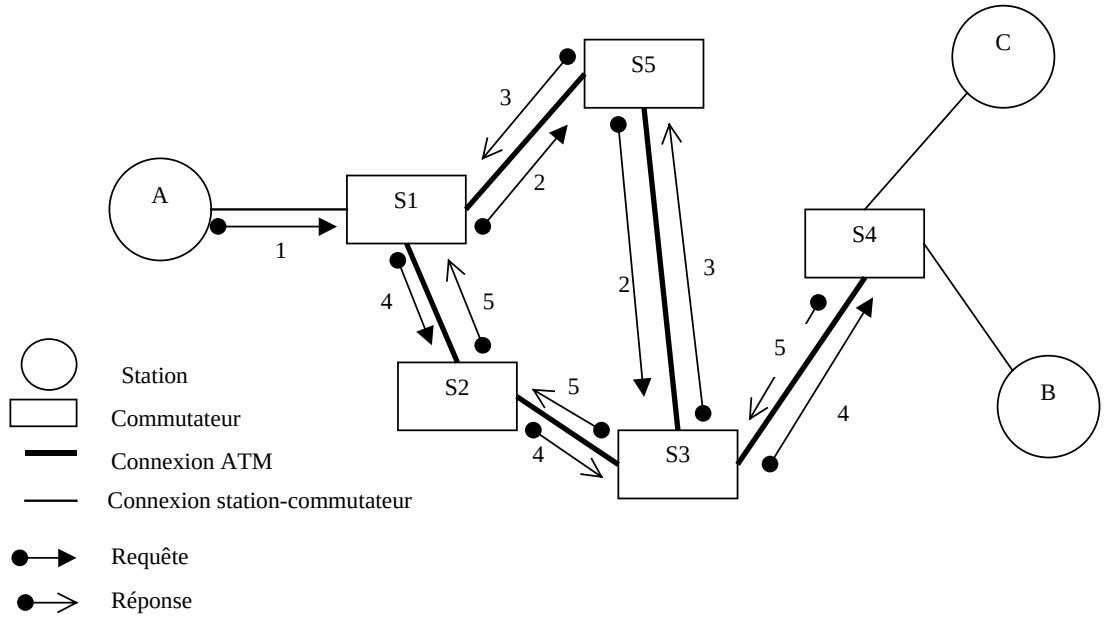


figure 42 Etablissement du PVC

1) Demande de l'utilisateur

Si l'agent de l'utilisateur (A) ne possède pas encore la compétence nécessaire à l'établissement de PVC, l'utilisateur doit demander le chargement de la compétence

SkUserPVC (voir VI.2.3.1). Une fois l'interface affichée par l'agent, l'utilisateur peut spécifier le site destination de la connexion du PVC, par exemple le site de l'agent (B). L'agent (A) demande alors à (S1) de créer le PVC, et si celui-ci n'a pas la compétence pour cela (SkMasterPVC), l'agent (A) lui demande de la récupérer.

2) Recherche de routes

Une fois les compétences chargées par l'agent (S1) la requête peut être validée par (S1). Celui-ci demande au module SkTopologie les chemins possibles pour établir le PVC . La figure () nous indique les deux chemins possibles qui sont renvoyés par SkTopologie:

1) S1,S5,S3,S4
2) S1,S2,S3,S4

3) Réservation des ressources

L'agent (S1) envoie tout d'abord une demande de réservation (2) à (S5), (S3) et (S4). Si (S3) répond qu'il n'a pas assez de ressources disponibles pour le lien S3-S4, S1 annule alors la réservation et fait une demande (4) sur le deuxième chemin proposé par SkTopology, et que nous supposons ici comme pouvant satisfaire la demande. Tous les agents renvoient donc une réponse positive (5).

4) Création du PVC

Si la réservation aboutit, l'agent (S1) envoie une demande de création aux agents concernés (S1,S2,S3,S4), et une fois qu'il a reçu toutes les confirmations émises par les agents des commutateurs, il peut alors confirmer la création du PVC en renvoyant une réponse positive à l'agent (A) de l'utilisateur.

VI.2.6 Déroulement de l'expérimentation

Pour réaliser cette expérimentation, nous avons utilisé deux commutateurs FORE dont nous disposions, un FORE Runner LE, et un FORE ASX 200. Ces commutateurs ont été connectés entre eux, et nous avons connecté deux stations sur chacun d'eux comme le montre la figure 43.

Nous avons attribué à l'agent Baltazar le FORE Runner LE, et à l'agent Douchka le FORE ASX 200. Nous rappelons qu'en raison de l'impossibilité d'installer les agents directement sur les commutateurs, nous avons installé Douchka et Baltazar sur des machines connectées aux commutateurs. Il semble qu'il y ait une réelle volonté de la part des constructeurs d'installer des machines virtuelles Java sur leurs équipements, ce qui permettra à nos agents de communiquer directement avec l'équipement sans être obligés de passer par SNMP ou CMIP.

La figure montre aussi que nous avons installé deux agents utilisateurs, Shiva et Snoopy sur deux machines connectées au commutateur FORE Runner LE.

En utilisant ces agents, les utilisateurs peuvent établir des PVC entre leur machine et n'importe quelle autre machine connectée sur les commutateurs, en l'occurrence Violette, Lys, Nelke, ou Giroflée.

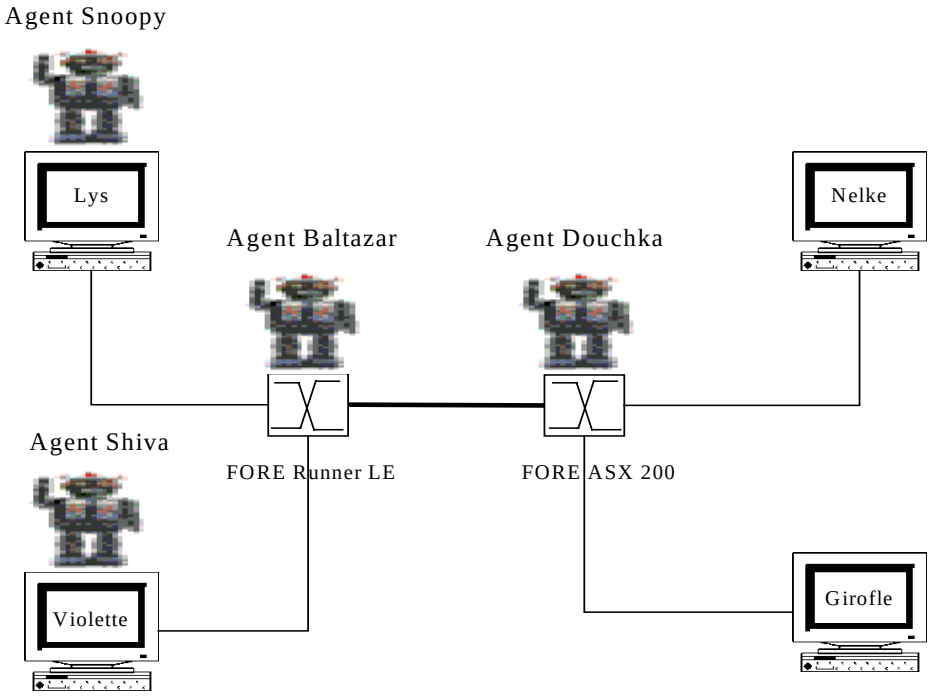


figure 43 Topologie du réseau ATM utilisée

Pour suivre les échanges de messages qui sont réalisés entre, non seulement les agents, mais aussi les modules de compétences à l'intérieur des agents, nous avons réalisé un module de compétences dont le seul rôle est de s'abonner à tous les messages concernant les PVCs et de les envoyer à un autre agent non visualisé sur la figure 44 pour affichage.
Nous avons ainsi pu tracer l'établissement d'une connexion en direct, avec l'affichage suivant:

En (1) l'agent Snoopy envoie une requête à l'agent Baltazar pour l'établissement d'un PVC. Cette requête est transmise par le cerveau au module SKMasterPVC, qui émet un message vers le module SkTopology (2), et celui-ci renvoie le message (3) en retour. Les messages (4) et (5) sont ceux émis par le module SkMasterPVC, et qui sont transmis par le cerveau de Baltazar auprès de l'agent Douchka (4) et de son propre module SkSlavePVC.

Les réservations sont confirmées par les messages (6) et (7) produits par les compétences SkSlavePVC.

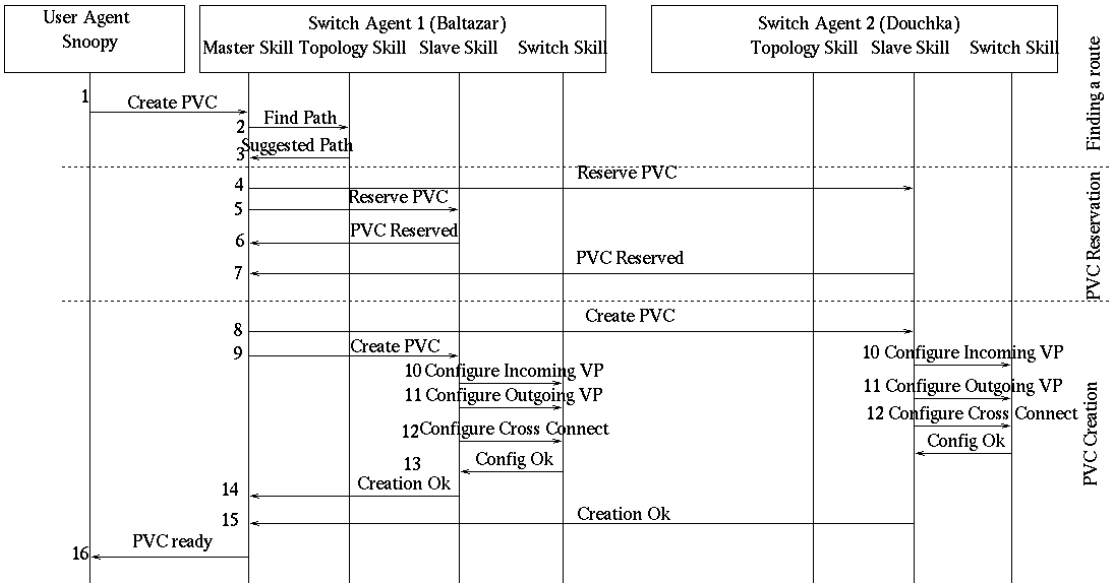


figure 44 Suivi des échanges de croyances

A la réception de ceux-ci, le module SkMasterPVC demande la création des PVC (8) et (9), ces messages étant de la même manière adressés par le cerveau respectivement à l'agent Douchka et à son module SkSlavePVC.
Les configurations sont réalisées par les compétences SkForeATM suite aux demandes (10) et (11), et le succès est confirmé par le message (13) produit par cette même compétence.
Après avoir reçu les messages (14) et (15), la compétence SkMasterPVC produit un message (16) de confirmation de création qui est transmis par le cerveau de Baltazar à l'agent Snoopy.

VI.2.7 Discussion

Cette étude de cas a été particulièrement intéressante pour plusieurs raisons. Tout d'abord parce qu'elle a permis de démontrer les avantages de l'approche distribuée promue par la technologie agent, pour les opérations de configuration. Contrairement à l'approche centralisée qui aurait exigé que l'utilisateur contacte le gestionnaire de réseaux, que ce dernier interroge les équipements, puis qu'il fasse toutes les opérations de configuration à partir de sa station de contrôle, l'approche agent n'a exigé que quelques échanges de messages entre les agents (reserve, create). Cette solution est donc insensible au problème de facteur d'échelle qui est un problème majeur pour la gestion de réseau (voir chapitre II.3.7).
En attendant que les agents puissent être installés directement sur les équipements, cet avantage se traduit essentiellement par une réduction de consommation de bande passante

auprès de la station de gestion, puisque chaque agent doit encore utiliser SNMP ou CMIP pour configurer son commutateur.
Ensuite, cette application est peu sensible à l'hétérogénéité des réseaux, car un seul module de compétences doit être modifié pour répondre aux spécificités des équipements. Cette application est donc facilement et dynamiquement extensible, car on peut ajouter à la demande de nouveaux types de commutateurs sans la modifier.
Grâce aux travaux réalisés, la configuration dynamique de PVCs ne présente plus de problèmes avec les agents DIANA, et un utilisateur aussi bien qu'un administrateur peut maintenant en quelques secondes établir un PVC sur un réseau ATM.
Par rapport à l'approche utilisant des agents mobiles, proposée par [PLBS98], les agents statiques DIANA sont beaucoup plus performants pour établir un PVC. Cette affirmation repose sur l'évaluation du temps de sérialisation (mise sous forme de paquets), du temps de transfert, du temps de désérialisation (opération inverse de la sérialisation), que nécessite un agent mobile avant de pouvoir fonctionner. Ces opérations liées au déplacement de l'agent mobile sont à répéter entre chaque commutateur, et nous devons de plus rajouter les temps d'allocation de ressources nécessaires à l'exécution du code de l'agent mobile. Les agents DIANA, une fois leurs modules de compétences chargés, sont prêts à fonctionner, et n'ont que des messages de quelques dizaines d'octets à échanger pour créer un PVC.
Une comparaison a été réalisée entre les deux approches, en se basant sur un modèle mathématique adéquat [CL00]. Les résultats montrent que s'il s'avère un peu plus facile de développer une application de configuration avec des agents mobiles, en ce qui concerne les performances et la consommation des ressources, l'avantage est nettement en faveur des agents statiques tels que les agents DIANA.

Cette solution présente aussi l'avantage de la robustesse, même sans utiliser les compétences SLD de la précédente étude de cas. En effet si un agent quelconque vient à s'arrêter, l'application continue de fonctionner. Si c'est un agent utilisateur, les autres utilisateurs ne sont pas impactés par la panne, et si c'est un agent particulier à un commutateur, les seules connexions PVC qui ne pourront être établies sont celles dont le chemin passe obligatoirement par ce commutateur.

VI.3 Analyse des résultats

Ces études de cas nous ont permis de déterminer certains avantages de l'approche agent pour le développement d'applications de gestion de réseaux.
Nous avons déjà attiré l'attention du lecteur sur la facilité avec laquelle l'approche agent permettait de développer des applications distribuées, en offrant par nature des propriétés reconnues comme importantes voire indispensables aux prochaines générations de SGR. Ces propriétés sont celles qui permettent de diminuer ou de résoudre les problèmes identifiés au chapitre II.3, c'est-à-dire principalement la délégation et la coopération.

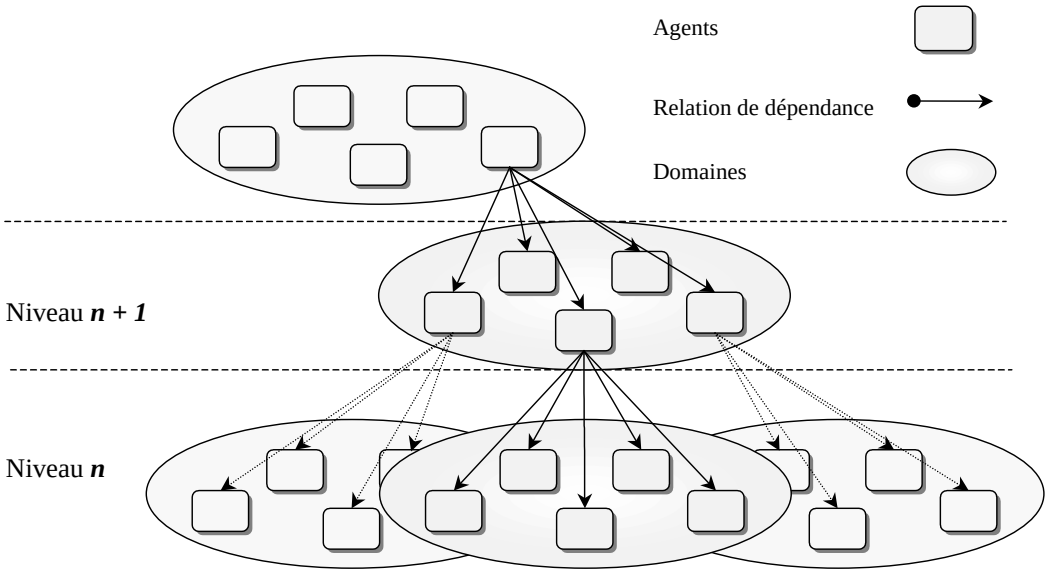


figure 45 Contrôle de l'adaptation

Mais nous avons découvert d'autres avantages, qui sont issus de l'utilisation conjointe de ces propriétés, ou de l'architecture même des agents DIANA.
De la première étude de cas, nous pouvons retenir comme un avantage important, la ***propriété d'adaptation*** du système. Obtenue grâce au développement d'un module de gestion de domaine et de l'implémentation de l'algorithme SLD, cette propriété nous permet d'entrevoir la puissance d'un système agent, pour fiabiliser les applications distribuées.
Une application distribuée, basée sur une répartition hiérarchique des tâches, peut profiter à tous niveaux de cette propriété, une défaillance d'un élément à un niveau *n*, étant détectable et donnant lieu alors à une redistribution des tâches à un niveau *n+1* (figure 45)
La propriété d'adaptation nous permet non seulement de fiabiliser automatiquement le système de gestion, mais aussi de prévoir la répartition automatique de charge (load balancing).
Il est en effet tout à fait envisageable que les agents envoient régulièrement à un agent responsable de leur domaine une information sur leur charge de travail, afin que ceux-ci puissent redistribuer les activités.

Nous avons découvert un autre avantage de l'approche agent, mais plus particulièrement évident avec l'architecture DIANA, lors de la deuxième étude de cas. Il s'agit de la possibilité de modifier dynamiquement le comportement des applications.
Nous avons eu un premier aperçu de ce phénomène avec l'expérimentation SLD, où la mise en fonction de SLD va influencer le comportement global du système. Mais il s'agit là plus particulièrement d'une adaptation à l'évolution de l'environnement que d'une réelle modification du comportement.
Lors de la deuxième étude de cas pour pouvoir visualiser les messages échangés entre les agents, ainsi que ceux échangés dans un même agent entre les modules de compétences,

nous avons rapidement développé un module d'espionnage (SkSpy), dont la seule tâche était de transmettre les messages concernant l'établissement de PVC à un autre agent en charge de l'affichage. Sa conception était particulièrement aisée puisque celui n'exige aucune précondition (voir chapitre IV.2.4.2) et ne nécessite que la déclaration des modèles de messages utilisés par les compétences impliquées dans le traitement de l'établissement des PVC.
Une mission d'espionnage a donc été attribuée à tous les agents participant à l'expérimentation.

Ce que nous avons remarqué alors, c'est que nous pouvions utiliser un tel module pour intercepter une croyance produite par une compétence quelconque, et modifier son contenu.
Le résultat d'une telle opération est que nous pouvons alors insérer facilement de nouvelles fonctionnalités dans une application, sans avoir besoin de la modifier.

VI.3.1 Agent et comportement

Ici nous étudions les mécanismes dit *souples* permettant la modification dynamique du comportement d'un agent et par propagation, du système. La modification du comportement est une modification programmée dans le sens où elle ne fait pas intervenir des mécanismes d'apprentissage au sens de l'intelligence artificielle. Il s'agit en fait de rajouter du code (nouvelle compétence) qui sera utilisé par l'agent sans qu'il ait eu une connaissance préalable de celui-ci. Nous pouvons comparer cette approche à celle d'un système délibératif à base de règles, dans lequel on introduit de nouvelles règles qui s'intercalent dans la logique du programme et sont prises en compte par le moteur d'inférence.

VI.3.2 Comportement adaptatif

Lorsqu'une application de gestion est développée, les différents modules de compétences sont identifiés puis les comportements sont élaborés. Lorsque l'agent découvre les modules, il récupère les renseignements fournis par le module pour mettre à jour ces bases de compétences et de connaissances. Ceci permet la modélisation de comportements de type tâches (voir chapitre IV).

Pour mieux comprendre le mécanisme lié au comportement de l'agent, nous allons examiner les étapes effectuées par l'agent à la réception d'une demande d'exécution d'une opération (tâche ou activité).

Lorsqu'une opération est invoquée par un agent ou un utilisateur, l'agent responsable de l'exécution d'une opération effectue le travail suivant :

- Vérification de l'existence de la tâche dans sa base de compétences. Si la tâche n'est pas identifiée, l'agent peut soit interroger les agents dont il

dépend pour leur demander un transfert de compétence, soit refuser la tâche (comportement agent).

- Quand il a la compétence correspondante pour l'exécution de cette tâche, il vérifie alors les préconditions nécessaires à l'exécution de l'opération. Si les préconditions ne sont pas satisfaites, il peut soit chercher à créer ces préconditions (*acheive*) soit refuser d'exécuter l'opération.
- Lorsque les préconditions sont satisfaites, il transmet au module de compétences correspondant le contenu de l'invocation.
- L'opération est alors considérée par l'agent comme étant activée (statut: *en cours*)

Si l'opération est une activité, celle-ci nécessite des informations pour se maintenir. Ces informations représentent un ensemble de croyances qui est déterminé par la relation **U**:

$$\mathbf{U}(o_i) = \{u_1,..u_n\} \qquad \text{où } o_i \text{ représente l'opération en cours}$$

L'agent utilise la relation inverse qui veut que:

$$\forall\, u_i \in U\,, \exists\, o_i \in O \,/\, u_i \in \mathbf{U}(o_i)$$

pour transmettre au module de compétences de o_i une notification sur les opérations ayant affecté u_i .

Maintenant en considérant les relations suivantes:

$$\mathbf{R}(o_1) = u_1,\ \mathbf{R}(o_2) = u_2,\ \ldots\ ,\ \mathbf{R}(o_n) = u_n$$

qui expriment que les croyances sont le résultat de l'exécution d'opérations, la modification du comportement de l'application peut se faire en ajoutant une autre relation

$$(\mathbf{R}(o_1) \mid \mathbf{R}(\gamma)) = u_1,\ \mathbf{R}(o_2) = u_2,\ \ldots\ ,\ \mathbf{R}(o_n) = u_n$$

$\mathbf{R}(\gamma)$ est une nouvelle opération produisant ou modifiant la croyance u_1. Le résultat produit par cette nouvelle relation est que l'application originale, dont le comportement était défini par les relations entre les croyances produites et consommées, se trouve modifiée par l'introduction de cette nouvelle opération γ.

A tout moment, alors que l'agent est en fonction, nous pouvons insérer une application complémentaire qui va modifier les croyances produites par l'exécution d'une tâche.
Par exemple dans l'étude de cas sur la fiabilité du système multi-agents, nous avons pu observer une modification du comportement de l'agent chargé de gérer la distribution des tâches de monitoring après l'introduction du module SLD et l'activation de la tâche de surveillance des agents.

Dans cette étude de cas, une nouvelle information a été créée par le module SLD sur la fiabilité des agents, et l'information sur la liste des agents disponibles a été prise en charge par le module SLD qui l'a alors modifiée (figure 46)

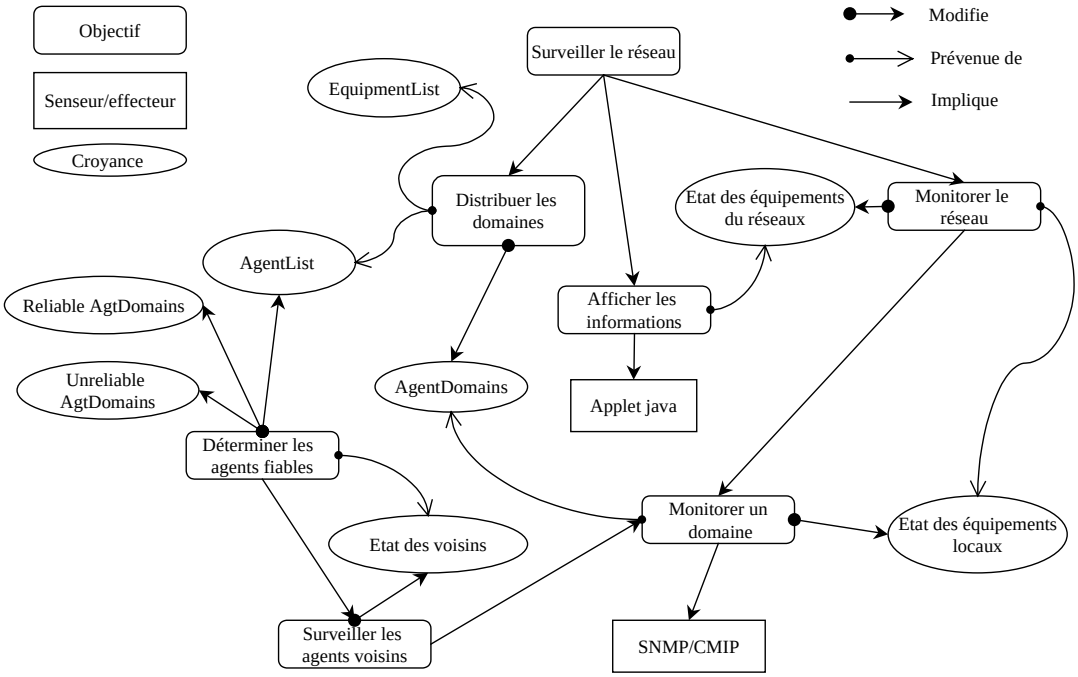


figure 46 Insertion de l'application SLD dans le système de surveillance du réseau

Sur cette figure, l'objectif qui a été donné à l'agent de déterminer la fiabilité des agents se traduit par une modification de *Agentlist* qui est une croyance utilisée pour répondre à l'objectif de distribution de domaines.

Le comportement observé pendant le déroulement du scénario de l'étude de cas est bien une redistribution des domaines.

VI.3.3 Modification du comportement

Comme nous l'avons fait remarquer au début de cette analyse, un deuxième mécanisme de modification de comportements nous est apparu comme très intéressant.

Nous avons déjà indiqué que le comportement de l'agent est déterminé par les relations entre les croyances produites et consommées par les opérations en cours dans un agent. La modification du comportement va donc passer par la modification de ces relations. Nous venons aussi de rappeler les effets obtenus par l'ajout d'une nouvelle compétence dans le cadre de la première étude de cas, où une opération appartenant à cette nouvelle compétence modifie la liste des agents disponibles pour l'application de gestion de fautes.

Dans la deuxième étude de cas, le module espion allait quant à lui être averti par le cerveau de l'agent que des demandes de création, de modification, ou de suppression de croyances étaient produites par les modules de compétences.
Il est alors possible pour lui de modifier, de retenir ou de supprimer cette croyance en fonction de ses besoins.
Pour comprendre l'intérêt que présente ces possibilités, examinons les deux exemples suivants.

Application de facturation .

L'opérateur veut mettre en place une nouvelle politique commerciale, et décide de proposer différents abonnements. Il possède déjà son application d'établissement de PVC que nous avons présentée dans la deuxième étude de cas. Il n'est pas nécessaire de modifier celle-ci, car l'opérateur peut demander le développement d'une application supplémentaire qui va intercepter toutes demandes émanant de ses clients pour, en fonction de leur identité et de leur contrat d'abonnement, refuser ou comptabiliser les connexions. Pour ce faire, le concepteur de l'application utilise les croyances produites par l'agent utilisateur lors de la demande de connexion ou de déconnexion.

Application d'optimisation

De la même manière, si l'opérateur veut optimiser ses ressources en fonction du coût des ressources utilisées, il peut demander la mise en place d'une autre application qui va se *greffer* sur les premières pour retourner une estimation du coût en fonction de la route. L'implémentation d'une telle application peut se faire par exemple en interceptant la réponse à la demande de route émise par le module *SkTopology*, l'application interrogeant alors les agents sur les différentes routes pour avoir le meilleur coût.

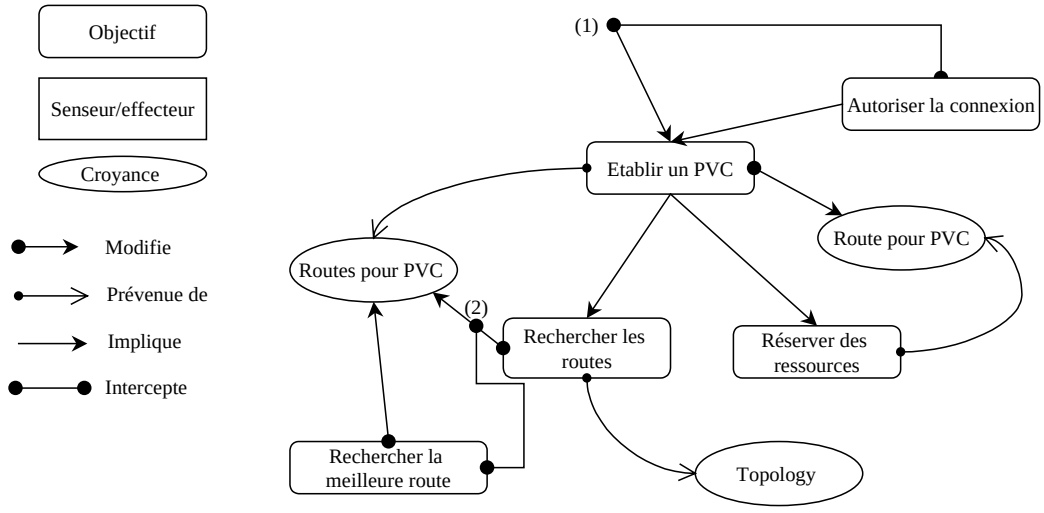


figure 47 Modification du comportement par interception des croyances

La figure 47 montre en (1) l'interception de la demande de connexion PVC émise par le client, pour activer l'objectif de recherche d'autorisation de connexion. La flèche de type implication qui ressort de l'objectif *autoriser la connexion* vers l'objectif é*tablir un PVC* indique que la demande ne sera réellement transmise à ce dernier qu'une fois que l'objectif autorisation sera terminé.
En (2) l'interception de la modification de la croyance *routes pour PVC* indique que cette croyance sera modifiée par l'objectif *rechercher la meilleure route*.

L'interception de ces croyances est possible si elles font partie des ensembles *U* (confer chapitre V.4) des modules de compétences associés à la nouvelle application. L'agent transmet alors les croyances à ces modules de compétences (sous réserve bien entendu que les opérations correspondantes aient été préalablement activées), qui ont alors la possibilité de les lire, les modifier, les bloquer ou les relâcher.

Dans le cadre d'une utilisation normale, les croyances sont transmises par notification d'un module à un autre module de compétences par le cerveau de l'agent, chaque module ne faisant que copier cette croyance, et l'utilisant par la suite (chaque module est exécuté dans un thread différent). Mais ici nous pouvons comprendre qu'il y ait un intérêt à "retenir" une information avant de la transmettre, et même quelquefois à la modifier (changement de l'ordre des routes).

VI.3.4 Le couplage lâche

Nous considérons qu'il y a là un relâchement des liens entre les composants de base d'une application, et nous pouvons considérer que la conception d'un SMA avec les agents DIANA implique le développement de composants reliés par *couplages lâches*. Cette notion est particulièrement importante pour les applications de gestion de réseaux.
Comme nous l’avons déjà rappelé, les SGR sont connus pour leur manque de flexibilité au niveau opérationnel. Il est en effet quasi impossible de leur rajouter dynamiquement de nouvelles fonctionnalités. Chaque nouvelle fonctionnalité demande une étude approfondie pour vérifier que l’architecture du SGR puisse la supporter, puis une phase complète de design, de codage, d'intégration et de test avant de pouvoir fournir une nouvelle release ou version du SGR. Un inconvénient secondaire est que le gestionnaire de réseau est fortement dépendant de son fournisseur, et ne peut faire évoluer son SGR en toute liberté.
L’architecture d’agent que nous avons développée présente l’avantage de supporter dynamiquement l’ajout de nouvelles fonctionnalités au travers des modules de compétences, et présente de ce fait une contribution à l’évolution des technologies distribuées.
En comparaison, l’approche CORBA implique que chaque objet doit communiquer au travers de son interface spécifique (décrite en IDL). Cette approche apparaît comme une contrainte rendant a priori impossible l’évolution dynamique des applications par insertion de comportements. L'approche ODP de son côté prévoit que les seules interactions possibles entre objets sont celles décrites par leur interface. Néanmoins, que ce soit pour ODP ou CORBA, c’est pendant la phase de conception de l’application que

doit être prévue la possibilité de coupler (« bind ») les objets avec un gestionnaire de messages dont le rôle serait de gérer les interactions entre les objets. Ces contraintes ont été identifiées depuis plusieurs années [RUM92], et des travaux ont été entrepris pour séparer les interactions des objets eux-mêmes [FRO94], l'objectif étant dans certains cas identique au nôtre lorsqu'il s'agit de capturer leurs messages afin de modifier leurs comportements [And99]. Un langage [BDF98] et un gestionnaire de communications ont été développés, permettant une programmation orientée interaction.

L'approche agent et la technologie que nous avons développée dans le projet DIANA se distingue de ces technologies et de ces travaux par, dans un premier temps, le fait que les modules de compétences qui encapsulent les comportements de l'agent utilisent tous la même interface de communication. La seconde différence est que c'est l'agent lui-même qui gère les échanges de croyances et donc les interactions entre les composants en utilisant un mécanisme de pattern-matching. Il est alors possible de demander à l'agent de modifier les règles de distribution des informations qui sont par défaut des abonnements

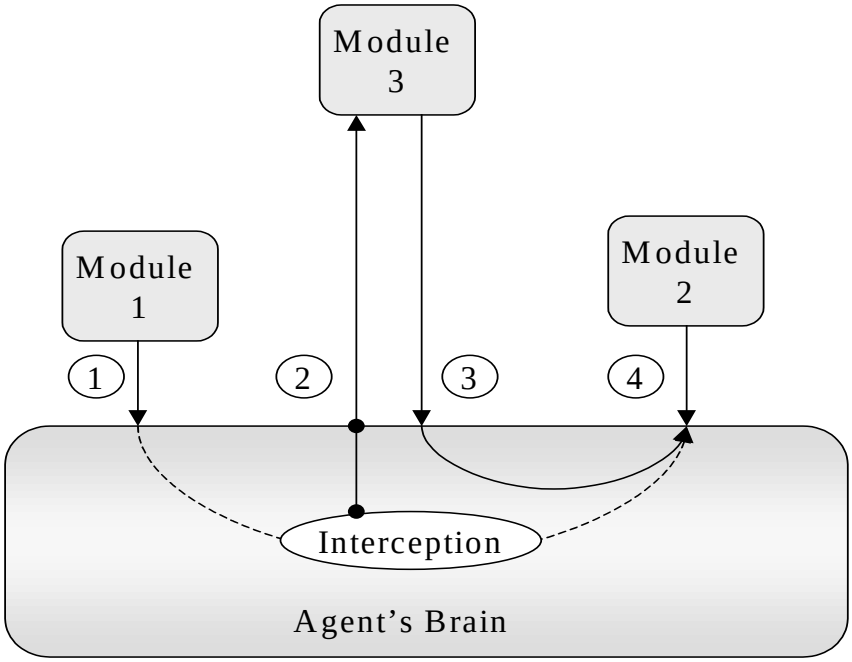


figure 48 Interception des croyances

automatiquement réalisés lors du déclenchement d'une activité.

La figure 48 montre une application agent développée autour de 2 modules (1 et 2) dont le comportement est modifié par le chargement d'un *module 3*. En effet le *module 3* utilise les messages générés par le *module 1* à l'intention du *module 2* et les modifie en fonction de ses objectifs avant de les transmettre au *module 2*.

VI.3.5 Conclusion

L'analyse des résultats obtenus dans ces études de cas montre que la possibilité de modifier les applications existantes, et qui plus est pendant la phase opérationnelle, est un avantage indéniable de l'approche agent telle que nous la défendons avec l'architecture générique DIANA.
Contrairement à l'approche objet qui impose un couplage encore relativement fort des classes, par l'obligation qui est faite de spécifier toutes les relations entre les classes et les méthodes avant même le codage, l'approche agent permet de son côté un couplage lâche des composants de l'application. Le couplage est dit *lâche*, car il est réalisé dynamiquement suivant les circonstances par l'agent, et il ne nécessite pas la connaissance préalable de modules cibles, contrairement à l'utilisation d'autres mécanismes comme les DLL (Dynamic Link Library) utilisées par Windows de Microsoft ou les librairies dynamiques sous Unix.

Ces études de cas nous ont permis de valider l'utilisation des propriétés agent comme la délégation, la coopération, l'apprentissage par transfert de connaissances et de compétences. Grâce à ces propriétés, il nous a été possible de développer des systèmes multi-agents flexibles et extensibles car n'imposant pas de modification, même lorsque l'on modifie le nombre d'agents ou que la taille du réseau à gérer varie.
Du fait même de la distribution de l'intelligence, ces applications présentent naturellement une réactivité supérieure à tout système centralisé, car le traitement des informations se fait au plus près des équipements ou des agents producteurs de ces informations.
Nous avons pu constater l'insensibilité de ces applications au problème de l'hétérogénéité des équipements en raison de la décorrélation qui est faite entre les compétences de gestion et les compétences propres aux équipements.

Enfin nous avons vu dans ce chapitre qu'il était possible de modifier dynamiquement le comportement d'un SMA au travers de ses applications, non seulement en intervenant sur le contenu d'une croyance mais aussi sur la distribution des croyances par les agents eux-mêmes.

Tous ces avantages sont importants pour le développement d'applications de SGR, car ces dernières sont confrontées à l'évolution permanente de leur environnement aussi bien matériel que logiciel. La réutilisabilité des composants d'une application, apportée par l'approche objet est poussée à une réutilisabilité complète de l'application qui peut évoluer facilement sans modification de son code initial, mais uniquement par rajout de nouveaux composants, ou de nouvelles applications complémentaires.

VII Conclusion

Les systèmes de gestion de réseaux traditionnels sont des applications lourdes, rigides, complexes, et donc peu adaptées à l'évolution rapide des technologies et à la demande très forte des utilisateurs pour l'intégration de ces technologies et la mise en place de nouveaux services. Ce constat à l'origine de cette thèse nous a incité à nous intéresser au paradigme agent intelligent, qui semblait le mieux adapté pour proposer une alternative au développement classique des systèmes de gestion.
Dans ce chapitre nous rappelons dans un premier temps la démarche que nous avons suivie dans l'objectif de démontrer l'intérêt de cette approche. Nous décrivons ensuite les résultats que nous avons obtenus, aussi bien en ce qui concerne le renforcement du paradigme agent que son intérêt pour la gestion de réseaux. Puis nous présentons les risques qui sont associés à cette nouvelle technologie; et nous proposons des axes de recherche dans la continuité des travaux que nous avons entrepris.

VII.1 Rappel du contexte

Après avoir rappelé les besoins en gestion de réseaux, nous avons revu les principaux standards utilisés pour le développement de SGR. Nous avons constaté au chapitre II que non seulement les plates-formes développées autour de ces standards ne répondent pas aux attentes des utilisateurs, mais que la longueur des cycles de développement et de standardisation ne leur permettent pas de suivre l'évolution rapide des technologies entrant dans la composition des réseaux. La conséquence de ce décalage de plus en plus important est que les plates-formes de gestion de réseaux sont peu adaptées aux nouvelles exigences des utilisateurs concernant la mise en service dans des délais très courts de ces nouveaux équipements et de ces nouveaux services.

L'approche centralisée, les faibles niveaux d'abstraction utilisés par ces plates-formes, combinés avec l'hétérogénéité des équipements, rendent les solutions complexes, difficiles à utiliser, et sont autant de justifications pour la recherche de nouveaux concepts.

Pour mieux comprendre les objectifs que devrait atteindre un système idéal, nous avons identifié les principaux problèmes non encore résolus auxquels devraient faire face les nouveaux SGR. Puis nous avons recherché les concepts existants ou en cours d'étude qui seraient à même de résoudre ces problèmes. Nous nous sommes basés pour cela sur les études existantes et leur examen nous a permis de dégager les tendances et les orientations des recherches.
De celles-ci ressortent principalement l'importance de la distribution aussi bien de l'information, de l'intelligence que des systèmes, et l'apparition et l'utilisation de nouveaux niveaux d'abstraction comme les services, rôles ou politiques.
Nous nous sommes aussi intéressés à l'évolution des technologies utilisées dans le cadre de la gestion de réseaux afin de mesurer la prise en compte de ces concepts dans les nouvelles technologies.
Nous avons remarqué que si des approches innovantes comme les réseaux actifs étaient l'objet de quelques recherches, la majorité des travaux concernaient l'évolution du paradigme objet. Parmi ces travaux, la mobilité du code et les agents mobiles sont les dernières propositions d'évolution du monde objet, dont l'objectif est de rendre indépendant et autonome chaque objet. Mais nous avons souligné que si ces approches pouvaient être considérées comme une avancée vers la distribution du traitement de l'information, elles présentaient l'inconvénient de ne pouvoir intégrer l'ensemble des concepts que nous avions identifiés comme importants dans le cadre d'une solution de gestion de réseaux, et justifiaient donc la recherche de nouveaux paradigmes plus appropriés.
L'examen des besoins exprimés comme la flexibilité, l'abstraction, couplés avec les concepts identifiés comme l'autonomie, la délégation, la coopération, nous ont poussé assez naturellement à rechercher une solution globale dans le paradigme agent intelligent, que [Woo98] considère comme le seul paradigme important ayant émergé depuis le paradigme objet.

VII.2 Contributions

VII.2.1 Identification d'un paradigme agent

Pour valider notre hypothèse, il nous a fallu commencer par identifier ce paradigme. Nous nous sommes en effet rapidement rendus compte à la lecture des publications de plus en plus nombreuses sur le sujet, qu'une grande confusion existait dans la définition de l'agent, ou plutôt qu'aucune définition ne faisait l'unanimité. L'explication que nous avons donnée à cet état de fait est la diversité des origines de ce paradigme.

Suivant l'exemple de nombreux chercheurs du domaine, nous ne nous sommes pas engagés dans une définition pouvant prêter à polémique et nous avons préféré définir ce paradigme par les propriétés qui y sont attachées. Nous avons pu constater que mêmes les propriétés étaient quelquefois l'objet de vives controverses, aussi en l'absence de consensus, nous avons choisi les définitions qui nous paraissaient les plus appropriées.
De notre étude sur ce paradigme, nous avons retiré que s'il semblait se dégager quelques architectures principales que nous avons présentées au chapitre III, il était par contre impossible de trouver une technologie agent intégrant les propriétés dont nous avions besoin pour valider ce paradigme comme solution à nos problèmes de gestion de réseaux.
Nous avons attiré l'attention du lecteur sur la confusion qui était souvent faite entre les agents mobiles et les agents intelligents, les premiers étant très souvent uniquement des unités de code mobile, et exceptionnellement des programmes capables de choisir le lieu de leur exécution. Du fait de leur nécessité de se déplacer, ces agents mobiles ne peuvent posséder un système délibératif conséquent, ni même effectuer un travail coopératif, ce qui à notre point de vue est rédhibitoire à l'appellation agent intelligent.

Nous avons cité des travaux mettant en avant l'utilisation d'agents intelligents dont seules certaines propriétés ont été implémentées pour résoudre des problèmes en gestion de réseaux, mais nous n'avons pu identifier de technologie suffisamment élaborée pour valider notre hypothèse. Aussi nous nous sommes penchés sur l'étude et la réalisation d'une architecture originale d'agent intelligent, possédant les propriétés dont nous avions besoin.

VII.2.2 Développement d'une architecture agent

L'architecture que nous avons proposée devait permettre le développement de systèmes utilisant les principales techniques que nous avions référencées comme connues pour traiter les problèmes de taille et de complexité des systèmes d'information, c'est-à-dire:

- la modularité
- la distribution
- l'abstraction
- l'intelligence

De plus, nous avons eu comme préoccupation lors de la conception de cette architecture le fait de pouvoir offrir une généricité dynamique, c'est-à-dire la possibilité d'accroître les compétences des agents au cours de leur utilisation.
Une des approches permettant d'atteindre cet objectif est l'utilisation d'un des mécanismes d'apprentissage issus du domaine de l'intelligence artificielle. Or la littérature sur ce domaine ne nous laissait pas entrevoir la possibilité de créer facilement des systèmes évolutifs adaptés aux problèmes de la gestion de réseaux. Nous nous sommes donc intéressés aux systèmes à base de modules, proches de la notion de JAVA-Beans apparue peu avant, tel que nous l'avons décrit dans le chapitre III. Cela nous a permis de créer des agents dont le comportement peut évoluer dynamiquement en

fonction des modules de compétences qu'ils chargent. Ces modules peuvent être transmis d'un agent à un autre, créant ainsi un apprentissage par transfert de compétences.
Cette propriété permet de mettre rapidement en fonction une nouvelle politique de gestion, comme par exemple la mise à l'écart de tout agent peu fiable comme dans la première étude de cas. Ce transfert de compétences est à rapprocher des études sur la gestion par délégation utilisant le transfert de scripts [Gol95], et permet aux gestionnaires de réseaux non seulement d'accroître les capacités de traitement de leurs systèmes de gestion de réseaux en terme de qualité (ajout de compétences nouvelles) mais aussi en terme d'échelle (scalabilité) en leur permettant de répartir les charges de travail sur les différents équipements de leurs réseaux.

VII.2.3 Validation des propriétés agent

Nous avons vu au chapitre IV que certains concepts et les propriétés agent correspondantes étaient plus intéressants à retenir dans le cadre particulier d'applications de gestion de réseaux. L'architecture que nous avons développée tient compte de cette analyse, et nous avons pu au cours de nos études de cas vérifier qu'il était possible d'ajouter facilement de nouvelles propriétés à celles intégrées directement au sein des agents.
Dans les propriétés de base d'un agent, l'**autonomie** et la **délibération,** qui permettent à un agent de choisir les moyens pour atteindre un objectif fixé, sont utilisables grâce à des instructions comme *achieve*. Nous avons fait remarquer à ce sujet que contrairement à une approche qui aurait été menée en intelligence artificielle, les agents DIANA ne proposent qu'une autonomie bridée. Il est en effet encore difficile de concevoir une application de gestion dont chaque entité pourrait décider ou non d'effectuer un travail qui lui serait demandé. Les études de cas que nous avons d'ailleurs développées proposent une approche directive de la gestion, les agents n'étant pas autorisés à refuser une requête émanant d'un autre agent. Nous avons en effet privilégié une distribution hiérarchique de la gestion où les agents sont organisés et ont des responsabilités données par le gestionnaire de réseaux. L'autonomie est dans ces cas-là exprimée par la faculté qu'a l'agent, et par conséquent le SMA, à fonctionner sans intervention du gestionnaire de réseaux.
La propriété de **communication,** qui est la propriété fondamentale de tout système multi-agents, a bien sûr aussi été intégrée dans cette architecture. De même pour la propriété de **délégation** qui a été utilisée plusieurs fois dans les études de cas, comme dans la deuxième étude où l'agent en charge de l'établissement du PVC délègue les opérations de configuration aux agents en charge des équipements. Nous avons considéré que le transfert de compétences était la manifestation d'une capacité d'**apprentissage** plus adaptée dans la plupart des cas en gestion de réseaux, que l'apprentissage proposé dans le domaine de l'intelligence artificielle. C'est donc cette propriété qui a été retenue pour le fonctionnement des agents.

Si nous venons de voir que la délégation et l'apprentissage étaient des propriétés intégrées par l'architecture DIANA, d'autres propriétés agent ont elles aussi été développées dans le cadre des études de cas.

Par exemple la **coopération,** que nous avons testée dans nos deux études de cas. Dans la première étude, les agents s'échangent des informations permettant de déterminer l'état global du système et de redistribuer les tâches lorsqu'une défaillance d'une partie est découverte.
Nous pouvons aussi prétendre ici à un certain niveau de **proactivité,** car l'agent prenant la décision de redistribuer les tâches le fait en comparant les résultats produits par chaque agent, et peut de ce fait prendre sa décision dès les premiers signes de défaillance de l'un des agents, anticipant ainsi l'évolution des symptômes.

L'utilisation de systèmes distribués et coopératifs dont les entités ont une autonomie même relative soulève la question de la coordination des activités. Pour nos deux études de cas, nous avons opté pour la solution consistant à charger un agent de contrôler la coordination de l'ensemble des actions menées en coopération. C'est pour cela que dans le premier cas, un seul agent est chargé de centraliser les renseignements produits par les autres sur l'état de leurs voisins. Dans le deuxième cas, c'est aussi un seul agent, celui en charge de l'établissement de la connexion, qui va coordonner les différentes actions de configuration. D'autres solutions comme le «contract net protocol» sont proposées dans le monde des agents pour gérer la coopération et la coordination. Mais ces solutions nécessitent généralement un nombre important d'échanges d'informations entre chaque agent, pouvant ainsi porter préjudice aux performances recherchées, car chaque agent négocie lui-même les tâches qu'il accepte.

VII.2.4 Apports pour les systèmes de gestion

Nous revenons brièvement ici sur les résultats concernant l'intérêt de l'approche agent intelligent pour la gestion de réseaux, que nous avons établi lors des études de cas.
Pour cela nous associons les problèmes que nous avions identifiés aux résultats que nous avons obtenus:

- **Hétérogénéité**. Nous avons vu lors de la deuxième étude de cas que les agents pouvaient utiliser des informations ayant un niveau d'abstraction suffisant pour rendre l'application quasi insensible à l'hétérogénéité des équipements. Les agents DIANA peuvent utiliser des compétences spécialisées en fonction des équipements, des applications et des protocoles, avec lesquels ils doivent inter-agir, et sont donc bien une solution au problème d'hétérogénéité (chapitre VI.2.7).
- **Complexité**. L'utilisation de modules de compétences, la distribution dynamique des traitements et donc de l'intelligence, la possibilité d'utiliser différents niveaux d'abstraction sont un premier pas vers la réduction de la complexité des systèmes. De plus l'approche organisationnelle que nous avons présentée dans le chapitre V permet d'affirmer que l'approche agent contribue à la diminution de la complexité de la conception des systèmes de gestion.
- **Coût**. Développés en Java, donc facilement portables et maintenables, les systèmes développés autour des agents intelligents sont donc économiques à

produire. L'économie est d'autant plus importante que le système de gestion est simplifié du fait même de la distribution de l'intelligence (chapitre II.3.3).

- **Délai**. Développant des propriétés d'adaptation et montrant une grande flexibilité, les délais de développement, d'intégration et de mise en place sont extrêmement faibles par rapport à ceux des approches traditionnelles.
- **Flexibilité**. Nous avons vu en conclusion des études de cas qu'il était possible de modifier les comportements des applications de gestion par ajout de nouvelles compétences et fonctionnalités. Grâce à cet apport, les applications de gestion de réseaux peuvent s'enrichir de nouveaux services sans avoir besoin de repenser l'architecture du système.
- **Fiabilité**. La fiabilité des systèmes a été l'objet de la première étude de cas où nous avons pu démontrer les possibilités qu'offrent les systèmes multi-agents en terme de stabilité et de fiabilité des applications. Nous avons aussi discuté de l'avantage de l'approche agent où une partie du réseau peut être déconnecté sans interruption majeure de la gestion du réseau déconnecté.
- **Extensibilité**. Nous avons vu que l'extensibilité des systèmes de gestion de réseaux traditionnels passaient par la duplication des systèmes eux-mêmes. La solution la plus défendue passe par la délégation des traitements et des opérations, qui est une des propriétés les plus implémentées dans les technologies agent.

Nous n'avons pas traité dans les travaux présentés dans cette thèse des problèmes de la QoS ni de la sécurité dans la gestion de réseaux. L'utilisation des agents pour la gestion de la QoS a déjà donné lieu à un travail intéressant [Oli98], où l'apport de l'approche agent a déjà été démontré. En ce qui concerne les problèmes de la gestion de la sécurité par les agents, des travaux sont en cours au sein de l'Institut Eurécom dans le cadre d'une autre thèse.
Si nous avons pu démontrer que l'approche agent adressait correctement les problèmes de gestion de réseaux que nous avions identifiés, nous avons pu démontrer plus particulièrement l'intérêt de l'approche agent pour développer un système de gestion fiable et présentant de fortes capacités d'adaptation.

VII.2.5 Fondement d'une technologie agent

Lorsque nous avons abordé l'étude du paradigme agent, nous avons constaté qu'il est actuellement impossible de considérer la technologie agent comme une véritable technologie. En effet malgré une certaine unanimité sur les propriétés que doivent posséder les agents, la diversité des agents développés rend impossible tout effort de standardisation. Nous pouvons d'ailleurs constater que le seul standard agent, résultant des efforts de la FIPA, concerne ACL, un langage de communication inter-agents. Ce standard semble d'ailleurs assez peu suivi au regard du nombre de publications où c'est KQML ou d'autres langages qui sont utilisés.
D'un autre côté, une nouvelle technologie ne peut s'imposer que si elle représente des avantages indéniables par rapport aux technologies existantes. L'analyse des résultats de nos études de cas nous montre qu'au delà des propriétés reconnues des agents, ce sont les

propriétés d'adaptation et de modification de comportements qui sont les plus intéressantes pour affirmer l'intérêt de la technologie agent.
Ces propriétés sont les conséquences du couplage lâche entre les composants d'une application, et du type de gestion des informations proposé par l'approche agent.
Nous avons pu montrer que tout en conservant les avantages de la modularité des composants tel que le propose l'approche objet, l'approche agent ouvre, avec le relâchement du couplage inter-composants et même intra-composants, la voie vers la manipulation d'abstractions de plus haut niveau tel que le comportement.

VII.3 Risques et axes de recherches

Si nous avons fait ressortir dans cette thèse les éléments essentiels à la création d'une nouvelle technologie agent qui soit capable de traiter les problèmes de la gestion de réseaux, cette technologie présente encore des risques qui sont susceptibles de freiner son développement.

Tout d'abord une première objection importante que nous pouvons faire à l'emploi de la technologie agent est son manque de maturité. Il n'existe pas pour l'instant de vues communes ni sur la définition des agents ni sur l'architecture de ceux-ci, et encore moins sur leurs propriétés. Par exemple, de nombreux travaux ont été récemment effectués sur la mobilité des agents, mais leur contenu technique se rapproche plus de l'objet mobile que d'une réelle approche agent. Sans cette maturité il est impossible de créer des standards de développement, et sans standards peu d'entreprises se risqueraient à investir dans ce qui ressemblerait alors à une aventure risquée. Il existe néanmoins une exception que nous avons déjà citée à ce constat, c'est la création de l'organisation FIPA dont l'objectif est de proposer des standards pour le développement du paradigme agent, mais dont le seul réel succès pourrait être la proposition d'ACL (Agent Communication Language) dérivé mais encore très proche de KQML (Knowledge Query Meta Language). Ce manque de maturité se constate aussi par le nombre de problèmes encore ouverts, comme la résolution efficace de problèmes de planification dans un système distribué.

Si nous avons pu démontrer la possibilité de fiabiliser le fonctionnement d'un système multi-agents, il reste encore à démontrer que nous pouvons faire confiance à un système délibératif. En effet l'approche traditionnelle de la gestion de réseaux repose sur le déterminisme des applications intégrées ou associées au système de gestion. Toute modification de configuration repose sur un ensemble d'ordres ordonnés et synchronisés qui sont exécutés à la demande de l'administrateur du réseau. Les implications de la modification des configurations sont évaluées par ce même administrateur, et il semble pour l'instant difficile de laisser un agent ou le système multi-agents décider lui-même des actions à entreprendre sans faire au préalable confiance à ses capacités de raisonnement. Ici la décentralisation, la distribution de la gestion qui représente un des atouts essentiels de l'approche agent, peut apparaître un handicap si chaque agent n'a qu'une vue restreinte de son environnement et des conséquences de ses actions.

Toujours à propos du contrôle d’un système multi-agents, d’autres problèmes comme celui de la gestion de la résolution de demandes conflictuelles sont des voies de recherches à explorer.

Bibliographie

AC96

Yehuda Afek and Yoram Cohen.
A generic web-based network management application.
In *Proceedings of the Distributed Systems and Operations Management Workshop - DSOM'96,* L'Aquila, Italy, October 1996.

ACFF99

Laurent Andrey, Isabelle Chrisment, Olivier Festor, and Eric Fleury.
Supervision et contrôle dans les réseaux actifs: Une nécessité à la mise en oeuvre et au déploiement dans les réseaux de télécommunication.
In Omar Cherkaoui, editor, *GRES'99 3ème Colloque Francophone sur la Gestion de Réseaux et de Services*, pages 23-33, Montréal Canada, Juin 1999.

AD98

Mathias Astier and Bertrand Duval.
Optimized monitoring services for network management applications.
Rapport de Stage, December 1998.
Institut Eurécom.

AFY99

Laurent Andrey, Olivier Festor, and Nizar Ben Youssef.
Intégration wbem/rgt. conception et mise en oeuvre dans les environnements modérés et coj.
In Omar Cherkaoui, editor, *GRES'99 3ème Colloque Francophone sur la Gestion de Réseaux et de Services*, pages 23-33, Montréal Canada, Juin 1999.

AG98

Sahin Albayrak and Francisco J. Garijo, editors.
Proceedings of the Second International Workshop on Intelligent Agents for Telecommunications Applications, IATA'98.
IOS Press, Berlin, June 1998.

Age98

Agent Oriented Software Pty. Ltd., Carlton, Victoria, Australia.
JACK Intelligent Agents User Guide, 1998.
http://www.agent-software.com.au.

AGLP99

P. Abel, P. Gros, D. Loisel, and J. P. Paris.
Virtual reality and network management.
In *7ème journées du GT Réalité Virtuelle*, June 1999.

Alb99

Sahin Albayrak, editor.
Intelligent Agents for Telecommunication Applications, number 1699 in Lecture Notes in Artificial Intelligence, Stockholm, Sweden, August 1999. Springer.

And99

Thomas Andersson.
An interaction and communication manager for Java RMI.
Master's thesis, ESSI, Université de Nice - Sophia Antipolis, February 1999.

Ane98

Nikos Anerousis.
An information model for generating computed views of management information.
In *Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, Newark, DE, October 1998.

Ark90

Ronald Craig Arkin.
Integrating behavioral, perceptual, and world knowledge in reactive navigation.
Journal of Robotics and Autonomous Systems, 6:105-122, 1990.

AS94

S. Appleby and S. Steward.
Software agents for control.
In P. Cochrane and PJT Mealthey, editors, *Modelling Future Telecommunication Systems*. Chapman & Hall, 1994.

AT94

M. Ahmed and K. Tesink.
Definitions of managed objects for ATM management version 8.0 using SMIv2.
RFC1695, August 1994.
ftp://ftp.nus.sg/pub/docs/rfc/rfc1695.txt.

BCS99

Paolo Bellavista, Antonio Corradi, and Cesare Stefanelli.
An open secure mobile agent framework for systems management.
Journal of Network and Systems Management, 7(3):323-339, September 1999.

BDF98

L. Berger, A.-M. Dery, and M. Fornarino.
Interactions between objects: An aspect of object-oriented languages.
In *ICSE'98 Workshop on Aspect-Oriented Programming*, Kyoto, Japan, April 1998.

Ber96

Guy Berthet.
Extension and Application of System-level Diagnosis Theory for Distributed Fault Management in Communication Networks.
PhD thesis, École Polytechnique Fédérale de Lausanne, Lausanne, CH, 1996.

BF96

Mihai Barbuceanu and Mark S. Fox.
The design of a coordination language for multi-agent systems.
In *Intelligent Agents III. Agent Theories, Architectures, and Languages*, pages 341-355. Springer, 1996.

BG93

M. Busuioc and D. Griffiths.
Co-operating intelligent agents for service management in communications networks.
In *Proceedings of CKBS-SIG*, Keele, UK, 1993.

BGP97

Mario Baldi, Silvano Gai, and Gian Piertro Picco.
Exploiting code mobility in decentralized and flexible network management.
In *Proceedings of the First International Workshop on Mobile Agents*, Berlin, Germany, April 1997.
Available at http://www.polito.it/~picco/papers/ma97.ps.gz.

BHG 98

Eric Bonabeau, Florian Henaux, Sylvain Guérin, Dominique Snyers, Psacale Kuntz, and Guy Theraulaz.
Routing in telecommunications networks with ant-like agents.
In Albayrak and Garijo [AG98], pages 60-71.

Bie97

Andrzej Bieszczad.
Application-oriented taxonomy of mobile code.
Technical Report SCE-97-13, Systems and Computer Engineering, Carleton University, June 1997.

Bla98

Uyless Black.

Signaling in broadband networks, volume ATM: Volume II.
Prentice-Hall, 1998.

BM99

C Bäumer and T. Magedanz.
Grasshopper - a mobile agent platform for active telecommunication networks.
In Albayrak [Alb99], pages 19-32.

BN84

Andrew D. Birrell and Bruce Jay Nelson.
Implementing remote procedure calls.
ACM Transactions on Computer Systems, 2:39-59, February 1984.

Boo94

Grady Booch.
Object Oriented Analysis and Design with Applications.
Addison-Wesley, 1994.

Bor99

Rafael Heitor Bordini.
Contributions to an Anthropological Approach to the Cultural Adaptation of Migrant Agents.
PhD thesis, University of London, 1999.

Bou92

Thierry Bouron.
Structures de communication et d'organisation pour la coopération dans un univers multi-agents.
PhD thesis, Université Paris VI, 1992.

BP97

Mario Baldi and Gian Pietro Picco.
Evaluating the tradeoffs of mobile code design paradigms in network management applications.
In R. Kemmerer and K. Futatsugi, editors, *20th International Conference on Software Engineering (ICSE'97)*, Kyoto (Japan), 1997.

BP98

Luis Bernado and Paulo Pinto.
Scalable service deployment on highly populated networks.
In Albayrak and Garijo [AG98], pages 35-44.

BPW

John Boyer, Bernard Pagurek, and Tony White.
Methodologies for PVC configuration in heterogeneous ATM environments using intelligent mobile agents.
Submitted to MATA'99.
ftp://ftp.sce.carleton.ca/pub/netmanage/jb-mata99.ps.gz.

Bra87

M. E. Bratman.
Intentions, Plans, and Practical Reason.
Harvard University Press, 1987.

BRHL99

Paolo Busetta, Ralph Rönnquist, Andrew Hodgson, and Andrew Lucas.
Jack intelligent agents - components for intelligent agents in Java.
AgentLink News Letter, January 1999.
White paper, http://www.agent-software.com.au.

BRJ99

Grady Booch, James Rumbaugh, and Ivar Jacobson.
The Unified Modeling Language User Guide.
Addison-Wesley, February 1999.

Bro91a

Rodney A. Brooks.
Intelligence without reason.
A.I. Memo 1293, Massachusetts Institute of Technology, April 1991.

Bro91b

Rodney A. Brooks.
Intelligence without representation.
Artificial Intelligence Journal, (47):139-159, 1991.

BTWW95

F.M. Brazier, J. Treur, N.J. Wijngaards, and M. Willems.
Formal specification of hierarchically (de)composed tasks.
In B.R. Gaines and M. Musen, editors, *9th Banff Knowledge Acquisition for Knowledge Based Systems Workshop*, Calgary, 1995.

Bus98

Marius Busuioc.
Intelligent Agents For Telecommunications Applications, chapter Distributed Intelligent Agents - A solution for the Management Complex Services, pages 105-116.
IOS Press, 1998.

BWP98

Andrzej Bieszczad, Tony White, and Bernard Pagurek.
Mobile agents for network management.
IEEE Communications Surveys, September 1998.

CBD99

Sang Choy, Markus Breugst, and Mohit Datta.
Management issues of a mobile agent-based service environment.
Journal of Network and Systems Management, 7(3), September 1999.

CC99

Pierre Conti and Morsy Cheikhrouhou.
Agents intelligents et gestion de réseaux ATM.
In Omar Cherkaoui, editor, *GRES'99 3ème Colloque Francophone sur la Gestion de Réseaux et de Services*, pages 133-149, Montréal Canada, Juin 1999.

CCL99

Morsy M. Cheikhrouhou, Pierre O. Conti, and Jacques Labetoulle.
Flexible software agents for the automatic provision of PVCs in ATM networks.
In *Second IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems*, Helsinki, Finland, 1999.
Extended version available at http://www.eurecom.fr/~cheikhro/docs/dais99-long.ps.gz.

CCLM99

Morsy Cheikhrouhou, Pierre Conti, Jacques Labetoulle, and Karina Marcus.
Intelligent agents for network management: Fault detection experiment.
In Sloman et al. [SML99], pages 595-610.

CCOL98

Morsy Cheikhrouhou, Pierre Conti, Raúl Texeira Oliveira, and Jacques Labetoulle.
Intelligent agents in network management, a state of the art.
Networking and Information Systems, 1(1):9-38, 1998.
http://www.eurecom.fr/~cheikhro/docs/StateOfTheArt.ps. gz.

CGH00

Stephen Corley, Francisco Garijo, and John Hickie.
Agent-based operational support systems.
In *6th International Conference on Intelligence in Networks(ICIN'2000)*,
Arcachon/Bordeaux, January 2000.

Cha97

Deepika Chauhan.

JAFMAS: A Java-based Agent Framework for Multiagent Systems Development and Implementation.
PhD thesis, ECECS Department, University of Cincinnati, 1997.
available from http://www.ececs.uc.edu/~abaker/JAFMAS/index.html.

Che97

Morsy Cheikhrouhou.
Agent's sensors and effectors, 1997.
(internal report).

Che99

Morsy M. Cheikhrouhou.
BDI-oriented agents for network management.
In *Proceedings of Globecom'99*, Rio Di Janeiro, December 1999. IFIP.

CJ96

David Cockburn and Nick R. Jennings.
Foundations of Distributed Artificial Intelligence, chapter ARCHON: A distributed artificial intelligence system for indiustrial applications, pages 319-344.
John Wiley & Sons, 1996.

CL00

Morsy Cheikhrouhou and Antonio Liotta.
Mobile versus static agents in network management: A case-study based comparison.
Technical report, Institut Eurecom, University of Surrey, 2000.

CLFZ95

L. Crutcher, A. Lazar, S. Feiner, and M. Zhou.
Managing networks through a virtual world.
IEEE Parallel and Distributed Technology, 3(2):4-13, 1995.

CLMC98

Pierre Conti, Jacques Labetoulle, Karina Marcus, and Morsy Cheikhrouhou.
Network management system with intelligent agents: A first step with SLD.
In *HPOVUA'98 Workshop*, Rennes, France, 1998.

CN99

Jaron Collis and Divine Ndumu.
The Role Modelling Guide.
BT labs, 1.01 edition, August 1999.
available at: http://www.labs.bt.com/projects/agents/zeus/docs.htm.

CNK96

Graham Chen, Michael Neville, and Qinzheng Kong.
Distributed network management using corba/tmn.
In *Proceedings of the Distributed Systems and Operations Management Workshop - DSOM'96*, L'Aquila, Italy, October 1996.

Com95

Douglas E. Comer.
Internetworking with TCP/IP.
Prentice Hall, Englewood Cliffs, N.J. 07632, 1995.

CST98

A. Corradi, C. Stefanelli, and F. Tarantino.
How to employ mobile agents in systems management.
In *Proceedings of the third International Conference on The Practical Applications of Intelligent Agents and Multi-Agent Technology*, pages 17-26, London, UK, March 1998.

CTC 98

Stephen Corley, Marius Tesselaar, James Cooley, Jens Meinköhn, Fabio Malabocchia, and Francisco Garijo.
The application of intelligent and mobile agents to network and service management.
In Albayrak and Garijo [AG98], pages 118-129.

Der97

Luca Deri.
A Component-based Architecture for Open, Independently Extensible Distributed Systems.
PhD thesis, University of Bern, June 1997.

Des98

CIM specification v2.0.
http://www.dmtf.org/cim/index.html, March 1998.

DFJN97

J. E. Doran, S. Franklin, N. R. Jennings, and T. J. Norman.
On cooperation in multi-agent systems.
The Knowledge Engineering Review, 12(3), 1997.
available at http://www.elec.qmw.ac.uk/dai/pubs/fomas.html.

Dil98

Bruno Dillenseger.
From interoperability to cooperation : Building intelligent agents on middleware.
In Albayrak and Garijo [AG98], page 221.

DLC89

E. H. Durfee, V. R. Lesser, and D. D. Corkill.
Cooperative distributed problem solving.
In A. Barr, P. R. Cohen, and E. A. Feigenbaum, editors, *The Handbook of Artificial Intelligence*, volume IV, pages 83-148. Addison-Wesley, 1989.

EDB99

M. El-Darieby and A. Bieszczad.
Intelligent mobile agents: Towards network fault management automation.
In Sloman et al. [SML99].

EDQD96

Babak Esfandiari, Gilles Deflandre, Joël Quinqueton, and Christophe Dony.
Agent-oriented techniques for network supervision.
Annals of Telecommunications, 51(9-10):521-529, 1996.

EMS97

Rolf Eberhart, Sandro Mazziotta, and Dominique Sidou.
Workshop on precise semantics for object-oriented modeling techniques.
In *Integrated Network Management V : integrated management in a virtual world*, San Diego, California, USA, May 1997. IFIP, Chapman & Hall.

Esk99

Philip Eskelin.
Component interaction patterns.
In *Proceedings of PLoP*, aug 1999.

Fal94

Boi Faltings.
Intelligence artificielle: au-delà des systèmes experts.
Flash Informatique SIC-EPFL, 1994.

Fer95

Jacques Ferber.
Les systèmes Multi-Agents: Vers une intelligence collective.
Jacques Ferber, Septembre 1995.
SBN 2-7296-0572-X.

FF98

Christian Frei and Boi Faltings.
A dynamic hierarchy of intelligent agents for network managemnt.
In Albayrak and Garijo [AG98], pages 1-16.

FFYA99

Olivier Festor, P. Festor, N. Ben Youssef, and Laurent Andrey.
Integration of wbem-based management agents in the osi framework.
In Morris Sloman, Subrata Mazumdar, and Emil Lupu, editors, *Sixth IFIP/IEEE International Symposium on Integrated Network Management IM'99*, volume 6, pages 49-64, Boston, MA, USA, May 1999.

FG96

Stan Franklin and Art Graesser.
Is it an agent, or just a program?: A taxonomy for autonomous agents.
In *Intelligent Agents III. Agent Theories, Architectures, and Languages*. Springer, 1996.

FG98

Jacques Ferber and Olivier Gutknecht.
A meta-model for the analysis and design of organisations in multi-agent systems.
In *Proceedings of ICMAS'98*, Paris, July 1998.

FIP97

Application design test: Network management and provisioning.
http://drogo.cselt.it/fipa/spec, June 1997.

FIP98

FIPA.
Fipa specifications, 1998.
http://www.fipa.org/spec/FIPA98.html.

fIPA99

Foundation for Intelligent Physical Agents.
Agent communication language.
http://www.fipa.org/spec/fipa9712.pdf, 1999.

FMNS97

Peter Fröhlich, Iara Móra, Wolfgang Nejdl, and Michael Schroeder.
Diagnostic agents for distributed systems.
In *Proceedings of ModelAge97*, Sienna, Italy, Junary 1997.
Available at http://www.kbs.uni-hannover.de/paper/96/ma1.ps.

FOR97

ForeRunner ATM switch configuration manual
.
http://www.fore.com/products/manuals.htm, March 1997.

FOR98

SPANS: Simple protocol for ATM network signaling.
http://www.mi.infn.it/INFN/atm/articles/spansfore.ps, 1998.

FPV98

A. Fuggetta, G. P. Picco, and G. Vigna.
Understanding code mobility.
IEEE Transactions on Software Engineering, 24(5):342-361, 1998.

Fro94

Svend Frolund.
Constraint-Based Synchronization of Distributed Activities.
PhD thesis, University of Illinois, 1994.

FTP97

FTP software agent technology.
Technical report, FTP Software, http://www.ftp.com/product/whitepapers/4agent.htm, 1997.

FW91

T. Finin and G. Wiederhold.
An overview of KQML: A knowledge query and manipulation language, 1991.
Available through the Standford University Computer Science Dept.

FYF99

Patricia G. S. Flofissi, Yechiam Yemini, and Danilo Florissi.
QoSockets: a new extension to the sockets API for end-to-end application QoS management.
In Sloman et al. [SML99].

GA97

Garry Grimes and Brian P. Alley.
Intelligent agents for network fault diagnosis and testing.
In *Integrated Network Management V : Integrated Management in a virtual World*, pages 232-244, San Diego, California, USA, May 1997. IFIP, Chapman & Hall.

Gar96

M. Garrett.
A service architecture for ATM: From applications to scheduling.
IEEE Network, 3, 1996.

Gen96

Guy Genilloud.
Towards a Distributed Architecture for Systems Management.
PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 1996.

GF99

Olivier Gutknecht and Jacques Ferber.
Vers une méthodologie organisationnelle de conception de systèmes multi-agents.
Technical report, Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier, Juin 1999.

GGGO99

Damianos Gavals, Dominic Greenwood, mohammed Ghanbari, and Mike O'Mahony.
Using mobile agents for distributed network performance management.
In Albayrak [Alb99].

Gha94

Malik Ghallab.
Past and future chronicles for supervision and planning.
In Jean Paul Haton, editor, *Proceedings of the 14th Int. Avignon Conference*, pages 23-34, Paris, June 1994. EC2 and AFIA.

Ghe97

I. G. Ghetie.
Networks and Systems Management. Platforms Analysis and Evaluation.
Kluwer Academic Publishers, 1997.

GHN 97

Shaw Green, Leon Hurst, Brenda Nangle, Pádraig Cunningham, Fergal Somers, and Richard Evans.
Software agents: A review.
Technical report, Trinity College Dublin and Broadcom Éireann Research Ltd, May 1997.
Available at http://www.cs.tcd.ie/research_groups/aig/iag/pubreview.p s.gz.

GI90

M. P. Georgeff and F. F. Ingrand.
Real-time reasoning: The monitoring and control of spacecraft systems.
In *Sixth IEEE conference on Artificial Intelligence Applications*, Santa Brabra, CA, USA, March 1990.

GJ97

M. A. Gibney and N. R. Jennings.
Market based multi-agent systems for ATM network management.
In *Proc. 4th Communications Networks Symposium*, Manchester, UK, 1997.

Available from http://www.elec.qmw.ac.uk/dai/projects/agentCAC/.

Gla96

Norbert Glaser.
Contribution to Knowledge Modelling in a Multi-Agent Framework.
PhD thesis, Université Poincaré, Nancy, France, Novembre 1996.

GME 95

German Goldszmidt, Kraig Meyer, Mike Erlinger, Joe Betser, and Yechiam Yemini.
Decentralizing control and intelligence in network management.
In Morris Sloman, Subrata Mazumdar, and Emil Lupu, editors, *Integrated Network Management*, IFIP/IEEE International Symposium on Integrated Network Management, pages x-y. IEEE Publishing, May 1995.

Gol93

Germán Goldszmidt.
Distributed system management via elastic servers.
In *IEEE First International Workshop on Systems Management*, pages 31-35, Los Angeles, California, April 1993.

Gre96

Christine Gressley.
Network management resources - reviews of network management systems, July 1996.

Available at http://tampico.cso.uiuc.edu/~gressley/netmgmt/reviews/ .

Gro97

The Intelligent Agents Group.
A survey on intelligent agents in telecommunications.
http://www.cs.tcd.ie/research_groups/aig/iag/survey.html , June 1997.

GT99

Luciano Paschoal Gsapary and Liane Rockenbach Tarouco.
Characterization and measurments of entreprise network traffic with rmon2.
In Stadler and Stiller [SS99], pages 229-242.

GY95

Germán Goldszmidt and Yechiam Yemini.
Distributed Management by Delegation.
In *The 15th International Conference on Distributed Computing Systems*. IEEE Computer Society, June 1995.

GY98

Germán Goldszmidt and Yechiam Yemini.
Delegated agents for network management.
IEEE Communications Magazine, 36(3), March 1998.

HAN99

Heinz-Gerd Hegering, Sebastien Abeck, and Bernhard Neumair.
Integrated Management of Networked Systems.
Morgan Kaufmann Publishers, San Francisco, California, 1999.

HB91

Jean-Paul Haton and Nadjet Bouzid.
Le raisonnement en intelligence artificielle, modèles, techiques et architectures pour les systèmes à bases de connaissances.
InterEditions, 1991.

HB98

Alex L. G. Hayzelden and J. Bigham.
Heterogeneous multi-agent architecture for ATM virtual path network resource configuration.
In Sahin Albayrak and Francisco J. Garijo, editors, *Intelligent Agents for Telecommunication Applications*, pages 45-59, Cité La Villette, Paris, France, July 1998. Springer.

HB99

Alex L.G. Hayzelden and John Bigham.
Agent technology in communications systems: An overview.
Knowledge Engineering Review, 1999.
To appear.

HBCa91

Jean-Paul Haton, Nadjet Bouzid, Francois Charpillet, and al.
Le raisonnement en intelligence artificielle.
Haton et al, Paris, intereditions edition, 1991.

HCCB94

David Hutchison, Geoff Coulson, Andrew Campbell, and Gordon S. Blair.
Quality of Service Management in Distributed Systems, chapter 11.
Addison-Wesley, 1994.

HD98

Jim Hardwicke & Rob Davidson.
Software agents for atm performance management.
In *Networks Operation and Maintenance Symposium (NOMS98)*, page 313.

Hic98

Richard C. Hicks.
The trimm methodology for maintainable rule-based systems.
Technical report, University of Nevada, Las Vegas, 1998.

HJ97

Gisli Hjálmtýsson and A. Jain.
An agent-based approach to service management - towards service independent network architecture.
In *Integrated Network Management V : integrated management in a virtual world*, pages 715-729, San Diego, California, USA, May 1997. IFIP, Chapman & Hall.

HKK 99

John Hickie, James Kennedy, Georgios Koudouridis, Vaggelis Ouzounis, and Matthew Studley.
A scaleable heterogeneous architecture for agent oriented workflow management.
In *Workshop on Intelligent Information Integration (IJCAI-99)*, Stockholm, July 1999.

HS99

Michael N. Huns and Larry M. Stephens.
Multiagents Systems, chapter Multiagent Systems and Societies of Agents, pages 79-120.
The MIT Press, Cambridge, MA, 1999.

Hub97

Oliver J. Huber.
Multimedia services based on agents.
In *IBC Intelligent Agents Conference*, Café Royal, Londres, March 1997.
Available from http://www.rennes.enst-bretagne.fr/~aber/oliver/index- fr.html.

Hui96

Christian Huitema.
Et Dieu créa l'Internet.
Eyrolles, 1996.

IFI97

IFIP.
Integrated Network Management V: integrated management in a virtual world, San Diego, California, USA, May 1997. Chapman & Hall.

IGG98

Carlos A. Iglesias, Mercedes Garijo, and José C. Gonzáles.

A survey of agent-oriented methodologies.
In *Intelligent Agents V. Agent Theories, Architectures, and Languages*. Springer, 1998.

IHM99

Leila Ismail, Daniel Hagimont, and Jacques Mossière.
Evaluation of the mobile agent technology: Comparison with the client/server paradigm.
In *Proceedings of OOPSLA'99, the Int. Conf. on Object-Oriented Programming, Systems and Applications*, 1999.

IIM93

Iso/ccitt and internet management coexistence (iimc): Translation of internet mibs to iso/ccitt gdmo mibs, 1993.

ISO98

Command sequencer.
International Standard ISO 10164-21, 1998.

JC97

Herbert Jaeger and Thomas Christaller.
Dual dynamics: Designing behavior systems for autonomous robots.
In S. Fujimura and M. Sugisaka, editors, *Proceedings of the International Symposium on Artificial Life and Robotics*, pages 76-79. Beppu, Japan, Februar 1997.

JCL 96

N.R. Jennings, J. Corea, I. Laresgoiti, E.H. Mamdani, P. Skarek, and L.Z. Varga.
Using archon to develop real-world dai applications for electricity transportation management and particle accelerator control.
IEEE Expert: Special Issue on Real World Applications of DAI systems, December 1996.

Jen00

Nicholas R. Jennings.
On agent-based software engineering.
Artificial Intelligence, 117:277-296, 2000.

JNL95

Timothy J., Norman, and Derek Long.
Goal creation in motivated agents.
In Michael Wooldridge and N. R. Jennings, editors, *Proceedings of the 1994 Workshop on Agent Theories, Architectures, and Languages*. Springer, 1995.

JNRS99

Jia Jiao, Shamin Naqui, Danny Raz, and Binay Sugla.
Minimizing the monitoring cost in network management.
In Sloman et al. [SML99].

JPC00

Heecheol Jeon, Charles Petrie, and Mark R. Cutkosky.
Jatlite: A java agent infrastructure with message routing.
IEEE Internet Computing, 2000.

JVS99

Ajita John, Keith Vanderveen, and Binay Sugla.
an XML-based framework for dynamic SNMP MIB extension.
In Stadler and Stiller [SS99], pages 107-120.

JW98a

Nicholas R. Jennings and Michael J. Wooldridge.
Agent Technology: Foundations, Applications and Markets, chapter Applications of Intelligent Agents.
Springer-Verlag, February 1998.

JW98b

Nicholas R. Jennings and Michael J. Wooldrige, editors.
Agent Technology: Foundations, Applications, and Markets.
Springer-Verlag, 1998.

KH99

Graham Knight and Reza Hazemi.
Mobile agent-based management in the INSERT project.
Journal of Network and Systems Management, 7(3):271-293, September 1999.

KMM99

Kwindla Hultman Kramer, Nelson Minar, and Pattie Maes.
Tutorial: Mobile software agents for dynamic routing.
Mobile Computing and Communications Review, 3(2):12-16, 1999.
http://nelson.www.media.mit.edu/people/nelson/research/routes-sigmobile/.

KOE 98

David Kerr, Donie O'Sullivan, Richard Evans, Ray Richardson, and Fergal Somers.
Experiences using intelligent agent technologies as a unifying approach to network and service management.
In *Fifth International Conference on Intelligence in Services and Networks Technology for Ubiquitous Telecom Services*, Antwerp, Belgium, May 1998.

Koo95

Richard Kooijman.
Divide and conquer in network management using event-driven network area agents.
In *ASCI Conference*, Heijderbos, Nederland, May 1995.
available at http://netman.cit.buffalo.edu/Doc/Papers/koo9505.ps.

KSSZ97

P. Kalyanasundaram, A.S. Sethi, C. Sherwin, and D. Zhu.
A spreadsheet-based scripting environment for snmp.
In *Fifth IFIP/IEEE International Symposium on Integrated Network Management IM'97*, volume 5, pages 752-765, San-Diego, California, USA, May 1997. Chapman & Hall.

Lau98

Markku Laukkanen.
Snmp based network management using corba.
In *Networks Operation and Maintenance Symposium (NOMS98)*, page 276, New Orleans, USA, February 1998.

LC97

D. B. Lange and D. T. Chang.
Ibm aglets workbench - programming mobile agents in java.
White Paper, 1997.
IBM Corp.

Lew99

Lundy Lewis.
Service Level Management for Entreprise Networks.
Artch House, 1999.

LF93

Allan Leinwand and Karen Fang.
Nerwork Management - A Practical Perspective.
Addison Wesley UNIX and Open Systems Series. Addison Wesley, United States of America, 1993.

LF97

Yannis Labrou and Tim Finin.
A proposal for a new KQML specification.
Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, Baltimore, Maryland, USA, February 1997.
http://www.cs.umbc.edu/kqml/papers/.

LL93a

S. Labidi and W. Lejouad.

De l'intelligence artificielle distribuée aux systemes multi-agents.
Technical report, INRIA Sophia Antipolis, INRIA Sophia Antipolis, 2004 route des Lucioles, BP 93, 06902 Sophia-Antipolis Cedex,France, 1993.

LL93b

Sofiane Labidi and Wided Lejouad.
De l'intelligence artificielle distribuée aux systèmes multi-agents.
Technical report, INRIA, Sophia-Antipolis, France, Aout 1993.

LNNW98

L. C. Lee, H. S. Nwana, D. T. Ndumu, and P. De Wilde.
The stability, scalability and performance of multi-agent systems.
BT Technology Journal, 16(3):94, July 1998.

LS97

E.C. Lupu and M. Sloman.
Towards a role-based framework for distributed systems management.
Journal of Network and Systems Management, 5(1), 1997.

LS99

D. B. Levi and J. Schönwälder.
Definitions of managed objects for the delegation of management scripts.
RFC 2592, 1999.
http://www.ietf.org/rfc/rfc2592.txt.

Lyn96

Nancy A. Lynch.
Distributed Algorithms.
Morgan Kaufmann, 1996.

M98a

Jörg P. Müller.
Architecture and applications of intelligent agents: A survey.
The knowledge Engineering Review, 13(4):353-380, 1998.

M98b

Jörg P. Müller.
The right agent (architecture) to do the right thing.
In *Intelligent Agents V. Agent Theories, Architectures, and Languages*. Springer, 1998.

MA98

Jens Meinkohn and Sahin Albayrak.

Future IN-platforms with agent technology.
In Albayrak and Garijo [AG98], page 80.

Mae89

Pattie Maes.
How to do the right thing.
Connection Science Journal, 1(3):291-323, 1989.

Mae94

Pettie Maes.
Agents that reduce work and information overload.
Communications of the ACM, 37(7):31-40, 1994.

Mag95

Thomas Magedanz.
On the impacts of intelligent agents concepts on future telecommunication environments.
In *Third International Conference on Intelligence in Broadband Services and Networks*, Crete, Greece, October 1995.

Mag99

Thomas Magedanz.
Agent concepts, standards, technologies and applications - a brief introduction.
URL:http://www.fokus.gmd.de/research/cc/ecco/climate/entry.html, April 1999.
Barcelona, Spain.

Mas97

M. C. Maston.
Using the world wide web and java for network service management.
In *Integrated Network Management V : integrated management in a virtual world*, volume 5, pages 71-84, San Diego, California, USA, May 1997. IFIP, Chapman & Hall.

Maz97

Sandro Mazziotta.
Spécification et Génération de Tests du Comportement Dynamique des Systèmes à Objets Répartis.
PhD thesis, Universite de Nice-Sophia Antipolis, 1997.

Maz99

Sandro Mazziotta.
Mach: Market mechanisms for selling on-demand ip bandwith.
In Omar Cherkaoui, editor, *GRES'99 3ème Colloque Francophone sur la Gestion de Réseaux et de Services*, pages 1-9, Montréal Canada, Juin 1999.

MC94

F. G. McCabe and K. L. Clark.
April: Agent process interaction language.
Technical report, Dept. of Computing, London, UK, November 1994.
available from http://www-lp.doc.ic.ac.uk/~klc/.

MF99

Jean-Philipe Martin-Flatin.
Push vs. pull in web-based network management.
In Sloman et al. [SML99].

MFBH99

Jean-Philippe Martin-Flatin, L. Bovet, and Jean-Pierre Hubaux.
JAMAP: a web-based management platform for IP networks.
In Stadler and Stiller [SS99].
Available at http://ica2www.epfl.ch/~jpmf/papers/dsom99.pdf.

MFS99

Jean-Philippe Martin-Flatin and Ron Sprenkels.
Bulk transfers of MIB data.
The Simple Times, 7(1), March 1999.

MFZ97a

Jean-Philippe Martin-Flatin and Simon Znaty.
Annotated typology of distributed network management paradigms.
Technical Report SSC/1997/008, EPFL, Lausanne, Switzerland, March 1997.

MFZ97b

Jean-Philippe Martin-Flatin and Simon Znaty.
A simple typology of distributed network management paradigms.
In *Proceedings of the Distributed Systems and Operations Management Workshop - DSOM'97*, Sidney, Australia, October 1997.

MFZH98

Jean-Philippe Martin-Flatin, Simon Znaty, and Jean-Pierre Hubaux.
A survey of distributed network and systems management paradigms.
Technical Report SSC/1998/024, EPFL, Lausanne, Switzerland, March 1998.

MFZH99

Jean-Philippe Martin-Flatin, Simon Znaty, and Jean-Pierre Hubaux.
A survey of distributed entreprise network and systems management paradigms.
Journal of Network and Systems Management, 7(1), March 1999.

MH99

Sheng Ma and Joseph L. Hellerstein.
Event browser: A flexible tool for scalable analysis of event data.
In Stadler and Stiller [SS99], pages 285-296.

Mir92

Equipe Miriad.
Approcher la notion de collectif.
In *Actes des Journées "Systèmes Systèmes Multi-Agents"*. P.R.C.- G.D.R, Nancy, Decembre 1992.

MKM99

Nelson Minar, Kwindla Hultman Kramer, and Pattie Maes.
Software Agents for Future Communications Systems, chapter Cooperating Mobile Agents for Dynamic Network Routing.
Springer-Verlag, 1999.
ISBN: 3-540-65578-6.

Mou96

Maria-Athina Mountzia.
Intelligent agents in integrated network and systems management.
In *Proceedings of the EUNICE'96 Summer School*, Lausanne, Switzerland, September 1996.

Mou98

Maria-Athina Mountzia.
A distributed management approach based on flexible agents.
Journal of Interoperable Communication Networks, 1998.
Special Issue on Telecommunications Service Engineering, Baltzer Science Publishers.

MR99

Maria-Athina Mountizia and Gabi Dreo Rodosek.
Using the concepts of intelligent agents in fault management of distributed services.
Journal of Network and Systems Management, 7(4):425-446, December 1999.

MRK96

T. Magedanz, K. Rothermel, and S. Krause.
Intelligent agents: An emerging technology for next generation telecommunications?
In *INFOCOM'96*, pages 464-472, USA, March 24-28 1996. IEEE.

MS96

Damian Marriot and Morris Sloman.

Implementation of a management agent for interpreting obligation policies.
In *IEEE/IFIP 7th International Workshop on Distributed Systems and Operations Management - DSOM'96*, L'Aquila, Italy, October 1996.

Mul96

Jörg P. Muller.
The Design of Intelligent Agents - A Layered Approach.
LNAI State-of-the-Art Survey. Springer, Berlin, Germany, 1996.

(NM99

TeleManagement Forum (NMF).
Network management detailed operations map.
Technical report, TM Forum, March 1999.

Nwa96

Hyacinth S. Nwana.
Software agents: An overview.
Knowledge Engineering Review, 11(3):205-244, October/November 1996.
Available at http://www.cs.umbc.edu/agents/introduction/ao/.

Obj97

Mobile agent facility specification, June 1997.
OMG TC Document cf/xx-x-xx.

OL95

Raúl Oliveira and Jacques Labetoulle.
Intelligent agents : a way to reduce the gap between applications and networks.
In J. D. Decotignie, editor, *Proceedings of the First IEEE International Workshop on Factory Communications Systems - WFCS'95*, pages 81-90, Leysin, Switzerland, October 4-6 1995.
Available at http://www.eurecom.fr/~oliveira/wfcs/wfcs.ps.gz.

OL96a

Raúl Oliveira and Jacques Labetoulle.
From intelligent agents towards management by request.
In *Proceedings of the International Conference in Intelligence on Networks - ICIN'96*, Bordeaux, France, November 1996.

OL96b

Raúl Oliveira and Jacques Labetoulle.
Intelligent agents a new management style.
In *Proceedings of the Distributed Systems and Operations Management Workshop - DSOM'96*, L'Aquila, Italy, October 1996.

OL98

Raúl Oliveira and Jacques Labetoulle.
Mania: Managing awareness in networks through intelligent agents.
In *Networks Operation and Maintenance Symposium (NOMS98)*, pages 431-440, New Orleans, USA, February 1998.

Oli98

Raúl Oliveira.
Gestion des Réseaux avec Connaissance des Besoins : Utilisation des Agents Logiciels.
PhD thesis, École National Supérieure des Télécommunications, 1998.

ONI 97

Akihiko Ohsuga, Yasuo Nagai, Yutaka Irie, Masanori Hattori, and Shinichi Honiden.
Plangent: An approach to making mobile agents intelligent.
IEEE Internet Computing, 1(4), July/August 1997.

OSL96

Raúl Oliveira, Dominique Sidou, and Jacques Labetoulle.
Customized network management based on applications requirements.
In *Proceedings of the First IEEE International Workshop on Enterprise Networking - ENW '96*, Dallas, Texas, USA, June 27 1996.

PAA99

PAAM´99. The Pratical Application of Intelligent Agents and Multi-Agent Technology, London, April 1999. The Practical Application Company Ltd.

Par95

Nick Parkyn.
Architecture for intelligent (smart) agents, June 1995.
available at http://www.citr.uq.oz.au/.

Pav99

George Pavlou.
A novel approach for mapping the osi-sm/tmn model to odp/omg corba.
In Stadler and Stiller [SS99], pages 67-82.

PGR99

Charles Petrie, Sigrid Goldman, and Andréas Raquet.
Agent-based process management.
In *IJCAI-99 Workshop on Workflow and Process Management*, Stockholm, Sweden, August 1999.

Pic98

Gian Pietro Picco.
Understanding, Evaluating, Formalizing, and Exploiting Code Mobility.
PhD thesis, Politecnico di Torino, 1998.

PLBS98

Bernard Pagurek, Yanrong Li, Andrzej Bieszczad, and Gatot Susilo.
Network configuration management in heterogeneous ATM environments.
In Sahin Albayrak and Francisco J. Garijo, editors, *Intelligent Agents for Telecommunication Applications - IATA'98*, number 1437 in Lecture Notes in Artificial Intelligence, Paris, France, July 1998.

Plu98

Michel Plu.
Agent Technology: Foundations, Applications and Markets, chapter Software Technologies for Building Agent Based Systems in Telecommunications Networks.
Springer-Verlag, 1998.

PM97

Anthony Phlipponneau and Jean-François Milhomme.
ATM to VPN performance management.
Second term project, Institut Eurécom, November 1997.
ACTS Project AC052: PROSPECT.

PM99

Jorge Luis Tellez Portas and Tayeb Ben Meriem.
Spécification et implémentation d'un modèle de gestion.
In Omar Cherkaoui, editor, *GRES'99 3ème Colloque Francophone sur la Gestion de Réseaux et de Services*, pages 290-302, Montréal Canada, Juin 1999.

PMC67

F.P. Preparata, G. Metze, and R.T. Chien.
On the connection assignement problem of diagnosable systems.
IEEE Transactions on Electronic Computers, EC-16(6):848-853, 1967.

PMP

Perpetuum mobile procura project.

http://www.sce.carleton.ca/netmanage/perpetuum.shtml.
Department of Systems and Computer Engineering, Carleton University.

PNM99

G. Penido, J.M. Nogueira, and C. Machado.

An automatic fault diagnosis and correction system for telecommunications management.
In Sloman et al. [SML99].

PP98

P712 and P815.
Agent based computing: A booklet for executives.
EURESCOM, September 1998.

Pra95

Aiko Pras.
Network Management Architecture.
PhD thesis, Universiteit Twente, Department of Computer Science, February 1995.
available from http://wwwsnmp.cs.utwente.nl/~pras/thesis.html.

PSS99

Peter Parnes, Kare Synnes, and Dick Schefström.
Real-time control and management of distributed applications using IP-multicast.
In Sloman et al. [SML99].

PT99

Antonio Puliafito and Orazio Tomarchio.
Advanced network management functionalities through the use of mobile software agents.
In Albayrak [Alb99].

Puj97

Guy Pujolle.
Les Réseaux.
Eyrolles, 1997.

PWW00

B. Pagurek, Y. Wang, and T. White.
Integration of mobile agents with SNMP: Why and how.
Submitted to NOMS'2000, 2000.

RBE 95

James Rumbaugh, Michael Blaha, Frederick Eddy, William Premelani, and William Lorensen.
OMT Modélisation et conception orientées objet.
Masson, 1995.

RC97

Anne-Isabelle Rivière and Stéphane Combes.
De nouvelles approches pour l'intégration de l'information de gestion.
In *GRES'97. 2ème Colloque Francophone Gestion de Réseaux et de Services*, pages 247-258, Rennes, France, Septembre 1997.

Ret98

Agent builder: An integrated toolkit for constructing intelligent software agents.
http://www.agentbuilder.com, September 1998.

RG91

Anand S. Rao and Michael P. Georgeff.
Modelling agents within a BDI architecture.
In R. Fikes and E. Sandewall, editors, *Second International Conference on Principles of Knowledge Representation and Reasoning*, pages 473-484, April, 1991.

RG92

Anand S. Rao and Michael P. Georgeff.
An abstract architecture for rational agents.
In Morgan Kaufman, editor, *Third International Conference on Principles of Knowledge Representation and Reasoning*, pages 439-449, 1992.

RG95

Anand S. Rao and Michael P. Georgeff.
BDI agents: from theory to practice.
In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 321-319, S. Francisco, CA, June 1995.

Roz99

Moshe Rozenblit.
Security for Telecommunications Network Management.
IEEE Press, 1999.

RP94

J. Reynolds and J. Postel.
Assigned Numbers.
http://ds.internic.net/, RFC 1700, 1994.

RPS96

Anne-Isabelle Rivière, Adrian Pell, and Michelle Sibila.
Network management information: From protocols to information integration.
In *Proceedings of the Distributed Systems and Operations Management Workshop - DSOM'96*, L'Aquila, Italy, October 1996.

RU95

Mike Rizzo and Ian A. Utting.
An agent-based model for the provision of advanced telecommunications services.
In *Proceedings of TINA '95*, Melbourne, Australia, 1995.

Rum92

J. Rumbaugh.
Horsing around with associations.
Journal of Object Oriented Programming, 4:20-29, February 1992.

Sah99

Akhil Sahai.
Conception et Réalisation d'un Gestionnaire Mobile de Réseaux Fondé sur la Technologie d'Agent Mobile.
PhD thesis, Université de Rennes 1, January 1999.

SBM97

Akhil Sahai, Stéphane Billiart, and Christine Morin.
Astrolog: A distributed and dynamic environment for network and system management.
In *Proceedings of the 1st European Information Infrastructure User Conference*, Germany, February 1997.
Available at http://www.irisa.fr/solidor/doc/pub97.html.

SBP98

Cheryl Schramm, Andrzej Bieszczad, and Bernard Pagurek.
Application-oriented network modeling with mobile agents.
In *Networks Operation and Maintenance Symposium (NOMS98)*, New Orleans, USA, February 1998.
Poster.

SC96

Nikolaos Skarmaeas and Keith L. Clark.
Process oriented programming for agent-based network management.
In *ECAI96 Workshop on Intelligent Agents for Telecommunication Applications (IATA96)*, Budapest, Hungary, August 12 - 16 1996.

Sch97

Jürgen Schönwälder.
Network management by delegation - from research prototypes towards standards.
Computer Networks and ISDN Systems, 29(15):1843-1852, 1997.

SD99

M. W. A. Steen and J. Derrick.
Formalising ODP Enterprise Policies.

In *3rd International Enterprise Distributed Object Computing Conference (EDOC '99)*, University of Mannheim, Germany, September 1999. IEEE Publishing.

SDPW97

K. Sycara, K. Decker, A.S. Pannu, and M. Williamson.
Designing behaviors for information agents.
In *First International Conference on Autonomous Agents*, February 1997.

Sea70

John R. Searle.
Speech Acts: An Essay in the Philosophy of Language.
Cambridge University Press, 1970.

SH99

Ruud Schoonderwoerd and Owen Holland.
Software Agents for Future Communications Systems, chapter Minimal Agents for Communications Networks Routing: The Social Insect Paradigm.
Springer-Verlag, 1999.
ISBN: 3-540-65578-6.

SHB97

Ruud Schoonderwoerd, Owen Holland, and Janet Bruten.
Ant-like agents for load balancing in telecommunications networks.
In *Proceedings of the First International Conference on Autonomous Agents*. ACM Press, 1997.
http://www-uk.hpl.hp.com/people/ruud/abcAA97.ps.

SHBR96

R. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz.
Ant-based load balancing in telecommunications networks.
Journal of Adaptive Behavior, 5(2):169-207, 1996.
http://www-incl.hpl.hp.com/people/ruud/thesisRuud.ps.Z.

Sho93

Yoav Shoham.
Agent-oriented programming.
Artificial Intelligence, pages 51-92, 1993.

Sid97

Dominique Sidou.
Precise semantics for a behavior model in the context of object based distributed systems.
Technical Report TUM-I9725, technical university of Munich, Haim, 1997.
Workshop on precise semantics for object-oriented modeling techniques.

Sid99

David J. Sidor.
An update on itu-t tmn standards.
Journal of Network and Systems Management, 7(3):357-362, September 1999.

Sin94

Munindar P. Singh.
Multiagent systems: A theoritcal framework for intentions know-how, and communications.
In J.G. Carbonell and J. Siekmann, editors, *Lecture Notes in Artificial Intelligence*. Springer, 1994.

Sin98

Munindar P. Singh.
Agent communication languages: Rethinking the principles.
Computer, 31(12):40-47, December 1998.

SKN97

Motohiro Suzuki, Yoshiaki Kiriha, and Shoichiro Nakai.
Delegation agents: Design and implementation.
In *Integrated Network Management V : integrated management in a virtual world*, volume 5, pages 742-751, San Diego, California, USA, May 1997. IFIP, Chapman & Hall.

Slo94

Morris Sloman.
Network and Distributed Systems Management.
Addison-Wesley, 1994.

SM87

Inc Sun Microsystems.
External Data Representation Standard.
http://ds.internic.net/, RFC 1014, 1987.

SM88

Inc Sun Microsystems.
Remote Procedure Call.
http://ds.internic.net/, RFC 1050, 1988.

SM89

Inc Sun Microsystems.
Network File System.
http://ds.internic.net/, RFC 1094, 1989.

SM98a

Akhil Sahai and Christine Morin.
Enabling a mobile network manager through mobile agents.
In *Proceedings of Mobile Agents'98,* Lecture Notes on Computer Science, Stuttgart, Germany, September 1998.

SM98b

Fotis Stamatelopoulos and Basil Maglaris.
A caching model for efficient distributed network and systems management.
In *3rd International Symposium on Computers and Communications - ISCC'98*. IEEE, July 1998.
http://www.netmode.ece.ntua.gr/papers/iscc98.ps.

SML99

Morris Sloman, Subrata Mazumdar, and Emil Lupu, editors.
Integrated Network Management VI: Distributed Management for the Networked Millennium, IFIP/IEEE International Symposium on Integrated Network Management, Boston, MA, USA, May 1999. IEEE Publishing.

Som96

Fergal Somers.
Hybrid: Unifying centralised and distributed management for large high-speed networks.
In *Networks Operation and Maintenance Symposium (NOMS96)*, Kyoto, 1996.

Som98

Fergal Somers.
Hybrid: Intelligent agents for distributed atm network management.
In Albayrak and Garijo [AG98], pages 55-66.

SQ99

J. Schonwalder and J. Quittek.
Secure management by delegation within the internet management framework.
In Sloman et al. [SML99].

SR96

P. Steenekamp and J. Roos.
Implementation of distributed systems management policies: A framework for the application of intelligent agent technology.
In *2nd International Workshop on Systems Management*, Toronto, Ontario, Canada, June 1996. IEEE.

SRG99

Munindar P. Singh, Anand S. Rao, and Michael P. Georgeff.
Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, chapter Formal Methods in DAI: Logic-Based Representation and Reasoning, pages 331-376.
The MIT Press, 1999.

SRI

Artificial Intelligence Center, SRI International, Menlo Park, CA, USA.
Procedural Reasoning System User's Guide.

SRM95

F. Stamatelopoulos, N. Roussopoulos, and B. Maglaris.
Using a DBMS for hierarchical network management.
Engineer Conference, NETWORLD+INTEROP'95, March 1995.
http://www.netmode.ece.ntua.gr/papers/interop_e16.eps.

SS99

Rolf Stadler and Burkhard Stiller, editors.
Active Technologies for Network and Service Management, number 1700 in Lecture Notes in Computer Science. Springer, October 1999.

Sta96

William Stallings.
SNMP, SNMPv2 and RMON, Practical Network Management.
Addison-Wesley, USA, 1996.

Ste90

Richard W. Stevens.
UNIX network programming.
Prentice-Hall, 1990.

Ste95

Douglas W. Stevenson.
Network management: What it is, what it isn't., 1995.
avaialable at http://netman.cit.buffalo.edu/Doc/DStevenson.

Sun98

Java management programmer's guide.
http://java.sun.com/products/JavaManagement/, February 1998.

SV98

Peter Stone and Manuela Veloso.
Task decomposition and dynamic role assignment for real-time strategic teamwork.

In *Intelligent Agents V. Agent Theories, Architectures, and Languages*. Springer, 1998.

TB94

K. Tesink and T. Brunner.
(Re)Configuration of ATM virtual connections with SNMP.
The Simple Times, 3(2), August 1994.

Ter92

Kornel Terplan.
Communication Networks Management.
Prentice Hall, 1992.

Ter98

Kornel Terplan.
Telecom Operations Management Solutions with NetExpert.
CRC Press, 1998.

TK97

Markus Trommer and Robert Konopka.
Distributed network management with dynamic rule-based managing agents.
In *Integrated Network Management V : integrated management in a virtual world*, pages 730-741, San Diego, California, USA, May 1997. IFIP/IEEE, Chapman & Hall.

Tok96

Mario Tokoro.
An agent is an individual that has consciousness.
In *Intelligent Agents III. Agent Theories, Architectures, and Languages*, pages 45-46, Budapest, Hungary, August 12-13 1996. Springer.

TSS 97

David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minde.
A survey of active network research.
IEEE Communications Magazine, 35(1):80-86, January 1997.

URL

The DIANA project homepage.
http://www.eurecom.fr/~diana.

Uto98

Utopia user manual version 4.1.
http://wwwsnmp.cs.utwente.nl/

nm/research/projects/utopia/release4.1/manual4.1.html, 1998.

Vig98

Giovanni Vigna, editor.
Mobile Agents and Security.
Number 1419 in Lecture Notes in Computer Science. Springer Verlag, August 1998.
ISBN: 3540647929.

WA98

Dirk Wieczorek and Sahin Albayrak.
Open scalable agent architecture for telecommunication applications.
In Albayrak and Garijo [AG98], pages 233-249.

Wag96

Gerd Wagner.
Vivid agents - how they deliberate, how they react, how they are verified.
In W. Van de Velde and J.W. Perram, editors, *Agents Breaking Away, Proc. of MAAMAW'96*, 1996.
Available from http://www.informatik.uni-leipzig.de/~gwagner/.

Wag00

Gerd Wagner.
Agent-oriented analysis and design of organizational information systems.
In *IV IEEE International Baltic Workshop on Databases and Information Systems*, Vilnius, Lithuania, May 2000.

WBP98

Tony White, Andrzej Bieszczad, and Bernard Pagurek.
Distributed fault location in networks using mobile agents.
In Albayrak and Garijo [AG98], pages 130-141.

Wei99

Gerhard Weiss, editor.
Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence.
The MIT Press, Cambridge, MA, 1999.

Wel93

M. P. Wellman.
A market oriented programming environment and its application to distributed multicommodity flow problems.
Journal of Artificial Intelligence Research, 1:1-23, 1993.

WJ95

Michael Wooldridge and Nicholas R. Jennings.
Intelligent Agents: Theory and Practice.
Knowledge Engineering Review, 10(2):115-152, 1995.

WJ98

Michael Wooldrige and Nicholas R. Jennings.
Pitfalls of agent-oriented development.
In Michael Wooldridge Katia P. Sycara, editor, *Proceedings of the Second International Conference on Autonomous Agents*, pages 385-391, St. Paul, Minnepolis, USA, May 9-13 1998. ACM Press, New York.

WJK99

Michael Wooldridge, Nicholas R. Jennings, and David Kinny.
A methodology for agent-oriented analysis and design.
In *Proceedings of the third annual conference on Autonomous Agents*, pages 69-76, Seattle, WA USA, May 1999.

Woo96a

Michael Wooldridge.
Practical reasoning with procedural knowledge: A logic of BDI agents with know-how.
In *International Conference on Formal and Applied Practical Reasoning*. Springer-Verlag, June 1996.
Available from http://www.doc.mmu.ac.uk/STAFF/mike/papers.html.

Woo96b

Michael Wooldrige.
Agents as a rorschach test. a response to franklin and graesser.
In *Intelligent Agents III. Agent Theories, Architectures, and Languages*. Springer, 1996.

Woo99a

Michael Wooldridge.
Intelligent Agents, chapter Multiagent Systems and Societies of Agents, pages 27-79. The MIT Press, Cambridge, MA, 1999.

Woo99b

Michael Wooldrige.
Barriers to the industrial take-up of agent technology.
In PAAM99 [PAA99], page 11.

WPB99

T. White, B. Pagurek, and A. Bieszczad.
Network modeling for management applications using intelligent mobile agents.

Journal of Network and Systems Management, pages 295-321, September 1999.
Available at ftp://ftp.sce.carleton.ca/pub/netmanage/jnsm98-draft.zip.

WT92

Robert Weihmayer and Ming Tan.
Modelling cooperative agents for customer network control using planning and agent-oriented programming, 1992.

YdS96

Yechiam Yemini and Sushil da Silva.
Towards programmable networks.
In *Proceedings of the Distributed Systems and Operations Management Workshop - DSOM'96*, L'Aquila, Italy, October 1996.

Yem93

Yechiam Yemini.
The osi network management model.
IEEE Communications Magazine, 31(5):20-29, May 1993.

YGY91

Yechiam Yemini, Germán Goldszmidt, and Shaula Yemini.
Network Management by Delegation.
In *The Second International Symposium on Integrated Network Management*, pages 95-107, Washington, DC, April 1991.

Zad84

Lotfi A. Zadeh.
Making computer think like people.
IEEE Spectrum, (1):26-32, 1984.

Zap97

Michael Zapf.
Design paradigms in agend-based systems.
In *Distributed Applications and Interoperable Systems*, pages 101-107, Cottbus, Germany, 1997. Chapman & Hall.
ftp://www.vsb.cs.uni-frankfurt.de/pub/papers/1997/dpiabs.pdf.zip.

ZHGW99

Michael Zapf, Klaus Herrmann, Kurt Geihs, and Johann Wolfgang.
Decentralized SNMP management with mobile agents.
In Sloman et al. [SML99].

ZLH96

Simon Znaty, Michel Lion, and Jean-Pierre Hubaux.
Deal: A delegated agent language for developing network management functions.
In *First International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agent Technology*, 1996.