MASTER EN SCIENCES INFORMATIQUES

Generix : un agent générique pour la gestion des systèmes distribués

Libert, Pascal

Award date:
1996

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES
NOTRE-DAME DE LA PAIX NAMUR

Institut d'Informatique

Rue Grandgagnage, 21
B-5000 Namur (Belgium)

Generix : un agent générique pour la gestion des systèmes distribués

Pascal Libert

3ÈME MAÎTRISE EN INFORMATIQUE
ANNÉE ACADÉMIQUE 1995-1996

LBS 6847664

307354

PASCAL LIBERT

Generix : un agent générique pour la gestion des systèmes distribués

---

Promoteur : Jean Ramaekers

Mémoire présenté en vue de l'obtention du grade de Maître en Informatique

Année Académique 1995-1996

Résumé

De par leur croissance et leur complexité, la gestion des systèmes distribués est d'une importance capitale.

Les agents de gestion jouent un rôle essentiel dans la gestion de tels systèmes. Les agents sont utilisés pour récolter des informations de gestion, créer, supprimer et changer l'état des objets gérés, ainsi que pour notifier les événements des objets gérés vers le manager. Il n'existe à notre connaissance aucune spécification précise d'agents de gestion *génériques*. Dès lors, leur développement est actuellement difficile et fastidieux.

Ce mémoire a l'intention de présenter la gestion des systèmes distribués. L'architecture et les services de ***generix***, l'agent générique développé pour la plate-forme logicielle d'ISM/OpenMaster de Bull sera au cœur même de notre sujet. Nous verrons également comment ***generix*** peut s'inscrire dans la philosophie d'un agent intelligent.

Abstract

The growth and complexity of distributed systems has made their management a vital concern for today.

Management agents play an important role in such systems management. Agents are used to gather information, create, delete and change the state of management objects, and forward notifications of events from managed objects to manager. Yet, there is no precise specification of *generic* management agents. As a result, developing management agents at present is difficult and time-consuming.

This thesis presents an overview of distributed systems management. A strong focus is placed on the architecture and services of ***generix***, a generic agent developed on the Bull's ISM/OpenMaster software. We will also see how ***generix*** can be seen as an intelligent agent.

Remerciements

Ce mémoire n'aurait pu se faire sans l'aide ni le support d'un ensemble de personnes qu'il convient ici de remercier.

Tout d'abord, Mr Jean Ramaekers, mon promoteur, qui m'a offert l'opportunité d'effectuer mon stage chez Bull sur un sujet qui était très enrichissant. Ses conseils auront été pour moi une aide précieuse.

J'aimerais remercier ensuite Hughes Deghorain qui m'aura aidé avant, pendant et après le stage : ses idées et ses commentaires détaillés sur les différents essais de ce mémoire m'ont été fort utiles.

Merci aussi à Christian Ritter qui m'a accueilli au sein de son équipe et qui m'a fait confiance durant les cinq mois de stage.

S'il y a bien une personne que je dois remercier, c'est Philippe. Je ne saurais oublier qu'il s'est occupé de moi pendant toute la phase de développement, toujours disponible, cherchant à allier mon intérêt et celui de Bull. Il a sans aucun doute contribué à mon perfectionnement de la programmation.

Merci enfin à tous ceux qui ont apporté une aide plus ponctuelle, en particulier Pascal qui m'a donné des conseils pour la partie théorique ainsi que Frédéric qui m'a aidé lors du développement de l'agent.

Sans oublier ma famille et tout spécialement mes parents pour leur soutien tout au long de mes études universitaires. Sans eux, rien ne serait jamais arrivé. Un merci tout particulier à Marlène dont le soutien et la compréhension m'ont été nécessaires pour terminer ce mémoire.

Table des matières

Table des illustrations

1. Introduction

De part leur dépendance sans cesse croissante avec un système distribué, les entreprises sont de plus en plus exposées aux risques inhérents à de tels systèmes. Dysfonctionnement, baisse de performances, allocation optimale des ressources, sécurité sont autant de problèmes étroitement associés à un système distribué.

Des méthodologies et des outils de gestion sont nécessaires afin d'aider les administrateurs à réduire autant que possible l'occurrence de ces problèmes. Une gestion efficace exige la surveillance, l'interprétation et le contrôle du comportement des ressources matérielles et logicielles hétérogènes d'un système distribué. Pour être efficaces, les administrateurs doivent sans cesse maîtriser le volume et la complexité qui caractérisent les systèmes distribués actuels. Leur gestion, aussi ardue soit elle, est simplifiée par l'existence d'environnements intégrés qui nous sont proposés par de nombreux constructeurs (Bull, HP, IBM,...). Jusqu'aujourd'hui, la plupart des constructeurs avaient opté pour la mise en place d'une architecture d'environnement de gestion fortement centralisée. Cette gestion est réalisée au moyen d'un manager central responsable d'exécuter les applications de gestion et d'agents situés sur le système tout entier, responsables de maintenir et de fournir les informations de gestion au manager. On a très vite constaté que l'empilement des nombreuses applications sur une seule machine entraîne des problèmes non négligeables de performances de part le taux d'utilisation des moyens de communication.

C'est ainsi que de nouvelles architectures de gestion sont nées. Parmi celles-ci, on retrouve des architectures distribuées où les agents deviennent des entités intelligentes, car certaines applications de gestion sont déléguées du manager vers l'agent.

Cependant, il n'existe, à notre connaissance, aucune spécification précise d'agents de gestion *génériques*. Dès lors, leur développement peu être qualifié de difficile et fastidieux. L'objectif recherché lors de la rédaction de ce mémoire a été de présenter aussi bien que possible un agent générique, appelé parfois "agent extensible", et également intelligent.

Ce mémoire s'articule autour de deux parties. La première ne poursuit d'autres buts que de fournir au lecteur les bases théoriques nécessaires. Nous n'aurions pu présenter *generix* sans avoir défini, au préalable, les concepts théoriques sur lesquels cet agent vient se

greffer. La seconde partie sera axée sur une étude de cas et sur l'explication de *generix*. C'est pourquoi nous l'avons appelée "partie pratique".

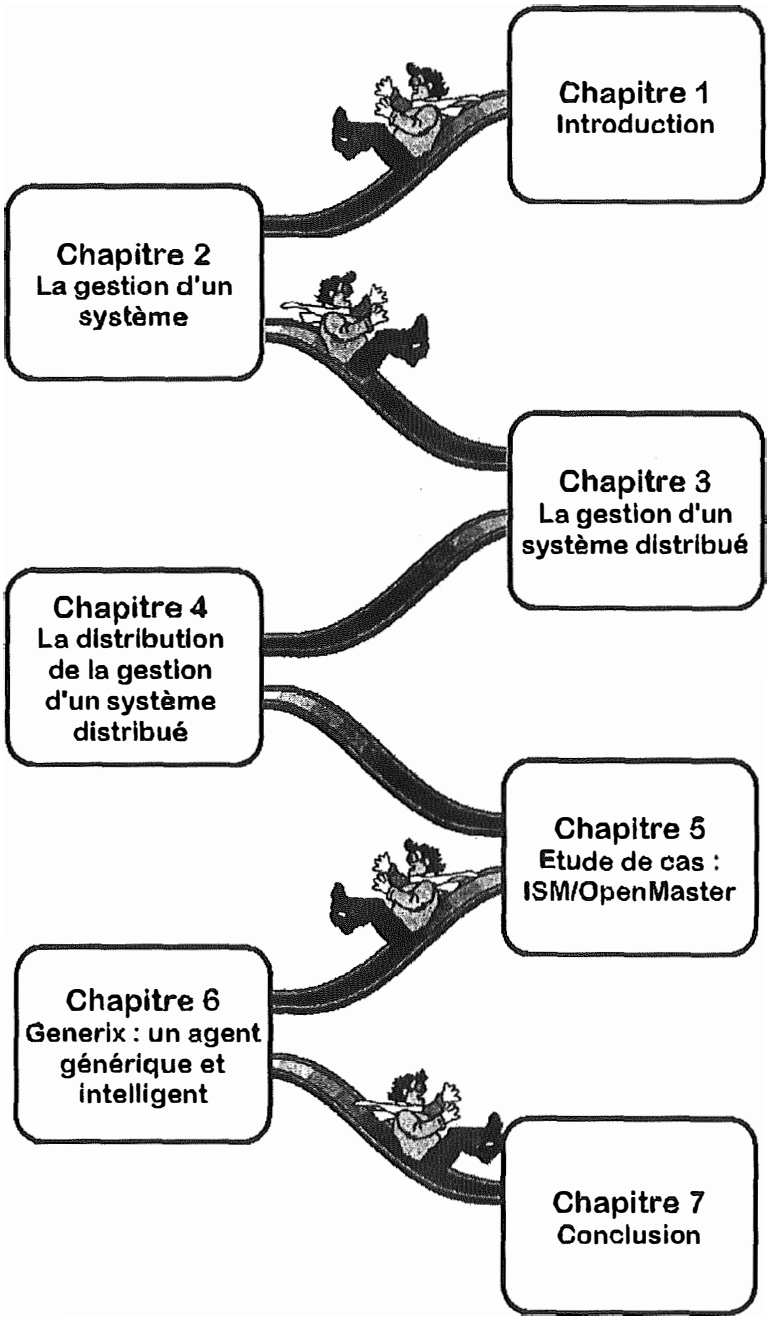

Figure 1 : Plan du mémoire

Comme on peut le remarquer sur la figure 1, le mémoire est décomposé en plusieurs chapitres. Ceux-ci ont la prétention de devoir se lire l'un après l'autre car la structure choisie suit une ligne générale de raffinements.

Dans le chapitre 2, nous nous familiariserons avec la gestion d'un système informatique et ses principales caractéristiques. Il s'agira tout d'abord de définir la notion de gestion d'un système informatique. Nous tenterons ensuite de souligner l'importance attachée à une gestion de système. Finalement, nous présenterons les deux principaux modèles de gestion.

Parce que *generix* s'insère dans un environnement distribué, nous devrons aborder les systèmes distribués et leurs concepts. C'est ce que tentera de faire le chapitre 3. Nous insisterons sur la nécessité de disposer d'outils de gestion pour de tels systèmes ainsi que sur les services de gestion d'un système distribué. Nous détaillerons également un protocole de gestion fortement utilisé parce qu'il est à la base d'une architecture de gestion centralisée. Il s'agira du protocole SNMP.

Le modèle de gestion centralisée est un paradigme important mais il présente certaines lacunes. Ces dernières peuvent être annihilées en adoptant d'autres modèles de gestion. Le chapitre 4 se concentrera sur une présentation de trois modèles différents. Nous essaierons par la même occasion de soulever les caractéristiques qui nous permettent de savoir quel type de modèle doit être pris.

Ces trois chapitres composeront la partie théorique de ce mémoire. Quant aux deux chapitres suivants, ils viennent illustrer la partie pratique.

Dans le chapitre 5, nous aurons l'occasion de découvrir une solution de gestion proposée par un constructeur, en l'occurrence, ISM/OpenMaster de Bull. Nous pourrons ainsi voir comment une plate-forme de gestion d'un système distribué peut s'articuler autour des différents modèles de gestion proposés.

Le chapitre suivant présentera le résultat de notre stage effectué chez Bull. Il s'agira d'un agent générique facilitant le développement de nouveaux agents SNMP tout en s'inscrivant dans la philosophie d'un agent intelligent.

La conclusion clôturera ce mémoire.

Partie 1

Partie théorique

OBJECTIFS

- établir les bases théoriques nécessaires
- expliquer le concept général de gestion
- introduire le concept de système distribué
- commenter la gestion d'un système distribué
- décrire les fonctions de gestion propres à un système distribué
- souligner les paradigmes de gestion d'un système distribué

2. La gestion d'un système.

2.1 Introduction

Les premiers systèmes que l'homme aura probablement gérés ont été les réseaux de transport comme, par exemple, les voies romaines. Il fallait repérer et remplacer les pavés cassés après leur signalement par des voyageurs ou des inspecteurs (notions d'alarme et de surveillance), décider des parties à paver ou à laisser en terre, planifier les extensions (configuration), élargir les routes aux goulots d'étranglement (performances), faire payer l'octroi (comptabilité) et instaurer un service de gendarmerie (sécurité). Toutes les fonctions de base de la gestion d'un système informatique actuel se retrouvent au travers de cet exemple.

Les réseaux de transmission de données[1], principalement développés au cours des années 70, ont été souvent conçus sans y incorporer des mécanismes de gestion. Les traditions techniques du milieu informatique s'étaient formées sur des systèmes de petite taille, équipés de quelques dizaines de terminaux localisés à courte distance. Dans ces conditions, la dispersion géographique n'était pas encore perçue comme un changement d'échelle.

La prolifération des systèmes informatiques à dimension nationale ou internationale a pris de court leur conception technique, souvent restée rudimentaire au plan des mécanismes de gestion. Cette prolifération va de pair avec une croissance considérable des communications de ces systèmes. C'est pourquoi, face à la complexité sans cesse croissante des systèmes informatiques et de leurs communications, de bonnes facilités de gestion doivent être développées et réalisées. Avant de proposer une nouvelle solution[2], il est important de connaître la signification exacte de la gestion d'un système.

L'objet de ce chapitre est de présenter une définition de la gestion ainsi que certains de ses aspects qui sont bien souvent oubliés dans la littérature. Dans la section 2.2, nous

[1] [ARPEGE,92]

[2] Dans l'étude de cas, on retrouvera une description de l'agent *generix* développé chez Bull S.A. et qui permet de répondre à de nouveaux besoins de gestion.

tenterons de donner une définition de la gestion. Nous essaierons ensuite de comprendre pourquoi il est important de gérer un système informatique. Pour sa part, la section 2.4 tentera de définir les principales méthodes de gestion. La gestion d'un système distribué, cas particulier d'un système informatique, sera abordée dans le chapitre suivant.

2.2 Qu'est-ce que la gestion[3] ?

Bon nombre de définitions nous sont proposées dans la littérature. La plupart de celles-ci sont données par des organismes de standardisation qui utilisent une terminologie précise et qui portent sur des champs spécifiques d'application. En ce qui nous concerne, nous avons choisi une définition de la gestion d'un système informatique qui se veuille la plus générale possible. Nous étendrons cette définition au chapitre suivant lorsque nous concentrerons notre étude sur le problème lié à la gestion d'un système distribué.

2.2.1 Les activités et les fonctions de gestion

Définition :

La gestion d'un système est le fait d'initialiser, de surveiller et de modifier le fonctionnement des fonctions primaires du système.

La gestion assure une utilisation efficace du système ainsi qu'un respect des objectifs de qualité des services souscrits.

Dans le cycle de base d'une activité de gestion, illustré à la figure 2, on voit apparaître les trois types d'activités prenant place lors de la phase opérationnelle[4] :

- la surveillance du système afin d'obtenir les informations adéquates,
- les prises de décisions sur base de ces mêmes informations complétées par une politique de gestion,

[3] Le terme *management* a délibérément été traduit par gestion plutôt qu'administration.

[4] La phase opérationnelle d'un système informatique prend en compte la période durant laquelle ce système fonctionne.

- la réalisation d'actions de gestion afin de contrôler le système.

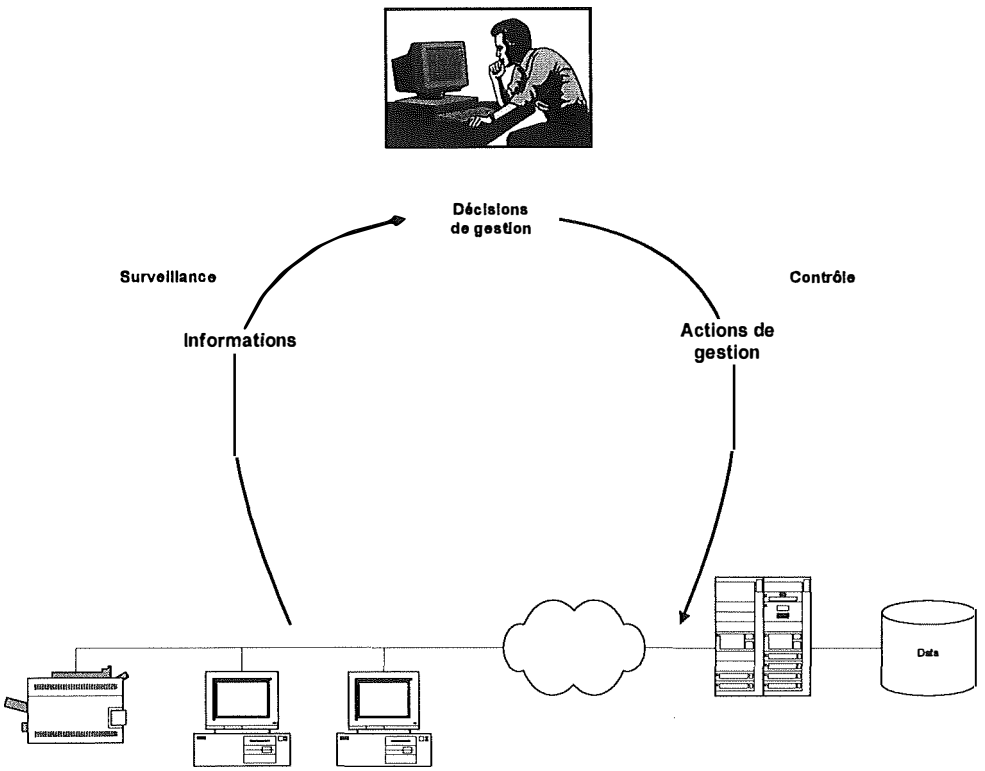

Figure 2 : Le cycle de gestion

Dans notre définition de la gestion, nous avons parlé de fonctions primaires. Nous ne pourrions continuer sans expliquer ce qu'elles représentent.

Les fonctions primaires sont les fonctions qui sont directement nécessaires à la gestion pour maintenir l'état opérationnel d'un système. Cela permet de s'assurer que le système se comporte correctement et efficacement. D'un autre côté, les fonctions secondaires fournissent des caractéristiques de gestion qui ne sont pas intrinsèques au but premier qui n'est autre que de maintenir un système en état de marche.

Le respect des objectifs de qualité des services souscrits se réalise en assurant une certaine disponibilité des ressources du système, en gardant un temps de réponse raisonnable et ce, quels que soient les changements apportés au système.

2.2.2 Aires fonctionnelles de gestion

La gestion est nécessaire afin de mettre sur pied et de garder opérationnels les systèmes qui réalisent les fonctions primaires. Ceci implique que la gestion doit avant tout initialiser les différentes ressources du système. Si aucune erreur n'intervient, le système devient aussitôt opérationnel. L'outil de gestion surveille alors les différents sous-systèmes afin de voir si un problème quelconque survient. Si tel est le cas, les présumés problèmes sont identifiés, isolés et réparés. Si les ressources défectueuses ne peuvent être réparées, elles seront remplacées par de nouvelles ressources qui seront à leur tour initialisées. D'un autre côté, la surveillance du système permet la détection des changements dans le flux de trafic. Dès qu'un changement est détecté, certains paramètres du système peuvent être modifiés afin d'optimiser les performances. La gestion d'informations comptables définit, pour chaque ressource facturable, les limites comptables et ses coûts d'utilisation, les tarifs, et le coût global d'une consommation. Finalement, la gestion de la sécurité doit permettre la mise en œuvre de politiques de sécurité (création, contrôle et suppression de mécanismes et de services de sécurité, compte-rendu d'événements, diffusion d'informations de sécurité).

Les aires fonctionnelles, mentionnées ci-dessus, sont illustrées à la figure 3 et feront l'objet d'une étude un peu plus approfondie dans le chapitre suivant..

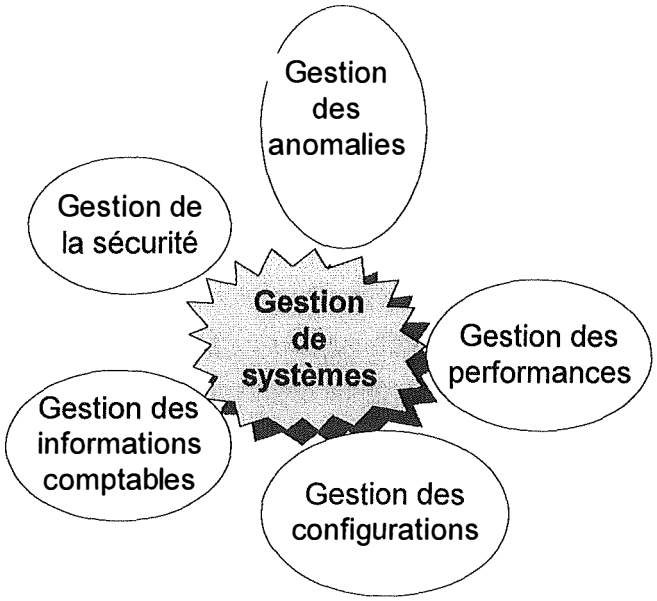

Figure 3 : Aires fonctionnelles relatives à la gestion de systèmes

2.3 Pourquoi doit-on gérer ?

[PRAS,95] nous fait remarquer qu'il est intéressant de constater que la littérature se focalise généralement sur *"ce que"* les opérations de gestion réalisent et que peu est publié à propos du *"pourquoi"* de l'existence de telles fonctions. Cependant, les enjeux de la gestion d'un système informatique sont nombreux.

2.3.1 L'importance d'avoir une gestion

Un outil de gestion doit avant tout contrôler le système pour lequel il est implanté, quelles que soient l'architecture et la complexité de ce dernier, de manière à satisfaire les diverses populations qui le mettent en œuvre et qu'il dessert.

Les services pris en charge par la gestion s'étendent à toutes les fonctions nécessaires pour rendre un quelconque système opérationnel. En d'autres termes, le système informatique doté d'un système de gestion doit savoir gérer son état de marche, ses performances, ses fautes, ses modifications, sa sécurité, son parc matériel et ses configurations.

La figure 4, proposée par [ARPEGE,92], identifie les différentes perceptions que peuvent avoir les personnes qui interagissent avec le système. Nous remarquons à quel point il est essentiel d'avoir une gestion qui soit efficace vu la diversité des utilisateurs ainsi que de leurs intérêts face au système.

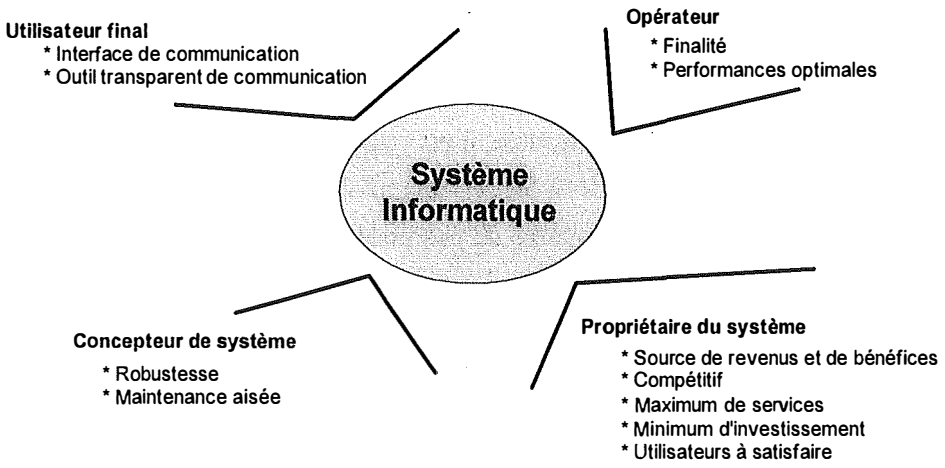

Figure 4 : Les catégories d'utilisateurs d'un système informatique

2.3.2 La réduction des coûts

Les utilisateurs finaux recherchent toujours le meilleur système possible au meilleur prix, à savoir donc au prix le plus bas possible. Une manière de satisfaire ce besoin est de répartir les coûts de conception non plus sur un seul utilisateur mais sur plusieurs d'entre eux. Ceci signifie que la conception ne doit pas répondre aux exigences spécifiques d'un utilisateur, mais elle doit être aussi générale que possible (Figure 5). Nous verrons dans la suite comment l'agent que nous avons développé peut répondre à un tel besoin.

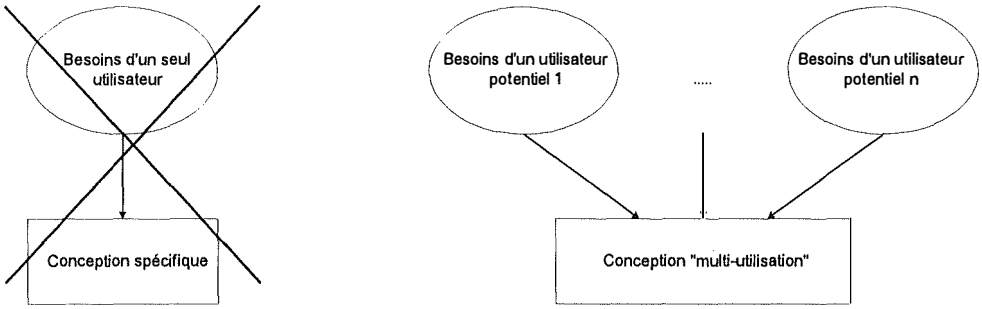

Figure 5 : Vers une conception "multi-utilisation" d'un système

La conception doit donc être une conception "multi-utilisation", ce qui signifie que les fonctions de gestion doivent répondre aux exigences d'un grand nombre d'utilisateurs.

2.3.3 Manque d'expérience

De rapides développements sont faits dans le domaine des systèmes informatiques. En peu de temps, leurs capacités ainsi que leur utilisation se sont accrues considérablement. En conséquence, le concepteur est confronté à de multiples problèmes. Parce que l'expérience de tout concepteur est limitée, il est quasiment impossible de s'attendre à ce qu'il trouve toutes les solutions à tous les problèmes durant la phase de conception. Pour certains problèmes, il est donc préférable de postposer la recherche de solutions jusqu'au moment de la phase opérationnelle. Résoudre de tels problèmes sera sous la responsabilité de l'outil de gestion qui fonctionnera à ce moment. L'avantage de cette approche est que l'on espère obtenir de plus amples expériences pendant la phase opérationnelle, ce qui apportera de l'aide dans la résolution de ces problèmes. [PRAS,95].

Afin de mieux comprendre ce problème, nous avons choisi de l'illustrer au moyen de l'exemple qui va suivre.

Exemple :

Le contrôle de la congestion est un de ces problèmes qui n'a pas encore été résolu de manière générale. Ceci est dû aux différentes causes possibles de congestion, chacune réclamant sa propre mesure de solution. Le premier problème soulevé dans la suite est illustré par le premier schéma de la figure 6, tandis que le second par le deuxième schéma.

☑ Dans un show télévisé, les téléspectateurs sont invités à appeler le studio. Ceci provoquera une surcharge du réseau téléphonique, auquel cas des solutions devront être prises pour éliminer ce problème. Une stratégie possible serait de limiter les tentatives d'appel vers le studio.

☑ Supposons que nous nous trouvions au carnaval de Rio. Beaucoup de personnes restent en ville et le réseau téléphonique est surchargé. Pour éviter une telle surcharge dans la zone de Rio, les appels entre les autres villes qui étaient habituellement déviés sur Rio seront renvoyés vers d'autres villes proches de Rio.

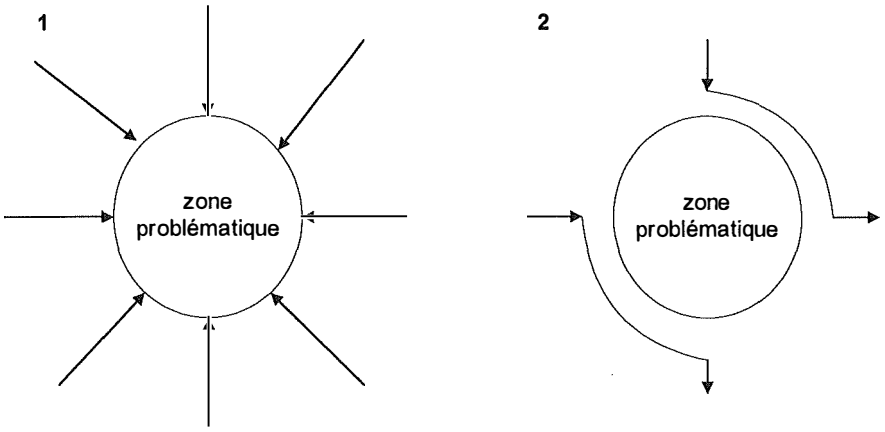

Figure 6 : Deux exemples de congestion.

Ces deux exemples illustrent le fait qu'il n'est pas toujours possible d'anticiper tous les problèmes lors de la phase de conception. Certains doivent donc être résolus lors de la phase opérationnelle. Un outil de gestion bien conçu offrira à l'opérateur humain les solutions les plus adéquates.

2.3.4 La manipulation des fautes

Lors d'une phase opérationnelle, des fautes peuvent survenir soudainement. Les fautes sont des situations dans lesquelles les ressources, tant matérielles et logicielles, ne se comportent pas comme prévu. D'une part, le système ne pourra plus fournir le service adéquat et d'autre part, il pourra même être complètement arrêté. Pour qu'un outil de gestion fonctionne correctement, les ressources matérielles et logicielles doivent fonctionner[5]. L'occurrence des fautes peut être provoquée par le hardware lui-même, aussi bien que par des erreurs humaines. La probabilité qu'une faute[6] arrive dépend de :

- la qualité des ressources du système informatique : pour un prix donné, une ressource de tel ou tel constructeur connaîtra une plus faible probabilité de faute qu'une autre. Il est donc utile et nécessaire de s'assurer du bon fonctionnement de ces ressources avant même de se préoccuper d'un outil de gestion.
- la manière de travailler : dans de nombreux cas, les erreurs humaines proviennent d'un manque de familiarité de la part des utilisateurs ou d'un non-respect des règles.

Parce qu'il n'est pas possible d'empêcher toutes les fautes et parce que ces dernières peuvent avoir de lourdes conséquences, le système doit être contrôlé par l'outil de gestion durant la phase opérationnelle. Un tel contrôle induit la *prédiction* des fautes possibles, la *détection* de fautes existantes, la *diminution* des effets causés par les fautes ainsi que leur *réparation.*

Afin de prévenir et de détecter les fautes, les gestionnaires doivent avoir la possibilité de :

- surveiller le comportement des différentes ressources composant le système,
- comparer le comportement actuel avec des comportements passés et/ou attendus,
- signaler des comportements exceptionnels.

---

[5] Geurts P. : *« Gestion des systèmes coopératifs »*, Cours de 3ème maîtrise en informatique, Philippe Van Baestelaer, Mai 1996.

[6] Cette faute peut prendre, par exemple, la forme d'une panne matérielle.

Pour réduire les effets des fautes et pour favoriser la réparation, l'outil de gestion doit avoir la possibilité de modifier l'état du système informatique sur lequel il est implanté.

2.3.5 La flexibilité

La conception d'un réseau et de tout système a un certain cycle de vie[7]. L'un des points primordiaux d'un tel cycle relève de l'importance des besoins des utilisateurs : la conception part de la définition des besoins des utilisateurs et la plupart des décisions relatives à la conception proviennent de ces besoins.

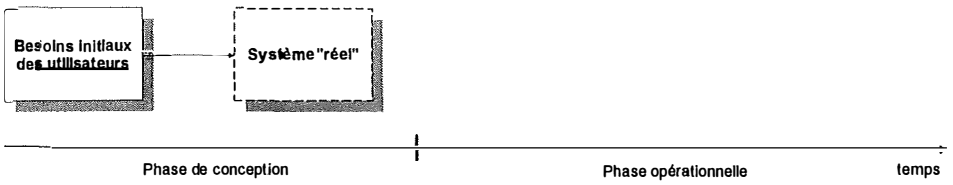

Figure 7 : Le processus de conception simplifié

Le danger de regarder le processus de conception tel qu'il est schématisé à la figure 7 est que l'on risque d'oublier le caractère dynamique des besoins des utilisateurs. En réalité, leurs besoins changent régulièrement. C'est pourquoi ils ne doivent pas être vus comme des entités purement statiques (Figure 8).

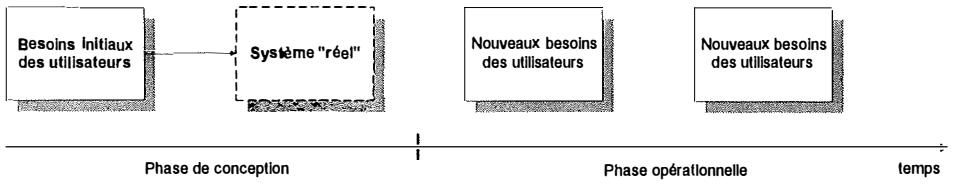

Figure 8 : Les besoins des utilisateurs dans le cycle de vie

Au lieu de construire un nouveau système chaque fois qu'un besoin change, il est préférable d'avoir une certaine dose de flexibilité au sein du système informatique. Grâce à cette flexibilité, le gestionnaire pourra réagir en conséquence face aux changements des besoins lors de la phase opérationnelle. Le concepteur doit ainsi anticiper cette volonté et doit ajouter un certain nombre de fonctions de gestion lors de la conception. Nous verrons dans la suite de ce mémoire que cette flexibilité existe sous *generix*, l'agent générique.

[7] Dubois P., *« Méthodologie de développement de logiciels »*, FUNDP, Institut d'Informatique, 1994

2.4 Comment peut-on gérer un système ?

Lorsqu'il conçoit des fonctions de gestion, le concepteur est confronté à de nombreuses questions. Deux d'entre elles sont particulièrement intéressantes, car elles affectent le processus de conception de manière non négligeable. Ces questions sont :

- les fonctions de gestion vont-elles être réalisées par des humains ou par des modules hardware ou software ?
- les fonctions de gestion vont-elles être distribuées à travers tout le système ou doivent-elles être aussi concentrées que possible ?

La réponse à la première de ces deux questions nous amène à considérer le caractère d'une gestion explicite d'une part, et implicite d'autre part.

Quant à la seconde, nous allons y répondre en examinant brièvement deux modèles de gestion que nous qualifions d'opposés : la gestion centralisée et distribuée.

2.4.1 La gestion explicite et implicite

Pour définir le fait que les opérations de gestion sont à caractères humains, nous utiliserons, tout comme [PRAS,95], le terme de *'gestion explicite'*. Dans cette forme de gestion, la décision d'initialiser des fonctions de gestion est prise explicitement par des opérateurs humains lors de la phase opérationnelle.

L'opposé de la gestion explicite sera la *'gestion implicite'*. Avec cette forme de gestion, toutes les fonctions de gestion seront prises par des modules hardware ou software : l'intervention humaine n'est dès lors pas nécessaire.

L'un des avantages de la gestion explicite est que l'on ne doit pas nécessairement élaborer toutes les fonctions de gestion lors de la phase de conception. Ceci est particulièrement vrai pour les fonctions qui déterminent à quel moment une opération spécifique de gestion doit être initialisée et quelles valeurs doivent être sélectionnées pour réaliser un but précis. La gestion explicite est particulièrement utile pour résoudre des problèmes inattendus qui apparaissent lors de la phase opérationnelle et qui demandent l'invention de nouvelles solutions. Cette forme de gestion est donc appropriée pour la gestion des

fautes. Par contre, l'un de ses inconvénients est son temps de réponse, trop long par rapport à la gestion implicite.

En général, un outil de gestion combine les deux formes de gestion : certains problèmes de gestion sont résolus implicitement tandis que d'autres exigent l'utilisation d'une intervention humaine.

Exemple :

Typiquement, un outil de gestion possédant une interface graphique peut être qualifiée d'explicite. En effet, l'initiative de déclencher telle ou telle application de gestion, comme par exemple la visualisation du nombre de pages imprimées par l'utilisateur X sur une imprimante Y, incombe à l'opérateur humain.

A l'inverse, une application de gestion telle qu'une migration de processus peut être lancée automatiquement, c'est-à-dire sans intervention humaine, lorsque le taux d'occupation du processeur d'une machine a atteint un certain seuil.

2.4.2 La gestion centralisée et distribuée

Le terme centralisé est utilisé dans le sens où des décisions de gestion sont prises sur un seul endroit. Ces décisions sont prises par le ***manager*** : il représente ce que l'on peut considérer comme l'intelligence de gestion et est souvent vu comme étant l'***application*** de gestion.

Pour gérer les opérations des fonctions primaires, des ***agents*** doivent être ajoutés aux systèmes. De tels agents représentent le support de gestion au travers desquels le manager initialise, surveille et modifie le comportement des fonctions primaires. Comparés aux managers, les agents sont bien souvent plus simples.

Avec une gestion centralisée, un grand nombre d'agents peuvent être contrôlés par un seul manager (Figure 9). Pour permettre leur communication, un protocole de gestion est nécessaire. Des exemples de tels protocoles nous sont donnés par [DEGHOR,92]. Ce sont le Common Management Information Protocol (CMIP) et le Simple Network Management Protocol (SNMP). Le premier est un standard *de jure* proposé par l'ISO [ISO9596] tandis que le second est un standard *de facto* lié à l'environnement Internet TCP/IP [RF1157,90].

L'inconvénient majeur inhérent à ce modèle de gestion, c'est que l'outil de gestion peut être arrêté suite à un dysfonctionnement au niveau du manager. De plus, une gestion centralisée accroît l'utilisation des moyens de transmission de part la quantité d'informations échangée entre le manager et ses agents.

Le terme distribué est opposé à la gestion centralisée dans le sens où les applications de gestion ne se retrouvent plus sur un manager unique. Les caractéristiques d'une gestion distribuée proviennent de ce que les agents deviennent des entités intelligentes puisqu'ils sont responsables d'effectuer certaines applications de gestion.

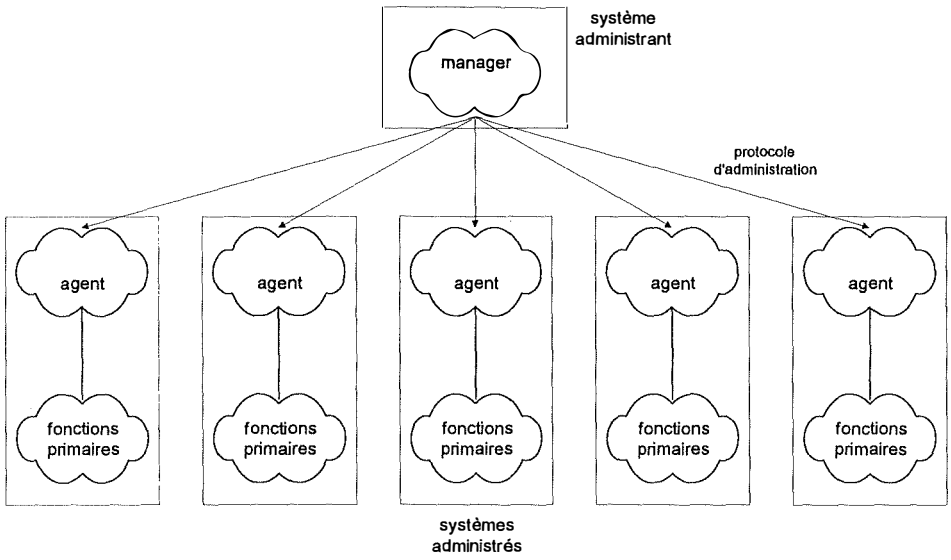

Figure 9 : Gestion centralisée

Ces deux modèles de gestion ainsi qu'un troisième feront l'objet d'une étude un peu plus approfondie dans le chapitre 4.

2.5 Conclusion

La gestion d'un système doit s'analyser selon deux axes : l'axe temporel et l'axe fonctionnel puisque l'échelle du temps à laquelle on se place est une composante essentielle à sa réalisation. La vérification de l'état de marche d'un système (mesurer les temps de réponse, planifier, contrôler, anticiper sur l'évolution probable d'un protocole,...) doit être considérée comme stratégique.

Un système requiert un pilotage en souplesse adapté à sa structure dynamique, sujette - tout comme nous l'avons illustré dans ce chapitre - à des modifications fréquentes. De plus, il est vulnérable, menacé de pannes, de dysfonctionnements et d'utilisation anarchique de ses ressources. La vulnérabilité et la complexité d'un système, et plus encore d'un système distribué, s'accroissent avec sa taille et les niveaux d'hétérogénéité qu'il sous-tend.

La gestion d'un système, quel que soit son modèle, est à prendre en compte lors de sa conception. Elle doit être modulaire et doit pouvoir se mettre en œuvre aussi bien explicitement qu'implicitement.

Réaliser une bonne gestion d'un système est un pari difficile à tenir. Les chapitres suivants de ce mémoire s'attacheront à apporter une meilleure compréhension des différents modèles de gestion du système qui est au cœur de notre mémoire : le système distribué.

3. La gestion d'un système distribué

3.1 Introduction

Bien que la terminologie et les concepts liés à la gestion d'un système ont été énoncés au chapitre précédent, ce chapitre n'a d'autres ambitions que de les raffiner tout en les adaptant à la gestion d'un système distribué, qui constitue lui-même le paysage sur lequel vient se greffer *generix*, notre solution de gestion.

Nous ne pourrions parler de gestion de système distribué sans savoir exactement ce qu'est un système distribué. C'est pourquoi, les trois premières sections auront pour but d'éclaircir cette notion. Nous pourrons ensuite aborder le sujet principal de ce chapitre en décrivant les difficultés liées à la gestion de tels systèmes. Ceci sera expliqué dans la section 3.5. Avant de conclure, nous verrons d'un peu plus près l'architecture du Simple Network Management Protocol fortement utilisé dans la gestion des systèmes distribués. Ce protocole se base sur le modèle de gestion centralisé que nous avons déjà vu dans le chapitre précédent et que nous détaillerons au chapitre suivant.

3.2 Echelle des systèmes

Comme le souligne [DEGHOR,92], *"un système distribué peut être vu comme le sommet d'une échelle sur laquelle les systèmes informatiques se placent en fonction de leur degré de centralisation. L'échelon le plus bas est occupé par les systèmes centralisés, alors que les systèmes distribués prennent place sur la barre la plus élevée. Entre ces deux extrêmes, une étape intermédiaire est constituée des réseaux."*

Au niveau le plus bas de l'échelle, viennent s'installer les systèmes centralisés. On peut facilement imaginer ce que représente un système centralisé en l'identifiant au concept de "salle-ordinateur". Une salle d'ordinateur est un lieu où les utilisateurs apportaient leurs travaux à traiter, au moyen par exemple de cartes perforées. Or, ce modèle présente au

moins deux défauts : celui de l'ordinateur unique autonome[8] qui fait tout, et l'idée que les utilisateurs devaient apporter le travail à l'ordinateur plutôt que l'inverse.

Même si ce concept relève de la "préhistoire" de l'informatique[9], nous pouvons tout simplement illustrer le concept de système centralisé par le biais des ordinateurs individuels. On peut difficilement trouver, à notre avis, meilleur exemple pour décrire ce concept.

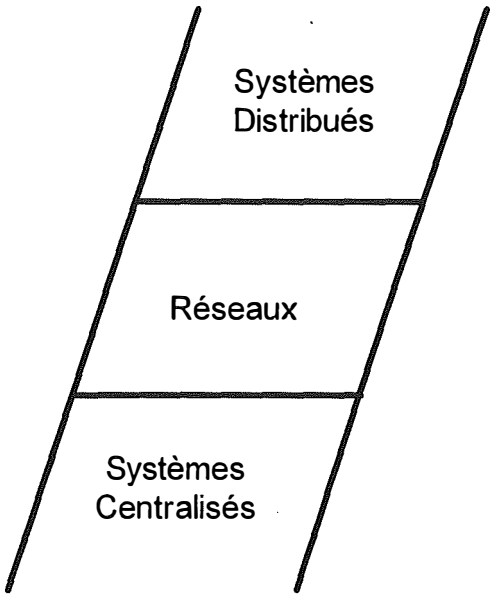

Figure 10 : Echelle des systèmes selon leur degré de centralisation

Ce modèle de l'ordinateur unique desservant tous les besoins de traitement de l'organisation s'est trouvé supplanté par celui de l'exécution du travail par des ordinateurs séparés mais interconnectés. De tels systèmes sont appelés réseaux d'ordinateurs et se retrouvent sur l'échelon intermédiaire entre les systèmes centralisés et distribués. Tout comme le souligne [TANENB,93], *"le terme réseaux d'ordinateurs signifie un ensemble interconnecté d'ordinateurs autonomes."*. Deux ordinateurs sont considérés comme interconnectés s'ils sont capables d'échanger des informations. Dans un réseau, on retrouvera donc plusieurs ordinateurs autonomes possédant leur propre système

[8] Lorsque nous parlons d'ordinateur unique autonome, nous voulons souligner par là qu'il n'existe qu'un seul processeur.

[9] Si nous nous sommes permis d'utiliser le terme *préhistoire*, c'est parce que nous imaginons mal qu'actuellement, un utilisateur doive se déplacer physiquement vers l'ordinateur pour lui demander de traiter ses données.

d'exploitation, capables de réaliser une tâche complètement mais à la différence des systèmes centralisés, ces ordinateurs peuvent communiquer entre eux.

Pour le sommet de notre échelle, nous rejoindrons les remarques pertinentes de [DEGHOR,92]. et [TANENB,93]. La distinction fondamentale entre réseau d'ordinateurs et systèmes distribués réside dans le fait que, dans un système distribué, la répartition en ordinateurs autonomes est transparente, invisible à l'utilisateur.

Le lecteur intéressé par la transparence des systèmes distribués trouvera de nombreuses informations dans [VINCEN,94].

3.3 Une définition d'un système distribué

A la lecture de différents mémoires écrits par nos prédécesseurs, nous avons pu remarquer qu'il était bien difficile de donner une définition concrète des systèmes distribués. Par contre, en donner les caractéristiques était chose plus commune. Nous illustrerons à notre tour ces caractéristiques par le biais d'un exemple dans la section suivante.

Ce que nous constatons aujourd'hui, c'est qu'un poste de travail[10] fonctionne rarement de façon totalement indépendante. Connecté à un réseau informatique (Ethernet, FDDI,...), il utilise des **ressources externes** (imprimante, disque dur, processeur...) en faisant appel, de manière **transparente**, à des services informatiques dits **distribués** géographiquement sur le système contenant ce propre réseau. Basés sur des protocoles standardisés, ces services sont interopérables entre différentes plates-formes matérielles (milieu **hétérogène** provenant de multiples constructeurs) et le plus souvent basés sur le modèle client/serveur : la composante qui fait appel à un service étant le client, celui qui fournit ce service étant le serveur.

Nous retrouvons ici les différentes caractéristiques d'un système distribué. Il est inutile d'épiloguer à ce sujet mais avant de continuer, nous aimerions reprendre ici une définition qui nous paraît juste. Cette définition a été proposée par [SLOMAN,94]. En effet, même si cette définition n'énumère pas les caractéristiques de ces systèmes, nous pensons comme l'auteur que le plus important est ici le caractère d'interopérabilité entre les différentes ressources du système. Nous soulignerons une nouvelle fois qu'il ne faut pas oublier la transparence de ces systèmes.

Définition :

"A distributed system is composed of a number of autonomous processors and/or data stores supporting processes and/or data bases which interact in order to cooperate to achieve a common goal. The processes coordinate their activities and exchange information by means of information transferred over a communications network ..."

3.4 Un exemple de système distribué

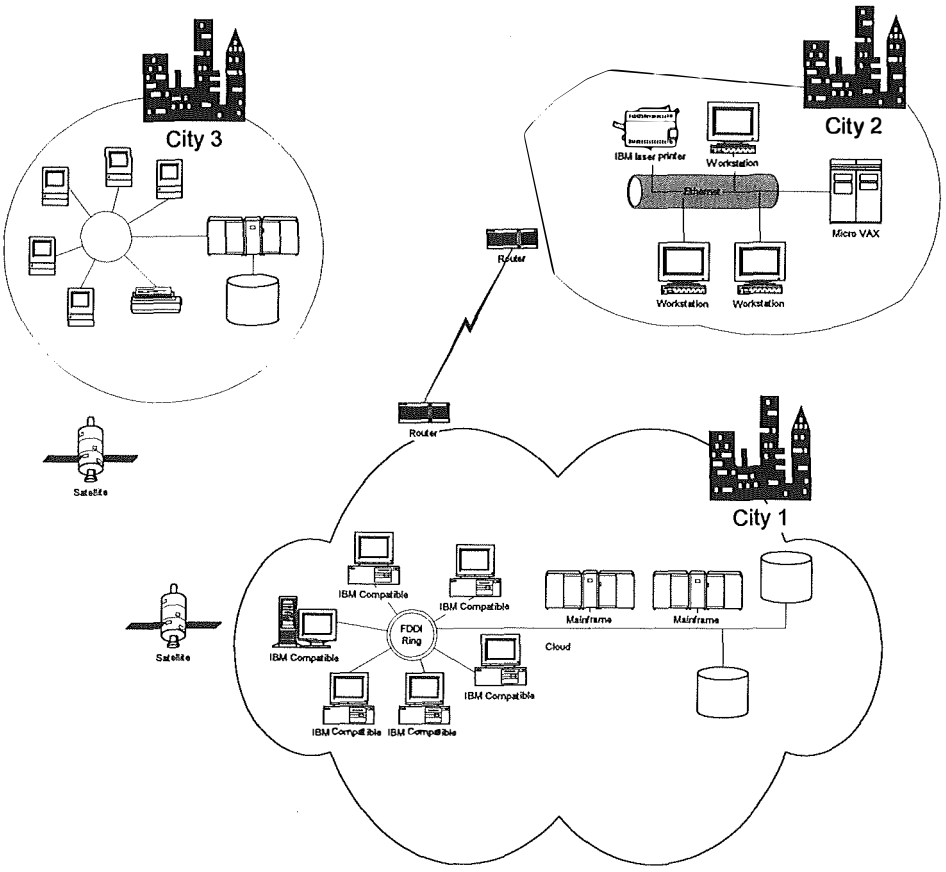

Figure 11 : Un exemple de système distribué

Dans cet exemple, on retrouve les caractéristiques principales d'un système distribué :

1°) **la distribution :** les ressources qui composent le système ne se trouvent plus dans une pièce, ni même dans un seul site mais peuvent être distantes de plusieurs

[10] Qu'il s'agisse d'une station UNIX ou d'un micro-ordinateur de type Mac ou PC

kilomètres. Ces ressources doivent être reliées l'une à l'autre via des moyens de transmission qui varient aussi bien en capacité, en services qu'en prix. Dans notre exemple, ces moyens sont représentés à l'aide de *satellites* ,de *LAN's*,...
En conséquence, les environnements distribués exigent une architecture qui distribue les tâches de gestion au travers de domaines géographiquement dispersés.

2°) **l'hétérogénéité :** depuis la fin des années 80, *l'hétérogénéité* a souvent décrit un système dont les équipements étaient fournis par différents constructeurs. Cependant, une nouvelle source d'hétérogénéité est née, à savoir les ressources qui doivent être gérées. Ces ressources sont au nombre de quatre : les réseaux, les systèmes, les applications ainsi que les bases de données. Dans un souci de lisibilité, nous n'avons pas représenté les ressources applications sur la figure 11.

Sous le vocable de ressources réseaux, on retrouve tout ce qui est multiplexeur, routeurs, passerelles,... Les ressources systèmes forment quant à elles les ordinateurs et les systèmes d'exploitation. Les applications représentent les ressources logicielles et finalement, on retrouve donc les ressources bases de données.
Chacune de ces ressources nécessite des besoins de gestion très divers.

3°) **le rendement d'échelle :** la taille d'un réseau était traditionnellement mesurée grâce au nombre de noeuds présents mais on calcule maintenant la taille d'un système distribué par le nombre de ses ressources. Nous devons ici nous convaincre qu'elles se sont accrues exceptionnellement. Un réseau composé de 100 noeuds présente sans aucun doute des besoins de gestion forts différents d'un système distribué, composé lui-même de plusieurs milliers de ressources.

4°) **la transparence :** même si nous ne pouvons représenter la transparence d'un système distribué, nous avons voulu reprendre ici l'une de ses caractéristiques majeures.

3.5 La gestion d'un système distribué

La gestion des systèmes a toujours été une fonction critique des systèmes d'information. La connaissance acquise depuis des années grâce à la gestion des réseaux est généralement applicable à la gestion des systèmes distribués. Or, les aspects spécifiques aux systèmes distribués rendent sa gestion un peu plus compliquée.

3.5.1 La complexité de la gestion d'un système distribué

La gestion des systèmes distribués est particulièrement complexe pour les raisons suivantes :

- une application distribuée peut faire interagir des systèmes appartenant à des organisations différentes. Il n'existe aucune autorité centrale pour maintenir un tel système inter-organisationnel. De même, la gestion de l'application aussi bien que de la communication inhérente à celle-ci peut devoir agir au travers de barrières organisationnelles et législatives.
- les systèmes qui doivent être gérés peuvent avoir des composantes[11] utilisant des technologies diverses. Ces composantes fournissent différents services qui peuvent interagir mutuellement et qui doivent dès lors être gérés.
- l'hétérogénéité des systèmes distribués. En effet, les composantes et les services proviennent de vendeurs différents possédant chacun leur interface de gestion. Cet inconvénient peut fortement restreindre une approche intégrée d'une solution de gestion.
  L'hétérogénéité des systèmes est résolue par l'utilisation d'une architecture de type manager-agent qui est elle-même standardisée. L'architecture proposée par le protocole SNMP[12] en est un exemple.
- les composantes et les services gérés sont physiquement distribués et agissent de manière parallèle. Cette double constatation rend difficile l'obtention d'un état global de consistance du système sur lequel des décisions de gestion peuvent être prises. Le problème de la distribution des ressources est résolu en utilisant l'architecture du *manager-agent* que nous avons vu au point 2.4.2.
- l'étendue et la complexité des grands systèmes distribués sont responsables de problèmes d'échelle de par le nombre de composantes qui doivent être gérées. Ceci rend impossible le traitement individuel de chaque composante à des fins de gestion. C'est pourquoi, on tente de regrouper les composantes en unités.

---

[11] Nous définirons ici les composantes d'un système distribué comme synonymes des ressources que nous avons décrites dans le point 3.4 : Un exemple de système distribué

[12] Ce protocole fera l'objet d'une étude un peu plus approfondie dans la suite de ce mémoire. Voir 3.6 Le protocole SNMP

On peut également résoudre ce problème en utilisant un agent que nous appellerons volontairement *agent-double.* Ce terme d'agent double se réfère à la capacité pour un agent d'agir aussi bien comme un agent que comme un manager.

Face à une telle complexité, nous avons besoin de fournir une vue intégrée des différentes tâches de gestion ainsi qu'un maximum de supports automatisés pour l'administrateur humain.

3.5.2 La nécessité d'intégration

Derrière leurs belles parures graphiques, les gestionnaires systèmes sont incapables de coopérer. Le résultat en est simple : codes et données affectés à l'exploitation s'ignorent jusqu'à devenir redondants. Un répertoire de données de gestion commun pourrait s'imposer pour mettre un terme à cette anarchie coûteuse. La prolifération des modèles de représentation des données orientés objets confirme la tendance actuelle. L'un de ces modèles est CORBA[13] (Common Object Request Broker Architecture).

Ces grands mouvements autour de l'intégration[14] révèlent qu'elle est urgente. Aujourd'hui, l'exploitation d'un système distribué passe par une multitude d'outils qui s'ignorent derrière leurs icônes. Gestion des fautes du système, mesure des performances et automatisation des configurations du système fonctionnent dans une indifférence réciproque, sans parler des applications comptables et de sécurité souvent occultées parce que moins importantes. Généralement, la coopération entre gestionnaires du système se cantonne à un simple appel d'exécutable sans aucun partage de données. Ce manque d'interaction se constate plus visiblement entre applications systèmes tierces, issues de fournisseurs différents.

Vient ensuite, comme [DEGHOR,92] nous le définit, une administration intégrée. Des systèmes de gestion intégrée sont proposés par différents constructeurs depuis quelques années. De tels systèmes permettent de répondre à la prolifération des applications de

---

[13] Corba représente des spécifications d'architecture informatique, mises au point par l'OMG (Object Management Group), portant notamment sur les mécanismes d'échanges entre objets distribués. Le lecteur intéressé peut trouver de plus amples renseignements sur Internet à l'adresse suivante : http://www.omg.org

[14] A ce titre, nous verrons un exemple concret d'intégration de système distribué dans la troisième partie de ce mémoire.

gestion. Un système de gestion intégrée combine, par intégration dans un même environnement, les différentes fonctionnalités de gestion pouvant provenir de divers fournisseurs. Les utilisateurs peuvent visualiser l'état du système géré à partir d'une interface utilisateur commune à toutes ces fonctionnalités.

Outre le gain en volume, une structure de données unique offre un modèle de représentation de données commun aux applications de gestion et offre également une économie d'énergie appréciable pour les développeurs.

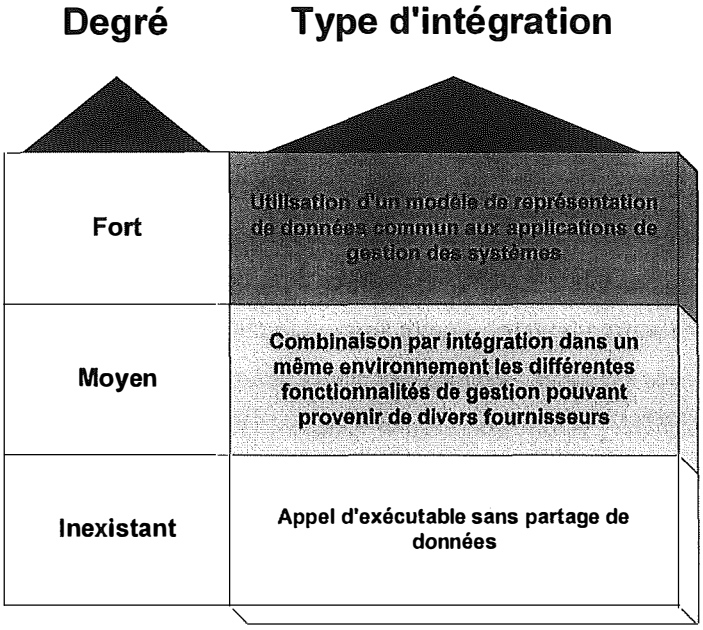

Figure 12 : Une vue des différents niveaux d'intégration

La solution adoptée, il y a 10 ans, était d'imposer une uniformité au niveau central. Or, ce n'est plus là une option acceptable pour les utilisateurs actuels. La solution prise aujourd'hui est de reconnaître et d'accepter la diversité, en offrant à l'administrateur une vue intégrée et complète du système. Interviennent ici des logiciels tels qu'ISM/OpenMaster de Bull ou encore HP OpenView de HP. Une structure de données unique se porterait au sommet de l'intégration mais dans une bataille aussi disputée que celle de l'administration, il sera certainement difficile de trouver un terrain d'entente.

Nous avons tenu à illustrer, sur la figure 12, une vue des différents niveaux d'intégration.

3.5.3 Les services de gestion d'un système distribué

Depuis la classification édictée par l'OSI, dans [ISO7498-4], la communauté informatique a coutume de découper la gestion des systèmes en cinq aires fonctionnelles[15].

Cette découpe, tout comme nous le fait remarquer [LANGSF,93], est utile pour structurer le problème de gestion mais il possède cependant deux inconvénients :

- il n'est pas idéal pour aborder techniquement les besoins de gestion et
- il omet de souligner que certains aspects des fonctionnalités de gestion peuvent s'appliquer à plusieurs aires fonctionnelles.

Finalement, la qualité de la gestion est jugée sur la qualité de service[16] qu'elle est capable de fournir.

Dans la gestion des systèmes distribués, selon [LANGSF,93], il existe quatre services clés. En ce qui nous concerne, nous ne retiendrons que les trois premiers, le dernier service évoqué par l'auteur peut facilement cohabiter avec le premier. Voici donc quels sont les services de gestion d'un système distribué :

1°) premièrement, la gestion des systèmes distribués doit fournir un service de rassemblement, de traitement et de maintien de données statistiques qui sont vitales pour les administrateurs. Ce service fournit les bases sur lesquelles des applications de comptabilité, de gestion des fautes et des performances peuvent être construites. Par exemple, ce service peut permettre de retrouver dans l'historique des données des cas similaires ou assimilés pour faire face à un diagnostic de panne. Ce service peut également offrir des analyses intelligentes, toujours basées sur les mêmes données, permettant de prouver le bon fonctionnement de l'environnement distribué (gestion des performances).

---

[15] Voir la section 2.2.2 : Aires fonctionnelles de gestion pour plus de détails

[16] Nous rejoignons à ce titre [DEGHOR,92] qui nous dit qu'une qualité de service *« se réalise en assurant une certaine disponibilité des composantes du système, en gardant un temps de réponse raisonnable, et ce, quels que soient les changements apportés au système. »*

2°) vient ensuite dans la gestion des systèmes distribués, la possibilité d'autoriser ou de refuser l'accès aux ressources et le contrôle de ces mécanismes d'accès. De tels services peuvent être rassemblés sous le vocable de gestion de la sécurité.

3°) troisièmement, la gestion des systèmes distribués doit offrir un service pour contrôler l'installation, le fonctionnement et la maintenance des systèmes. Il doit offrir les services de mise en service et/ou d'arrêt des ressources. Ces services peuvent être repris sous la dénomination de gestion de la configuration.

3.5.3.1 La gestion des fautes

La gestion des fautes est le nom donné à la gestion des problèmes qui peuvent survenir dans un système. Son objectif est de maintenir une disponibilité maximale du système. Une faute peut aussi bien survenir sur un composant matériel que logiciel. Si aucune action de rétablissement n'est prise, la faute peut engendrer la panne du système ou son dysfonctionnement.

Lorsqu'une faute est détectée, la première action à entreprendre est de tenter de **restaurer** le service. Autrement dit, on cherche à s'assurer une qualité de service en s'assurant de la poursuite de l'état de bon fonctionnement du système. En agissant de la sorte, le système a toujours la possibilité de continuer la tâche qu'il devait réaliser même si les performances sont réduites. La localisation de la faute et de sa cause sont alors **diagnostiquées** grâce à des analyses. Finalement, une action de ***correction*** peut être effectuée afin de résoudre la faute. Cette séquence d'actions que nous venons d'énoncer est vitale pour s'assurer un niveau de service satisfaisant.

a) La détection des fautes

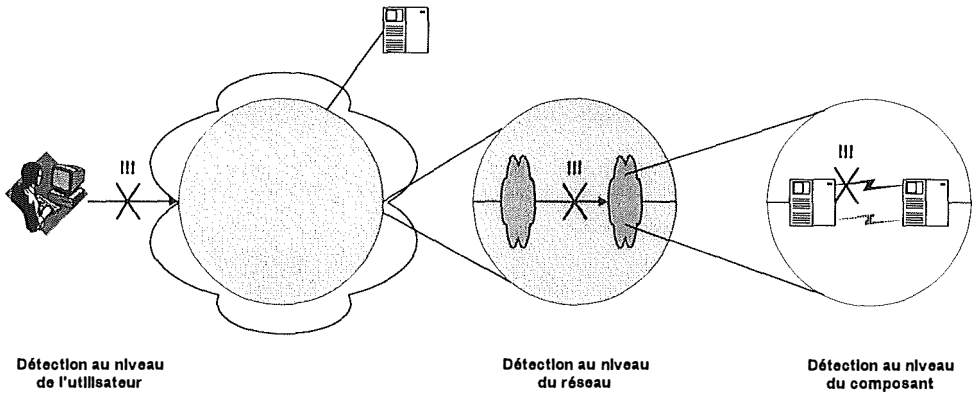

Figure 13 : La détection des fautes

Les fautes peuvent être auto-détectées, détectées par d'autres composants du système, par des fonctionnalités de surveillance ou encore par des utilisateurs. La figure 13 montre le processus suivi pour détecter une faute. Pour cela, nous supposerons que la faute survient dans une connexion entre deux ordinateurs.

Si ces deux ordinateurs peuvent détecter la faute, elle sera localisée au niveau du composant. La même faute peut se manifester au niveau du réseau si la panne a pour conséquence la perte de communication entre les composants. Elle peut également se manifester au niveau de l'utilisateur qui se rend compte d'une perte de service. Malgré cette constatation, l'utilisateur ne sait ni où se trouve la faute ni sa nature.

b) La restauration du service

Lorsqu'une condition de faute jugée sévère ayant pour résultat une perte significative de service a été identifiée, elle doit être isolée. Le service doit ensuite être rétabli.

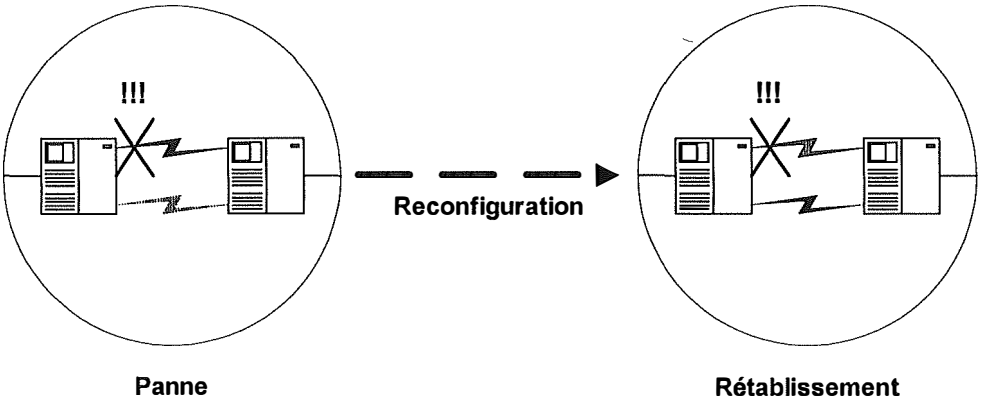

Figure 14 : Rétablissement d'une faute

La figure 14 montre un rétablissement du service de communication au moyen d'un processus de reconfiguration dans lequel une possibilité de connexion alternative est envisageable. Ce rétablissement peut être implicite[17] si une seconde connexion est disponible et si les composants ont également la possibilité de se synchroniser.

[17] Nous gardons ici le même sens du mot donné au point 2.4.1 : La gestion explicite et implicite, à savoir automatisé.

c) Le diagnostic de faute

Après avoir localisé la faute et après avoir répondu aux objectifs primaires de gestion en fournissant une continuité de la tâche, l'étape suivante est de diagnostiquer la nature de la faute.

d) La résolution de la faute

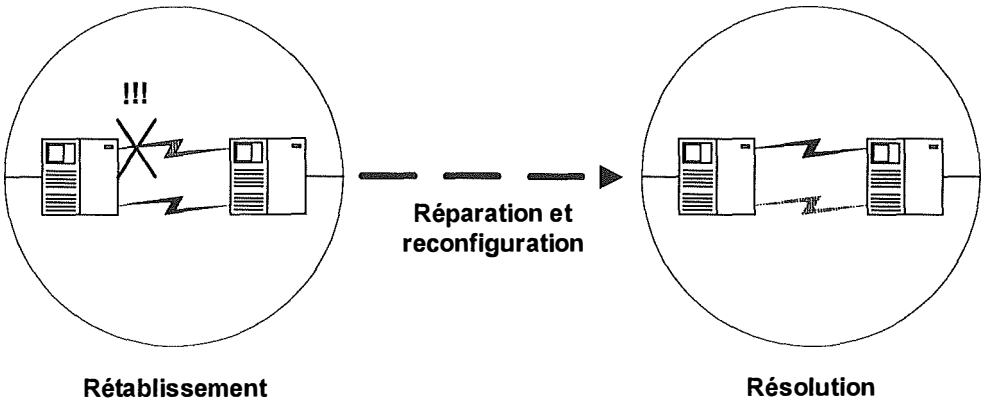

Figure 15 : Résolution d'une faute

Lorsqu'il s'agit d'une panne située sur un composant matériel, la résolution de la faute prend souvent la forme d'une réparation ou d'un remplacement de ce matériel défectueux. Il est évident que ce type de résolution nécessite l'intervention d'une personne humaine.

Par contre, lorsqu'il s'agit d'un problème purement logiciel, il se pourrait qu'il en résulte une réinitialisation du système.

Sur la partie gauche de la figure 15 se trouve illustré un rétablissement dans lequel une connexion provisoire a été choisie après avoir détecté la panne. L'une des solutions pour résoudre le problème initial de la perte de service pourra se faire en deux étapes qui sont :

- la réparation de la connexion défectueuse et
- la réinitialisation de la connexion.

3.5.3.2 La gestion des performances

La gestion des performances est d'intérêt pour :

- établir et planifier des objectifs de performances du système distribué,

- prédire les performances futures du système grâce à des méthodes de modélisation,
- mesurer et comparer les performances réelles avec les objectifs et
- optimiser les performances des différents sous-systèmes composant le système global.

Le chemin suivi dans cette section est de considérer en premier lieu les mesures de performances, d'analyser ensuite la manière par laquelle les performances peuvent être modélisées pour terminer par l'optimisation des performances d'un système distribué.

a) Les mesures de performances

Le but des mesures de performances est d'offrir au gestionnaire de performances la possibilité de déterminer si les objectifs de performance de l'entreprise sont atteints ainsi que d'identifier des possibilités de diminuer les coûts engendrés tout en respectant ces objectifs. Deux types d'information sont nécessaires. Le premier est le niveau de performance réel du système sous une charge connue. Cette information permet à l'administrateur de savoir si le système rencontre ses objectifs de performance. Le second type d'information est l'estimation des performances sous une charge potentielle.

Afin d'obtenir des mesures de performances, des métriques doivent être identifiées et définies.

Définition :

Les métriques représentent toutes les mesures faites sur un système en activité et qui sont utilisées, entre autres par les gestionnaires de performances.

Les problèmes que l'on rencontre lorsque l'on désire obtenir des mesures de performances se trouvent restreints par le manque de rapidité de communication dans un système distribué : il faut en effet envoyer un message à travers le système et attendre une réponse, ce temps de délai étant sensiblement augmenté par la gestion du protocole de communication. On se trouve alors devant un paradoxe : le meilleur moyen d'obtenir les informations nécessaires pour actualiser les métriques nécessite l'envoi de nombreux messages ralentissant ainsi la rapidité du système. De plus, si les entreprises ont opté pour la mise en place d'un modèle de gestion centralisée, les nombreuses applications traitant les métriques sur une seule machine entraînent des problèmes non négligeables de performances.

Dès lors, il peut coûter plus cher à l'entreprise de dériver des mesures de performances que le bénéfice que celles-ci pourraient leur apporter. Ceci est un facteur dont il est nécessaire de tenir compte et qui influencera inévitablement les investissements dans des services de mesures de performances.

b) La modélisation des performances

Dans la majorité des cas, lorsqu'un problème de performance survient, la première réaction est de faire évoluer le système. Même si cette réaction s'avère être une bonne stratégie pour un ordinateur personnel, ce n'est pas nécessairement le bon choix pour un plus gros système. Pour un système distribué, une évolution est sans nul doute inapproprié. En effet, les facteurs influençant les performances d'un système distribué sont plus complexes, et une simple mise à jour d'un seul composant peut engendrer un effet non désiré. Une telle tentative aura probablement comme conséquence de déplacer le problème sur une autre partie du système. C'est justement dans de telles situations que la possibilité de modéliser les performances du système est d'une grande importance.

Les buts d'une modélisation sont triples :

- obtenir une meilleure vue du fonctionnement du système,
- évaluer le comportement probable du système et
- tester des hypothèses sur des conceptions de systèmes alternatifs.

Ce que l'on retrouve dans un modèle de performances n'est composé ni plus ni moins que des caractéristiques pertinentes du système que l'on cherche à modéliser. Après avoir été défini, le modèle doit être initialisé et doit ensuite être soumis à différents facteurs choisis au préalable pour pouvoir suivre le comportement du système lorsqu'il est soumis à ceux-ci. Ces facteurs peuvent très bien être des changements de comportement du système distribué, une surcharge ou une reconfiguration d'un sous-système particulier.

Toute modélisation de performances nous montre aussi qu'il faut tenir compte de trois choses. La première est la validation du modèle, la seconde, le choix du niveau de détails et la dernière réside dans la compréhension et l'interprétation des résultats. La validation du modèle est assez subjective, car elle est basée sur l'expérience de son concepteur. Des facteurs tels que la taille du système distribué et la diversité de ses ressources influencent considérablement le niveau de détail que l'on va choisir et qui sera le plus adéquat pour le modèle. Si ce niveau n'est pas assez profond, certaines caractéristiques essentielles du

système risquent d'être omises, par contre si on choisit un niveau de détail élevé, il en coûtera cher lors de la validation du modèle.

Avant d'opter pour une modélisation, il s'agit donc pour les gestionnaires du système distribué de savoir si le bénéfice qu'ils en retireront sera primordial. Nous rejoignons donc la section précédente dans laquelle nous disions que cela peut coûter plus cher à l'entreprise de dériver des mesures de performances en termes d'investissements par rapport au bénéfice que celles-ci pourraient leur apporter.

c) L'optimisation des performances

L'optimisation des performances est un domaine complexe. Elle recouvre bien plus que la simple optimisation de chaque composante prise séparément.

L'optimisation comprend la surveillance du système distribué ainsi que celle de chacun de ses sous-systèmes, l'identification de ses faiblesses et leurs corrections. Des cas typiques de faiblesses sont illustrés par la congestion[18], les retards dans le transfert des informations de gestion,... [LANGSF,93] nous propose trois domaines, appartenant aux systèmes distribués, dans lesquels l'optimisation des performances est de mise. Il s'agit de la communication entre les sous-systèmes, de tout ce qu'il convient de mettre sous le vocable de traitement ainsi que de la disponibilité du système dans son entièreté.

La communication est une partie critique de tout système distribué. Ses objectifs d'optimisation sont généralement doubles : réduire les retards et accroître le débit des moyens de transmission. Toujours selon [LANGSF,93], ces objectifs peuvent être atteints en ajustant des paramètres tels que *"le nombre de buffers de chaque noeud ou les algorithmes de routage*" ou en choisissant un nouveau protocole de communication. Dans les situations où les valeurs optimales ont déjà été établies, une optimisation des performances peut se faire par un redimensionnement du système, par exemple en ajoutant de nouvelles ressources ou en utilisant une nouvelle technologie.

Les performances de traitement sont quant à elles affectées par l'allocation des parties logicielles et peuvent être optimisées par une meilleure distribution des ressources pour ces mêmes parties logicielles. Si la configuration est statique, l'optimisation doit être faite lors de la configuration. Dans le cas où les performances peuvent être optimisées dynamiquement en migrant les composantes logicielles, la surveillance des performances

[18] Ce problème a été soulevé au point 2.3.3 dans lequel nous avons illustré le problème de la congestion.

identifiera les composantes de l'environnement distribué qui sont peu utilisées ainsi que celles qui peuvent être saturées. Les composantes logicielles peuvent alors être redistribuées au travers des ressources peu utilisées.

La disponibilité du système peut être améliorée en augmentant la fiabilité de chaque composante.

L'optimisation des performances est basée sur les deux points précédents, à savoir les métriques et la modélisation. On a besoin des métriques pour obtenir des informations et la modélisation pour prendre des décisions.

3.5.3.3 La gestion de la configuration

Rendre un système distribué opérationnel, c'est tout d'abord le configurer. Il s'agit d'obtenir l'image de ce système, en considérant chacun de ses composants comme un objet élémentaire. Chaque objet peut être caractérisé par un type, des attributs permettant de connaître l'état de l'objet et des relations. Tous les objets sont regroupés dans une base de données qui permettra à la gestion de la configuration d'établir une carte du système distribué ou de connaître l'état de chaque objet (est-ce que l'objet est opérationnel, est-il en réparation, ... ?).

La configuration du système distribué permet ensuite sa génération. Le système prend alors connaissance de son architecture, de la localisation de chaque objet ainsi que des moyens pour y accéder.

Les phases de configuration et de génération constituent l'initialisation du système après lesquelles il devient opérationnel. Un système peut voir son architecture évoluer au cours du temps (ajout, suppression, modification) nécessitant une nouvelle configuration/génération. Dans de petits systèmes distribués, cette phase peut se faire de manière statique. Ce qui signifie que l'on peut éteindre le service système et effectuer les changements. Cependant, cette méthode ne peut se faire s'il s'agit de gros systèmes distribués où la phase doit se faire dynamiquement.

Ne sachant être exhaustif, nous préférons regrouper les fonctions de gestion de la configuration selon les trois caractéristiques suivantes :

- les fonctions ***administratives***,
- les fonctions de ***relations*** et

- les fonctions de *contrôle*.

Les **fonctions administratives** sont celles qui identifient les composants en leur assignant des services. Elles déterminent les composants matériels et logiciels d'un système distribué et décident d'attribuer au mieux les services sur ces composants.

Les **fonctions de relations** établissent, maintiennent et identifient les relations existantes entre les composants. En rendant un tel service, ces fonctions permettent à deux composants d'interagir par le biais de leur interface respective.

Les **fonctions de contrôle** établissent, surveillent et identifient l'état et le statut des composants.

3.5.3.4 La gestion des informations comptables

La gestion des informations comptables est un service de gestion relatif au comptage d'informations sur l'utilisation des ressources et des services par les utilisateurs afin généralement de permettre une facturation en fonction d'une tarification. Il est important de faire la distinction entre l'utilisation des ressources et l'utilisation des services. Les ressources, telles que l'espace disque disponible pour le stockage de données ou la capacité du moyen de transmission (fibre optique, câble coaxial,...), sont utilisées par des services, tels que le stockage de fichier ou le courrier électronique. Ce qui est important de constater, c'est que les services apportent une valeur ajoutée aux ressources. Aussi bien les utilisateurs que les administrateurs ont besoin des informations comptables. En effet, les utilisateurs interagissent avec le système distribué via les services qui leur sont offerts et sont dès lors intéressés par l'utilisation des services. Quant aux administrateurs, ils sont à la fois intéressés par l'utilisation des services ainsi que par l'utilisation des ressources.

Nous pouvons établir les deux principaux objectifs d'un tel service de gestion :

- compter et facturer chaque utilisation d'une ressource ou d'un service et
- maintenir et établir des informations comptables sur plusieurs utilisations d'une ressource ou d'un service.

A ces deux objectifs peut venir s'ajouter un service supplémentaire basé sur le contrôle et l'accès aux ressources ainsi qu'aux services au moyen d'un quota.

La facturation de l'utilisation des services peut facilement se faire en mesurant les ressources qui ont été utilisées pour satisfaire au service. Il se peut malgré tout que certaines métriques ne prennent pas en compte toutes les ressources utilisées. Nous pouvons illustrer ce dernier point par la facturation d'une utilisation d'un service d'impression qui prendrait en compte le nombre de pages imprimées tout en ignorant les données matérielles (temps d'utilisation du processeur pour imprimer ces pages).

3.5.3.5 La gestion de la sécurité

La gestion des services de sécurité doit permettre le contrôle d'accès, l'authentification des correspondants, la confidentialité, l'intégrité et la non répudiation des données.

La sécurité d'un système distribué revient à disposer d'outils qui, selon [ARPEGE,92], assurent :

- *"à l'émetteur d'un message : que ce dernier parvient bien au bon destinataire et qu'il ne pourra être compris que par celui-ci : que le destinataire ne pourra nier avoir reçu le message et prétendre avoir reçu un message non expédié :*
- *au destinataire d'un message : l'authentification de l'émetteur, l'intégrité du message : que l'émetteur ne peut nier avoir envoyé le message et que seuls les émetteurs autorisés pourront lui parler."*

3.5.4 Les utilisateurs d'un système distribué

Les utilisateurs des systèmes distribués ont déjà été mentionnés à la section 2.3.1 de ce mémoire. Nous citerons maintenant les différentes catégories de gestionnaires responsables, à juste titre d'ailleurs, de la gestion de systèmes distribués :

- **les superviseurs**, qui évaluent les composantes du système distribué. Ils sont entre autre responsables du niveau de performance du système, responsables d'évaluer les alarmes,... Un outil de gestion de système distribué qui pourra répondre à de telles exigences offrira un contrôle plus sécurisé et plus économique ainsi qu'une amélioration de la qualité de fonctionnement du système.

- **les administrateurs**, qui définissent et configurent l'exploitation du système. Ils assurent également un service global aux utilisateurs.
- **les gestionnaires**, qui mesurent la qualité de service du système.

3.6 Le protocole SNMP

3.6.1 Introduction

Il nous semblait utile de parler du protocole SNMP car, comme nous l'avons vu dans la section 3.5.1, l'architecture standardisée d'un tel protocole permet de faire face à l'hétérogénéité des systèmes distribués.

Cette section va se focaliser et analyser l'approche de la gestion qui est standardisée par l'Internet Engineering Task Force (IETF). Cette approche est connue sous le nom de Simple Network Management Protocol ou encore l'approche de gestion TCP/IP.

Dans la seconde moitié des années 80, Internet a atteint une taille telle que sa gestion ne pouvait plus longtemps être soutenue par une base ad hoc : une approche structurée et standardisée de la gestion d'Internet était nécessaire [PRAS,95]. Ainsi, en 1987, trois propositions de gestion sont apparues dont deux seulement furent retenues : le protocole SNMP et le protocole Common Management Over TCP/IP (CMOT).

Une différence intéressante entre l'IETF et l'ISO réside dans le fait que l'IETF offre une approche plus pragmatique et dirigée vers le résultat que l'ISO. C'est pourquoi, seuls les protocoles et les Management Information Base (MIB) ont été standardisés. Ce que l'on peut écrire à propos des principes de la gestion d'Internet, ce sont les idées suivantes :

- tous les systèmes connectés au réseau doivent être administrables via SNMP,
- il doit être relativement facile d'étendre les capacités de gestion des systèmes existants (en étendant la MIB),
- la gestion du réseau doit être robuste. Même en cas de panne, certaines applications de gestion doivent rester disponibles.

La situation anarchique vers laquelle se dirigeaient certains systèmes a propulsé SNMP au rang du plus grand dénominateur commun de gestion. Ce protocole fait aujourd'hui l'unanimité au sein de la communauté informatique et les plus grands comme Bull, Digital, HP ou IBM renforcent sa pérennité. Plus aucun équipement système n'échappe à son emprise. A côté, la concurrence, grâce à CMIP, reste timide même si l'ISO, par le biais de CMOT tente de se raccrocher à l'emprise d'Internet. En réaction, L'IETF a comblé les défauts de jeunesse de SNMP en engendrant une nouvelle version, SNMPv.2, qui gagne en fonctionnalités sans sacrifier pour autant à la simplicité d'origine. Nous ne nous en occuperons cependant pas, car notre agent utilise le protocole SNMP.

3.6.2 Le protocole SNMP

Les documents définissant le protocole SNMP et la structure de la MIB sont les suivants :

RFC1155	Structure & Identification of Management Information for TCP/IP-based Internets
RFC1156	Management Information Base for Network Management of TCP/IP-based Internets
RFC1157	A simple Network Management Protocol (SNMP)
RFC1213	Management Information Base for Network Management of TCP/IP-based Internets : MIB-II

Le lecteur intéressé pourra consulter ces documents.

3.6.2.1 Les objectifs

Le protocole SNMP est relatif à la gestion des réseaux TCP-IP. Son intérêt essentiel réside dans sa très grande généralité et dans la possibilité qu'il offre d'être étendu, généralisé, au moyen de l'usage d'objets privés.

Aujourd'hui, SNMP est utilisé pour permettre l'administration de systèmes informatiques. Il est léger, peu riche et peu sûr, très simple à implémenter sur des petits systèmes mais il n'est pas adapté pour de gros systèmes.

Avec SNMP, il faut maîtriser les limites et ne pas vouloir lui faire gérer ce pour quoi il n'a pas été conçu. Il doit être considéré comme une transition vers CMIP.

3.6.2.2 Les concepts du protocole

Les idées sous-jacentes au protocole SNMP sont assez faciles à comprendre. Elles se retrouvent assez bien dans l'approche de gestion Orientée Objet adoptée par l'ISO. En effet, on peut retrouver des idées communes telles :

- le concept manager-agent,
- l'utilisation de fonctions de gestion pour l'échange d'information de gestion,
- l'utilisation des PDUs *GET* et *SET* pour effectuer les opérations sur les informations de gestion.,
- l'utilisation d'ASN.1[19] pour définir les informations de gestion et
- l'idée de la MIB.

La notation ASN.1 est la seule syntaxe de transfert imposée (norme ISO 8825) au protocole SNMP.

SNMP est fondé sur plusieurs idées simples :

- il existe des ***agents*** qui sont des composantes administratives (probablement logicielles) résidents dans les composantes administrables du système (couches IP, couche TCP, routeur, appareil de surveillance,...) :
- chaque agent maintient une base de données de gestion (appelée MIB, pour Management Information Base) constituée d'un ensemble d'objets traduisant des éléments administrables du réseau :
- il existe également des ***managers*** destinés à traiter des informations de gestion en provenance des agents et d'offrir des services de gestion à des opérateurs.

[19] ASN.1 (Abstract Syntax Notation One) est un langage formel développé et standardisé par l'ISO (norme ISO 8824) et le CCITT (recommandation X.208).

Le protocole SNMP est simplement le protocole de dialogue entre une manager et un agent. Il permet trois actions :

- l'interrogation par le manager du contenu de la MIB d'un agent :
- la modification par le manager du contenu de la MIB d'un agent :
- l'émission spontanée (aussi rare que possible), par l'agent, de messages d'information destinés au manager. Chaque fois que possible, il sera en effet préféré que le manager interroge l'agent.

Le protocole ne prévoit pas d'ordres sophistiqués comme par exemple les ordres de rechargement à distance (*reboot*). L'idée est en effet de dire que tout ordre doit pouvoir se ramener aisément à la modification d'un paramètre de la MIB.

Avec SNMP, un seul manager peut contrôler plusieurs agents. Le protocole SNMP est construit au dessus du User Datagram Protocol (UDP), qui est un protocole de transport sans connexion (Figure 16).

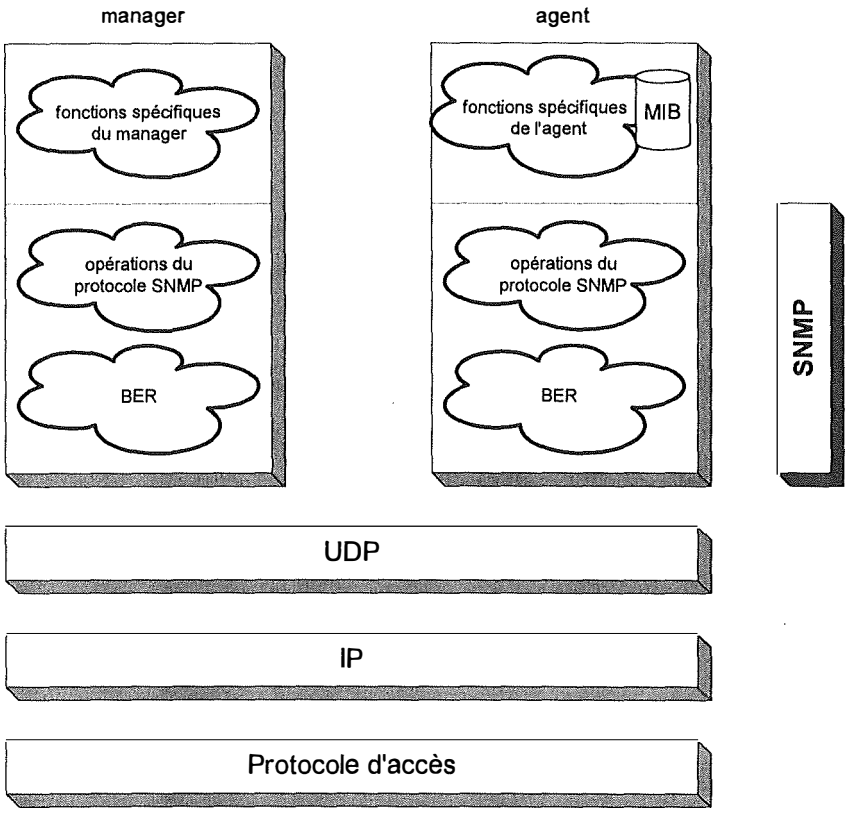

Figure 16 : Structure de gestion d'Internet

3.6.2.3 Avantages d'un mode non-connecté

Le choix d'utiliser le mode non-connecté UDP pour SNMP implique plusieurs avantages et inconvénients.

UDP n'est pas fiable, ce qui signifie que les informations de gestion peuvent se perdre. La décision d'utiliser ce mode de transfert a été prise de manière délibérée. La raison est que, même si certaines informations sont perdues, toutes les autres arriveront à destination. Par contre, en mode connecté, cela n'est pas possible. L'approche de ce dernier mode est "tout ou rien" : ce qui signifie soit que toutes les données seront livrées, soit qu'aucune ne le seront. C'est-à-dire qu'en mode non-connecté, même si une partie des informations arrive, il est possible que certaines applications puissent l'utiliser à des fins de traitements.

Nous pouvons cependant émettre une remarque concernant un tel mode de connexion. Lorsque l'on agit sur plusieurs ressources, il existe un risque d'incohérence si on ne peut obtenir toutes les informations atomiquement.

Il est intéressant de se rendre compte que le protocole SNMP ne s'occupe pas lui-même de la retransmission des données mais que celle-ci est l'œuvre du manager.

3.6.2.4 Les opérations supportées par SNMP

Sous SNMP, la communication entre le manager et l'agent est réalisée de manière confirmée. L'entité SNMP du côté du manager prend l'initiative d'envoyer l'un des PDU suivant : *GetRequest, GetNextRequest* ou *SetRequest*. Les deux opérations *GetRequest* et *GetNextRequest* sont utilisées pour obtenir les valeurs des informations de gestion situées dans la MIB de l'agent. Le *SetRequest* est utilisé pour initialiser ou modifier les valeurs informations de gestion. Après réception de l'un de ces PDU, l'entité SNMP du côté de l'agent répondra avec un *Response* PDU (Figure 17).

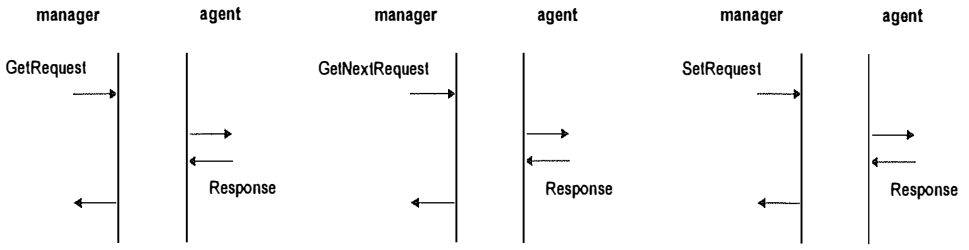

Figure 17 : Initiatives du manager

Il est également possible que l'agent prenne l'initiative d'envoyer un message vers son manager. Cette initiative arrive lorsque l'agent détecte un événement extraordinaire. Si tel est le cas, l'agent envoie un *Trap* PDU au manager (Figure 18).

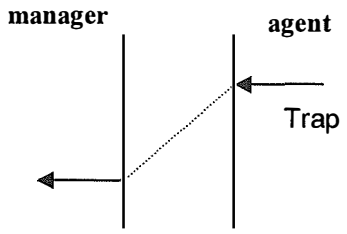

Figure 18 : Initiative de l'agent

Contrairement aux initiatives du manager, les *Traps* ne sont pas confirmées. Ce que le manager doit faire après réception d'un tel message n'est pas défini par le protocole SNMP mais il est de la responsabilité du manager d'agir en conséquence.

Comme CMIP, le protocole SNMP permet de consulter les valeurs des informations de gestion distantes (commandes GetRequest et GetNextRequest) et de modifier leur valeur (commande SetRequest).

Les réalisations ne sont toutefois pas complètes, certaines à cause du manque de sécurité du protocole. En guise de comparaison, le tableau de la page suivante essaie de mettre en exergue les différences majeures entre les deux protocoles mais nous tenons à faire remarquer qu'il est loin d'être exhaustif.

Les opérations d'*action*, de *création* et de *suppression* (action, create et delete) sous le protocole de gestion OSI (CMIP) ne se retrouvent pas directement sous les mêmes opérations SNMP. Cependant, les fonctionnalités de ces opérations peuvent être implémentées avec les deux opérations *get* et *set* de SNMP si la MIB a été proprement conçue et réalisée.

SNMP peut être considéré comme une version simplifiée de CMIP et vu comme un point de passage préliminaire vers CMIP. Par ses similitudes de fonctionnement avec le protocole CMIP, il autorise un apprentissage des concepts de gestion OSI tout en répondant à une catégorie de besoins urgents de gestion. Une importante faiblesse réside certainement dans l'inexistence de processus de contrôle d'accès aux variables gérées.

	CMIP	SNMP
1) empilement protocolaire	empilement OSI riche: • sur couche session en mode connecté • avec usage de la couche présentation OSI en mode connecté • avec usage de ACSE[20] • avec usage de ROSE[21]	empilement faible • UDP/IP • mode non-connecté, donc non fiable
2) PDUs	**GET** Permet de lire une instance avec possibilité de plusieurs réponses à une seule requête	**GetRequest** Requête de lecture sans réponses multiples
	CANCEL-GET Permet de provoquer l'annulation de la demande de lecture	☒
	CREATE Permet de créer une nouvelle instance (objet) d'une classe	☒
	DELETE Permet la destruction d'une ou plusieurs instances	☒
	SET Permet de modifier une instance	**SetRequest** Requête de modification d'information sur un attribut de la MIB
	☒	**GetNextRequest** Requête de déplacement dans la MIB
	ACTION Permet de définir une action à effectuer sur un ou plusieurs objets	☒
	EVENT-REPORT Permet la notification d'événements par un agent	**TRAP** Fourniture d'information spontanée par un agent

Tableau 1 : Différences fondamentales entre SNMP et CMIP

[20] ACSE (Association Control Service Element) est conçu pour gérer les connexions, qui s'appellent des *associations* au niveau application

[21] ROSE = Remote Operation Service Element

3.6.3 SNMP MIBs

Dans cette partie, nous allons expliquer comment pouvoir lire et utiliser une base d'information de gestion (MIB) définie pour le protocole SNMP. Avec une MIB particulièrement bien définie, SNMP peut être utilisé pour gérer la configuration, la performance, les anomalies, la comptabilité et la sécurité du réseau. Une MIB définit des objets gérés en utilisant une structure appelée structure d'information de gestion (SMI).

3.6.3.1 Qu'est-ce qu'une MIB ?

Le protocole SNMP est un outil de transfert et de manipulation d'informations de gestion.

Les concepts OSI et CCITT de gestion ont été largement repris par la communauté Internet pour développer SNMP. C'est pourquoi l'on retrouve également la notion de base d'information de gestion (ou MIB) qui regroupe conceptuellement les objets gérés. De même, comme dans l'environnement OSI, les objets gérés de la MIB SNMP sont spécifiés en ASN.1.

Une définition de la base d'information de gestion est donnée ci-dessous :

Définition :

Dans un système distribué, il y a un grand nombre d'agents (des agents SNMP en ce qui nous concerne). Chaque agent gère un ensemble d'objets réels. L'ensemble des objets gérés répartis dans le système distribué est regroupé dans la MIB.

3.6.3.2 La structure d'information de gestion (SMI)

a) Les Object Identifiers (OIDs)

La structure d'information de gestion est similaire au schéma d'un système de base de données. Il définit le modèle des objets gérés et les opérations qui peuvent être effectuées sur ces objets, aussi bien que les types de données qui sont autorisés pour ceux-ci. L'approche OSI est similaire au modèle orienté-objet tandis que l'approche SNMP est similaire au modèle relationnel d'une base de données [PERKIN,93].

Chaque objet dans un système distribué peut être accédé individuellement. Chaque objet a deux identifiants : un identifiant d'objet (*object identifier*) et un descripteur d'objet (*object descriptor*). Seul l'identifiant de l'objet est utilisé lors des échanges de protocole.

L'identifiant d'objet

Chaque objet SNMP est identifié de manière unique par un identifiant d'objet (OID). Un OID est formé soit par la concaténation des numéros assignés à chaque noeud de l'arbre d'enregistrement (*Registration Tree*) soit par la concaténation des noms. Ainsi par exemple, l'OID[22] de *generix* est **1.3.6.1.4.1.107.5000** pour la première forme et **iso.org.dod.internet.private.entreprises.Bull.generix** pour la seconde.

L'OID est utilisé lors des échanges de protocole.

Le descripteur d'objet

Le format d'un OID (une chaîne composée de nombres) n'est pas facilement administrable. Afin de rendre ces identifiants plus compréhensibles aux yeux d'une personne, chaque élément identifié par un OID l'est également par un nom - le descripteur d'objet. Ce nom prend la forme d'une chaîne de caractère qui est unique à travers tout le système distribué.

Des exemples de descripteur d'objets :

- generix (OID 1.3.6.1.4.1.107.5000)
- generixVersion (OID 1.3.6.1.4.1.107.5000.1)
- et ainsi de suite...

Le descripteur d'objet n'est pas utilisé lors des échanges de protocole. Ainsi, le descripteur d'objet doit être traduit en OID pour les échanges. Cette traduction sera faite par le SNMP Agent Toolkit[23] (SAT).

b) L'arbre d'enregistrement

L'arbre d'enregistrement regroupe de manière unique des objets de gestion, des compagnies, des organismes,...

---

[22] A des fins de compréhension, nous avons choisi de faire un saut vers la partie pratique afin d'illustrer les quelques concepts théoriques. L'OID, pris comme exemple, était en réalité l'OID choisi lors du développement de l'agent.

[23] Voir section 6.2. pour avoir plus d'informations sur le SNMP Agent Toolkit

La figure 19 illustre cet arbre ainsi que l'endroit où se trouve l'agent SNMP *generix* développé lors du stage.

L'enregistrement est utilisé pour identifier par exemple l'ISO, le NMForum, SNMP et les objets privés [SNMPAG,91]. Comme bon nombre de définitions d'objets sont ajoutées au sein de la communauté, on leur assigne un numéro unique d'identification. C'est le principe de l'OID dont nous venons de discuter.

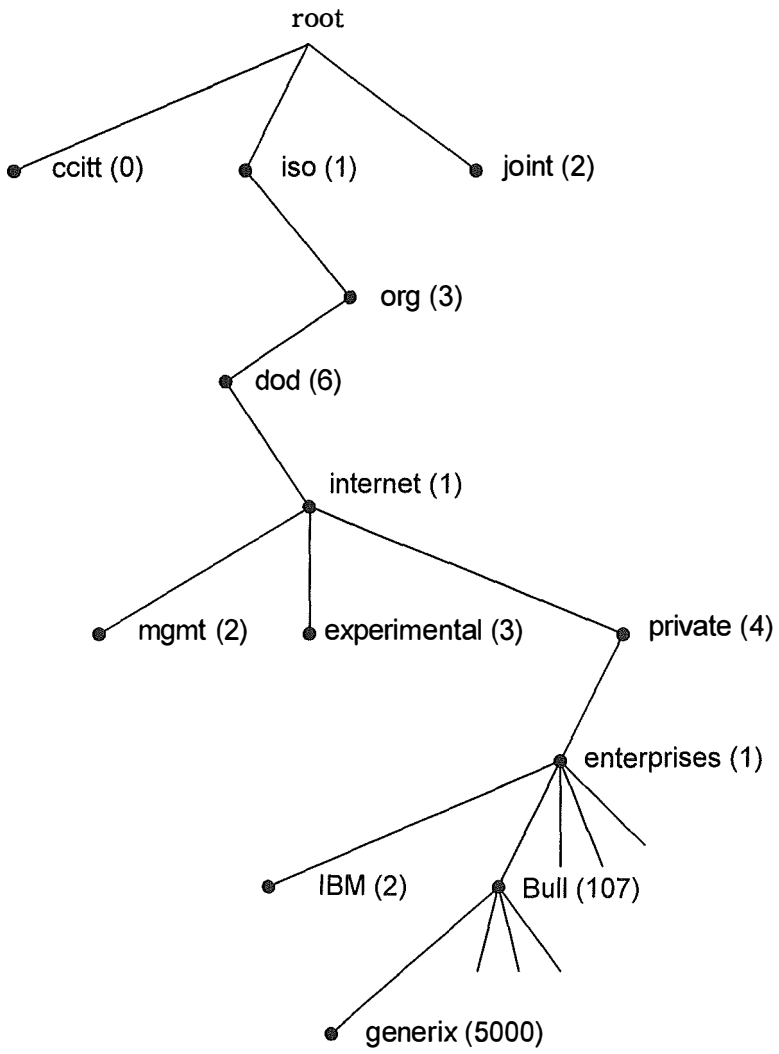

Figure 19 : L'arbre d'enregistrement

L'OID est composé d'une série d'entiers qui traversent un chemin depuis la racine (*root*) de l'arbre d'enregistrement jusqu'au noeud que l'on doit identifier. Chaque noeud de l'arbre possède sa propre "autorité d'enregistrement" qui détermine la manière d'allocation des numéros dans son sous-arbre.

Par exemple, le chemin 1.3.6.1.4.1.107 définit une branche de l'arbre d'enregistrement qui aboutit à l'entreprise Bull. Cette organisation est donc l'autorité qui décide du numéro identifiant à donner pour, dans notre cas, *generix*. Nous avons ainsi la certitude que cet agent possède un numéro identifiant unique.

3.7 Conclusion

Au regard du modèle de gestion d'un système distribué que nous avons présenté, on peut remarquer que l'interrogation (le polling) reste la méthode la plus communément utilisée pour récolter des informations de gestion.

Ce modèle de gestion, que nous venons d'illustrer au moyen du protocole SNMP, a pour conséquence une utilisation importante des moyens de transmission (fibre optique, câble coaxial,...) parce que les applications de gestion sont centralisées.

Depuis quelques temps déjà, on a pu assister à l'émergence de nombreuses plates-formes intégrées proposant d'autres modèles de gestion. Avant de proposer la solution de gestion que nous avons développée pendant notre stage chez Bull, nous nous devons de compléter cette partie théorique par la description des différents modèles de gestion.

4. La distribution de la gestion d'un système distribué

4.1 Introduction

Dans ce chapitre, nous allons examiner différentes manières actuellement utilisées ou développées pour gérer un système distribué. Nous montrerons qu'il existe de réels problèmes qui ne peuvent être correctement résolus en utilisant une approche de gestion totalement centralisée. Dans certains cas, une approche distribuée sera plus appropriée voire même nécessaire afin de répondre aux besoins de gestion. Nous tenterons ici de décrire une telle approche tout en dégageant les principales caractéristiques qui nous permettent de classer les applications sous le paradigme de la gestion centralisée ou distribuée.

La section suivante tentera de mettre en exergue les trois principaux modèles de gestion. La section 4.3 tentera d'élaborer une grille de distribution à l'intérieur de laquelle les applications peuvent s'installer. Nous terminerons ce chapitre par une conclusion.

4.2 Les modèles de gestion d'un système distribué

Nous commencerons cette section par définir les deux modèles extrêmes, à savoir le modèle de gestion centralisée et distribuée. Entre ceux-ci vient s'intercaler le troisième modèle : le modèle de gestion partiellement distribuée.

4.2.1 Le modèle de gestion centralisée

A la base, un système de gestion contient quatre types de composants : des managers, des agents, des protocoles de gestion et des informations de gestion. Un manager

communique avec un ou plusieurs agents au moyen d'un protocole de gestion. Les informations communiquées sont contenues dans une MIB[24].

La plupart des systèmes de gestion actuels ont adopté le paradigme d'une gestion centralisée fonctionnant comme nous l'avons signalé auparavant[25].

Même si un tel mode de gestion s'est très vite démocratisé chez les différents constructeurs d'applications de gestion, il n'en reste pas moins que l'on peut facilement lui émettre les quelques critiques et limitations suivantes :

- vu la simplicité des agents conçus actuellement, on ne pourrait envisager leur déléguer une tâche de gestion lorsque, par exemple, une connaissance globale du système leur est nécessaire.
- une gestion centralisée offrira un temps de réponse assez lent pour détecter une panne ou une alerte. Prenons le cas d'une imprimante reliée au système et qui tombe en panne. L'administrateur du système ne pourra constater la défectuosité de ce matériel que si un polling est effectué à intervalle régulier[26] ou également, si une notification survient de la part de l'agent responsable de la gestion de cette imprimante.
- une application de gestion centralisée se contentera de collecter des données brutes envoyées de l'agent vers le manager situé sur la plate-forme centrale. On imagine facilement que pour traiter des centaines voire des milliers de données brutes, il faut obligatoirement avoir une application de gestion très grosse, compliquée et gourmande en ressources.
- si une panne survient dans la communication entre un manager et son agent dispersé géographiquement, une application de gestion centralisée tentera d'accroître les accès aux informations alors que le système sera moins apte à les manipuler. La probabilité d'avoir une erreur de gestion sera dès lors plus grande puisque les informations de gestion sont plus difficiles d'accès voire même inaccessibles.

---

[24] Voir supra 3.6.3.1 :Qu'est-ce qu'une MIB ?

[25] Voir supra 2.4.2 : La gestion centralisée et distribuée

[26] Généralement, un polling est effectué à intervalle régulier si le développeur de l'application de gestion l'a conçu. Cependant, la fréquence d'interrogation des agents peut se faire suivant des intervalles de temps relativement long et par conséquent, trop tard pour détecter l'occurrence de la panne dans un laps de temps relativement court.

4.2.2 Le modèle de gestion distribuée

La gestion distribuée prend en compte deux choses :

1. la capacité de traitement des agents. Un agent était vu auparavant comme une entité assez simple et bien souvent très petite en code. Vu l'augmentation des capacités de traitement des ordinateurs ainsi que des différentes ressources, on peut très bien envisager développer des agents plus gros, possédant des fonctionnalités jusqu'alors réservées aux managers.
2. la réduction des échanges de données entre les différents agents et le manager.

Dans ce second modèle, les agents sont capables d'effectuer des fonctions de gestion que nous qualifierons de sophistiquées de par leur nature. Cette capacité offre l'avantage d'alléger les managers d'applications de gestion qu'ils étaient les seuls à exécuter. Il est naturel de constater une très nette diminution des échanges d'informations de gestion lorsque les applications de gestion se trouvent sur l'agent. En effet, les informations de gestion qui devaient être échangées entre l'agent et son manager seront maintenant gérées et traitées par l'agent.

L'étude menée lors de la réalisation de la solution *NETRIX* illustre à juste titre ce problème d'échange d'informations de gestion. Selon [WWWNET,96], le protocole SNMP est excellent pour gérer les composantes d'un réseau local (LAN[27]). La plupart de ces réseaux locaux offrent une capacité de transmission nécessaire pour qu'un seul manager puisse interroger plusieurs agents sans en affecter les performances du réseau. Or, la capacité de transmission utilisée par le protocole SNMP dépend directement de la fréquence des interrogations issues du manager. Cependant, le manager est confronté au dilemme suivant : une plus grande fréquence d'interrogation permet de détecter plus rapidement un problème mais nécessite également un moyen de transmission dont la capacité est grande, tandis que le choix de réduire les interrogations retarde la découverte d'un quelconque problème mais nécessite une capacité du moyen de transmission plus petite. Dans un réseau mondial (WAN), la capacité des moyens de transmission de données est plus faible que celle d'un réseau local. La solution recherchée immédiatement est de diminuer autant que possible l'échange d'informations de gestion. C'est pourquoi une solution de gestion distribuée serait tout à fait adéquate comme solution à ce problème.

[27] le lecteur intéressé pourra consulter [TANENB,93] pour de plus amples informations sur les réseaux.

Tout comme le souligne [WWWMIE,96], au plus haut niveau d'abstraction, *"the Decentralized Management By Delegation (MBD) paradigm and Centralized SNMP paradigm appear the same, as both have an Network Management Station (NMS) communicating with agents via protocol. But the MBD model supports a more distributed management environnement by increasing the management autonomy of agents."*. Ce qui est particulièrement intéressant avec une gestion par délégation[28], c'est le principe d'agent élastique. Au lieu de transmettre les informations de gestion de l'agent vers son manager afin que celles-ci soient traitées par les applications de gestion, les applications sont envoyées vers l'agent. La possibilité de déléguer des fonctions de gestion quand cela est jugé nécessaire permet de réduire l'utilisation du moyen de transmission. C'est dans ce sens que nous pouvons parler de distribution de gestion. Dans la suite de ce mémoire, lorsque nous parlerons d'agent *intelligent*, cela signifiera que nous nous situons dans ce modèle de gestion.

4.2.3 Le modèle de la gestion partiellement distribuée

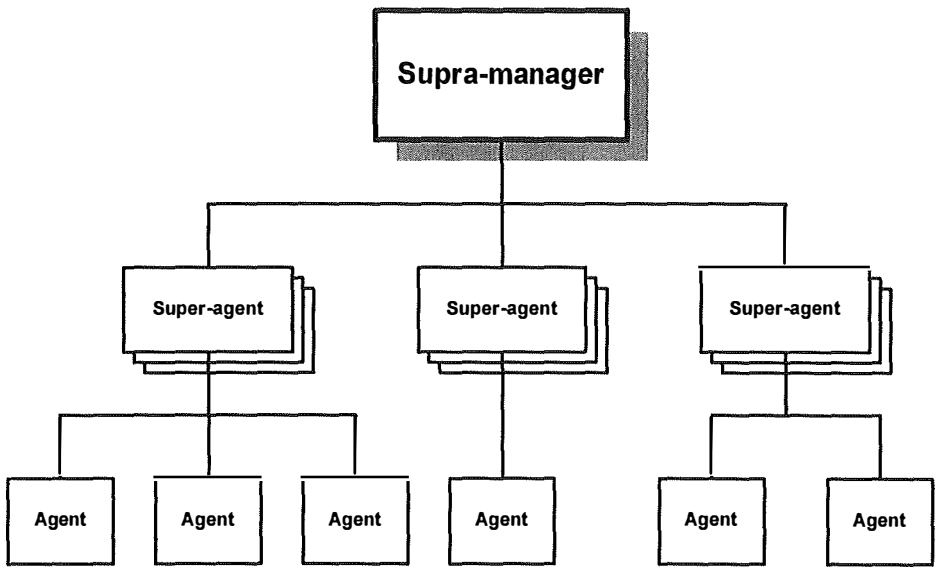

Figure 20 : Modèle de la distribution partielle

Un troisième modèle vient compléter les deux précédents. Nous définirons ce modèle comme étant une hiérarchie de gestion.

[28] la gestion par délégation est un cas concret de gestion distribuée tel que nous le présentons dans ce mémoire. Il existe de nombreux documents relatifs à ce sujet. Les plus intéressants sont selon nous [GOLYEM,95] et [GOLDSZ,94].

Le manager pourrait être considéré comme un *super-agent* qui aurait toujours la possibilité de communiquer avec les mêmes agents. Ce super-agent serait lui-même géré par un *supra-manager* unique. D'un autre côté, ce supra-manager peut gérer un ou plusieurs super-agents. Ceci signifierait donc que les différentes applications de gestion se retrouveraient échelonnées à plusieurs niveaux. Les agents exerceraient toujours la même fonction, à savoir la maintenance des informations de gestion brutes. Le super-agent collecterait les données des agents et les formateraient[29]. Ces données seront ensuite envoyées vers le supra-manager qui, à son tour, seront utilisées par les applications de gestion principales.

Ce paradigme de gestion est illustré à la figure 20.

4.3 Distribuer les applications de gestion

Les deux principaux paradigmes présentés dans la section précédente peuvent être vus comme des modèles tout à fait incompatibles.

En réalité, le modèle centralisé (SNMP par exemple) et le modèle distribué ne sont que deux points d'une échelle (voir figure 21) sur laquelle nous y ajoutons le modèle que nous avons qualifié d'intermédiaire.

Figure 21 : Hiérarchie des applications de gestion

La question que l'on est à même de se poser est la suivante : comment peut-on savoir si une application de gestion doit être distribuée ou pas ?

Pour y répondre, un certain nombre de métriques sont mises à notre disposition. Nous en avons relevé trois :

[29] Nous entendons par formater le fait d'analyser les informations et ensuite de pouvoir y appliquer diverses applications de gestion.

1°) **la fréquence d'interrogation** (*polling*).

On peut facilement imaginer une application de gestion qui surveille la valeur d'un objet critique existant sur un système distribué et qui lorsque cette valeur atteint un seuil prédéfini déclenche une alarme. Cette application aura peut-être besoin de connaître la valeur de l'objet en temps réel puisque ce dernier est qualifié de critique. Cela se fera donc avec une fréquence élevée.

2°) **la capacité du moyen de transmission** et

3°) **la quantité d'informations véhiculées.**

Ces deux dernières métriques peuvent être regroupées. On a déjà vu quels étaient les problèmes inhérents au protocole SNMP lorsque celui-ci était utilisé d'une part sur un réseau local et d'autre part sur un réseau mondial. Il est évident que si la quantité d'informations nécessaires pour gérer le système est faible, la capacité du moyen de transmission est de peu d'intérêt. Par contre, si on est confronté à une MIB composée de plusieurs centaines de tables, composés elles-mêmes de plusieurs centaines d'attributs, on n'hésitera pas à dire qu'un moyen de transmission possédant un grand débit est indispensable.

Sur la figure 22, nous avons tenté de représenter les métriques que nous venons d'expliquer dans la liste ci-dessus.

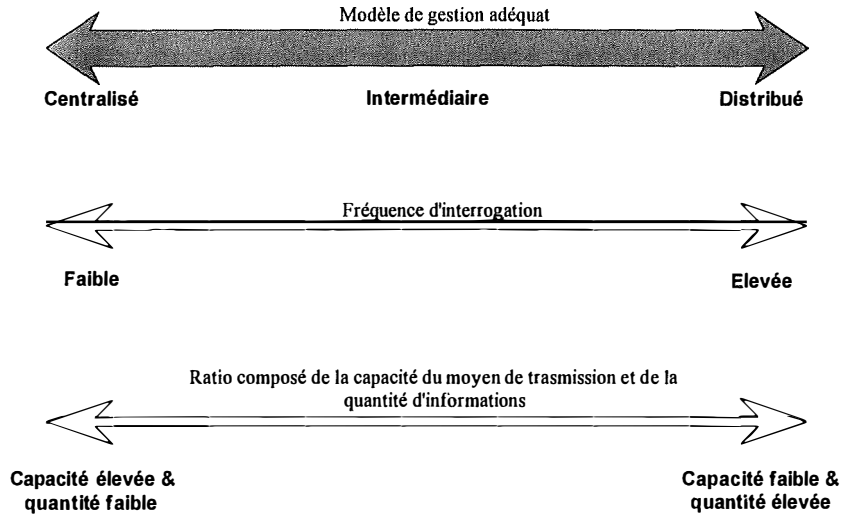

Figure 22 : Métriques utilisées pour déterminer le modèle de gestion adéquat

4.3.1 Les applications centralisées

A l'issue de ce que nous venons d'expliquer, nous pouvons constater qu'une centralisation de la gestion est appropriée pour des applications qui :

- disposent d'une fréquence d'interrogation faible,
- disposent d'une grande capacité de débit offerte par le moyen de transmission
- ont peu d'informations de gestion à leurs dispositions.

La plupart des applications de gestion actuellement utilisées possèdent ces caractéristiques. [GOLDSZ,95] souligne que certains pourraient arguer que c'est justement parce que le modèle centralisé (comme par exemple SNMP) est le seul présent dans la plupart des produits commerciaux. Après analyse, on constate que cette argumentation est fausse et que les trois caractéristiques font bel et bien partie intégrante des besoins des applications de gestion.

Exemple :

Pour ne citer qu'un exemple de ce modèle, nous pouvons prendre le cas d'une application existante sur un manager central qui se contente d'afficher les différentes valeurs des variables composant la MIB d'un agent, lui-même existant sur une autre machine du système.

4.3.2 Les applications partiellement distribuées

Exemple :

Un cas d'applications partiellement distribuées s'inscrit parfaitement bien dans la perspective d'une gestion de sous-systèmes autonomes, tels que des réseaux locaux, qui sont englobés au sein d'un système distribué. L'administrateur d'un département peut gérer son propre réseau et ne passer que les informations nécessaires au niveau supérieur.

La distribution partielle convient pour des applications qui exigent de grosses capacités des moyens de transmission mais qui requièrent un certain degré de contrôle central.

4.3.3 Les applications distribuées

L'analyse des métriques montre qu'une application distribuée est la plus appropriée pour des applications qui utilisent :

- des traitements sur de grandes quantités d'informations,
- des moyens de transmission dont le débit est plutôt faible et
- une fréquence d'interrogation élevée.

Exemple :

Un exemple de cette classe pourrait être illustré par le biais d'une application de surveillance sur des objets critiques d'un système distribué. La gestion d'un satellite (objet critique difficilement réparable dans des délais assez courts) illustrerait ce problème. En effet, si un satellite tombe en panne alors que des informations de gestion primordiales pour le manager central et collectées sur un sous-système sont véhiculées au travers de ce moyen, il faut obligatoirement trouver des moyens de transmission alternatifs qui n'entravent pas ou très peu l'application de gestion centrale. Dans un tel cas, il serait préférable de déléguer l'application de gestion vers un agent du sous-système.
Cet exemple prend en compte deux caractéristiques : une grande quantité d'information ainsi qu'une fréquence d'interrogation élevée.

4.4 Conclusion

Pour pouvoir fournir une gestion efficace, il faut quelque part centraliser les informations de gestion pour corréler les problèmes entre eux. Un système distribué est un tout. Son administration ne peut être faite sans agréger les opérations de gestion ainsi que les informations de gestion. On ne peut imaginer, à l'heure actuelle, l'optimisation des performances d'un réseau (partie intégrante d'un système distribué) en optimisant chacune des ressources sur ce réseau indépendamment des autres. D'où la nécessité de centralisation. Cette centralisation peut être mise en cascade comme dans le modèle de la gestion partiellement distribuée tel que nous l'avons présenté. Il nous semble que ce modèle est actuellement la meilleure solution pour gérer un système distribué. Il se peut cependant que l'on arrive un jour au modèle de gestion distribuée tel que nous l'avons présenté dans ce chapitre.

cependant que l'on arrive un jour au modèle de gestion distribuée tel que nous l'avons présenté dans ce chapitre.

Les bases théoriques étant installées, nous pouvons maintenant aborder la partie pratique de ce mémoire qui décrira l'agent *generix* après avoir étudié la plate-forme logicielle ISM/OpenMaster sur laquelle vient se greffer *generix*.

Partie pratique

OBJECTIFS

- illustrer les paradigmes des modèles de gestion d'un système distribué vus dans la partie théorique
- décrire la plate-forme logicielle ISM/OpenMaster
- montrer quels sont les modèles de gestion existants sous ISM/OpenMaster
- introduire et commenter l'agent *generix*, une solution de gestion distribuée

5. Etude de cas : ISM/Open-Master

5.1 Introduction

L'appellation de plate-forme désigne un environnement logiciel et matériel de support d'applications de gestion grâce à des interfaces définies. Les principaux constructeurs informatiques (IBM, DEC, HP, Bull,...) ont une offre de plate-forme.

L'objet de ce chapitre est de décrire, sans être exhaustif, ISM/OpenMaster. Cet outil de gestion correspond à la plate-forme proposée par Bull sur laquelle nous avons développé *generix.*

Nous[30] avons essayé de mettre en évidence les caractéristiques essentielles d'ISM/OpenMaster. Le lecteur intéressé par de plus amples informations pourra consulter la référence suivante : [GHLDD,94].

5.2 Présentation d'ISM/OpenMaster

ISM/OpenMaster, plate-forme de gestion intégrée est apparu suite à la complexification et à la diversité des applications de gestion. Contrairement à une multitude d'outils isolés, ISM/OpenMaster permet aux administrateurs d'avoir une vue homogène d'un système distribué. La figure 23 illustre le contexte dans lequel évolue ISM/OpenMaster.

ISM/OpenMaster est composé d'un ensemble d'outils de gestion de systèmes et de réseaux opérant dans un environnement multi-protocolaire et multi-domaine.

C'est-à-dire une plate-forme de gestion intégrée capable de supporter des protocoles de gestion tels que :

- des protocoles propriétaires AEP[31] (Bull DSA Administrative Exchange Protocol)

[30] Ce chapitre est le résultat de la partie commune avec le mémoire : [VANRYC,96].

- les standards de l'ISO CMIP
- les standards de fait SNMP
- des protocoles spécifiques SMT pour les Fiber Distributed Data Interface
  SNA pour IBM
  GMP[32],....

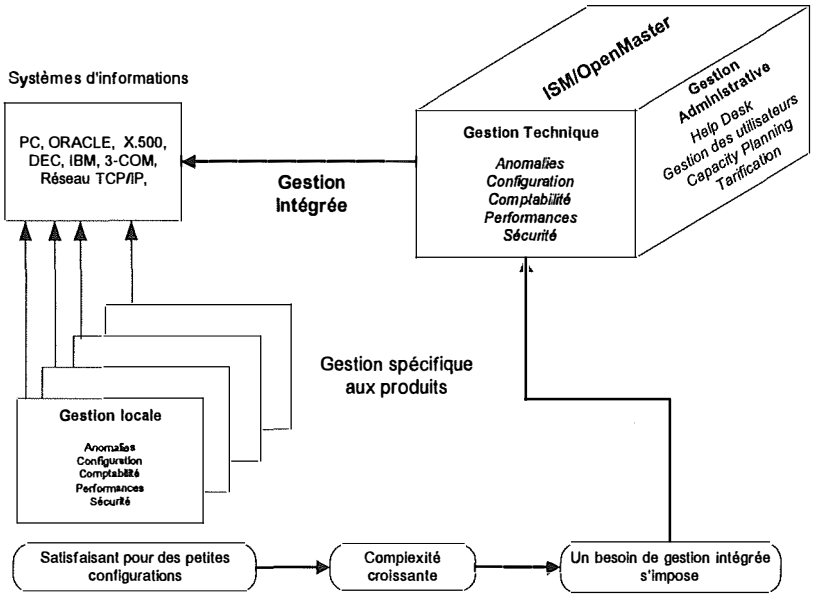

Figure 23 : Contexte d'ISM/OpenMaster

L'environnement multi-domaine permet de gérer les différents domaines suivants :

- ISM SQL Master pour les bases de données
- ISM TMN Master pour les télécommunications
- ISM Operation Master pour les systèmes
- ISM TransMaster pour les réseaux
- ISM Access Master pour la sécurité
- ISM PC Operation Master pour les groupes de travail

[31] Le protocole AEP est le protocole véhiculaire de base pour la communication des informations de gestion d'un réseau DSA. On distingue trois principaux types d'informations de gestion. Il s'agit des commandes, des réponses à ces commandes, et des messages non sollicités signalant des événements ou des erreurs se produisant sur un site.

[32] GMP GCOS Management Protocol

5.3 Architecture d'ISM/OpenMaster.

L'architecture d'ISM/OpenMaster est basée sur des standards qui ont été établis par l'ISO, le NM Forum (OMNIPoint), POSIX/IEEE, X/Open, l'IETF et l'ITU-T. L'un des concepts clé dans ISM/OpenMaster est celui de la MIB. Nous verrons cette partie plus en détails à la section suivante.

Dans un souci de compréhension, nous avons choisi d'illustrer nos propos au moyen de nombreux schémas explicatifs. Ces schémas, issus de la formation que nous avons reçue durant notre stage chez Bull, offrent une vision évolutive de l'architecture d'ISM/OpenMaster.

L'architecture globale d'ISM/OpenMaster repose en majeure partie sur le principe *Manager/Agent*. Il s'agit donc comme nous l'avons déjà vu dans les chapitres précédents d'une architecture à deux niveaux, d'une part l'agent et d'autre part de manager.
Le mécanisme *Manager/Agent* est basé sur un modèle orienté objet. Les objets réels d'un système distribué sont représentés par des objets abstraits formant eux-même la MIB. Les caractéristiques des objets peuvent être retrouvées grâce aux attributs des objets.

L'ISM/Manager perçoit le système distribué au travers des objets de la MIB et contrôle ce système via la manipulation des objets et de leurs attributs.

Les ISM/Agents vont permettre à l'ISM/Manager de visualiser les objets du monde réel en représentant l'état d'un objet réel par ses attributs dans la MIB. Comme illustré à la Figure A2- 1 de l'annexe 2, l'ISM/Manager envoie des commandes vers l'ISM/Agent, qui à son tour renverra des réponses et des notifications en utilisant des protocoles de gestion (SNMP,CMIP, DSAC/AEP, etc...).

5.3.1 ISM/OpenMaster Manager

5.3.1.1 Les applications de gestion

Les applications sont les composantes d'ISM/OpenMaster qui ont une interface homme-machine. Plusieurs copies d'une même application peuvent être actives. Pour chaque fenêtre ouverte par un utilisateur, il existe une copie active de l'application correspondante. Dans la plupart des cas, les applications sont développées dans le langage

SML (System Management Language), un langage de développement qui permet d'ajouter de nouvelles fonctionnalités à ISM/OpenMaster.

Ces applications de gestion interagissent avec les ressources gérées par les ISM/Agents au moyen d'une interface de gestion. Cette interface de gestion représente le cœur d'ISM/OpenMaster. Ces différents concepts sont illustrés à la figure A2-2 qui se trouve dans l'annexe 2 de ce mémoire.

Il nous semble inutile dans le cadre de ce mémoire d'énumérer toutes les applications existantes. Nous nous contenterons de présenter ISM/Monitor.

L'utilisation d'ISM/Monitor permet à l'administrateur de visualiser en temps réel le statut de toute composante existant dans le système. ISM/Monitor affiche un système au moyen d'une carte dans laquelle les différents objets gérés par l'agent peuvent être représentés et animés graphiquement.

5.3.1.2 Un modèle de gestion unique

Ce modèle de gestion contient les deux mécanismes principaux de l'ISM/Manager :

- le CMIS Dispatcher
- les Objects Managers qui se décomposent en services d'ISM/OpenMaster , en Intégrateurs d'agents (AI) et en Supra Management Interface.

Nous allons détailler chacun des mécanismes présents à la figure 24, sans être exhaustif en ce qui concerne les services.

Le *CMIS Dispatcher* permet la communication entre les applications ISM/OpenMaster et tous les Objects Managers. Tous les composantes de l'ISM/Manager vont communiquer entre elles grâce au CMIS Dispatcher.

Le CMIS Dispatcher, illustré par l'infrastructure de communication, comprend les fonctions suivantes :

- le **CMIS Application Programmatic Interfaces**, ces API fournissent une interface-CMIS donnant accès au *Internal Messaging Mechanism*

- l'**Internal Messaging Mechanism**, ce mécanisme est utilisé pour permettre la communication des applications, des services et des AI au sein de l'ISM/Manager.
- le **Command Router** qui oriente les requêtes et les réponses entre les applications et les Objects Managers.
- l'**Event Router** qui oriente les notifications entre des Objects Managers vers les applications
- Le **Root Object Manager (ROM)** qui est gestionnaire d'objet pour l'objet ROOT.

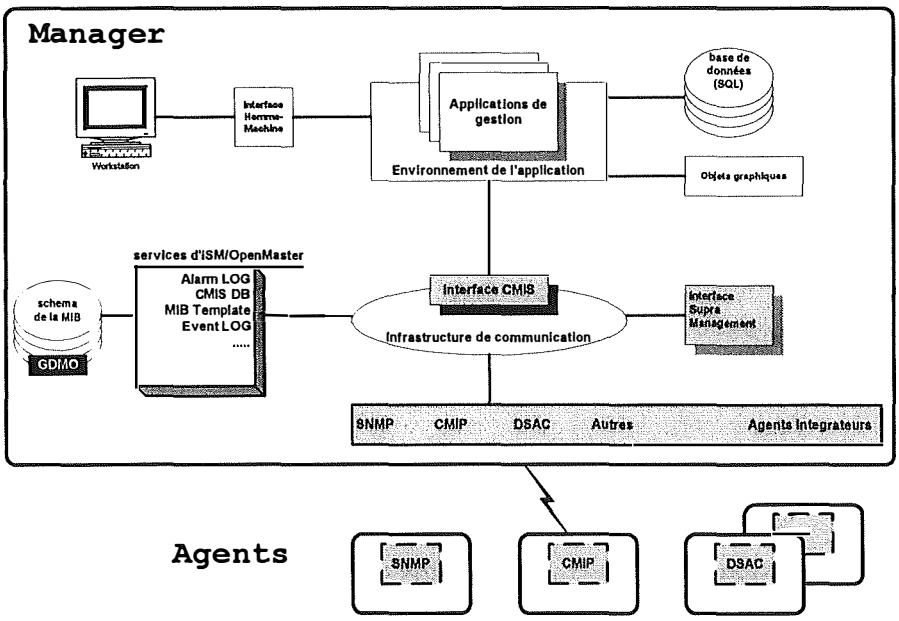

Figure 24 : Une vue unique du modèle de gestion d'ISM/OpenMaster

Les *services d'ISM/OpenMaster* sont les composantes d'ISM/Manager qui n'ont pas d'interface homme-machine et qui fournissent de fonctions indépendantes d'un protocole de gestion. Les services sont des Objects Managers ce qui signifie que leurs fonctionnalités sont accessibles via la MIB.

Les services disponibles au sein d'ISM/OpenMaster sont les suivants :

- **MIB Template Service** qui représente la base de données des classes d'objets gérés.

- **Alarm Log Service** qui permet de créer un journal d'alarmes
- **Event Log Services** qui permet de créer un journal de notifications d'événements
- **CMIS Database** est une base de données contenant les objets gérés accessible via CMIS.

Les *Supra Management Interfaces* permettent à un ISM/Manager de communiquer avec d'autres managers. Ces composantes permettent à un ISM/Manager d'agir en tant qu'agent contrôlé par un manager, appelé *supra-manager*. Ce concept se retrouve sous l'ISM/Concentrator. L'ISM/Concentrator filtre les informations de gestion qui devraient être envoyées des agents vers le manager central (qui, dans notre cas, n'est autre que le supra-manager), n'envoyant ainsi que ce qui est réellement nécessaire. L'ISM/Concentrator prend à sa charge l'interrogation des agents, la surveillance des paramètres et la notification d'alarmes vers le supra-manager lorsque cela est nécessaire. Ce que nous retiendrons des Supra Management Interfaces, c'est qu'elle permettent de hiérarchiser la centralisation des applications de gestion.

Les AI's communiquent avec les agents d'un sous-système de gestion spécifique en utilisant le protocole de gestion de l'agent. Ils communiquent avec les autres composantes d'ISM/OpenMaster à l'aide du service CMIS. La version actuelle d'ISM/OpenMaster comprend un AI par protocole de gestion supporté. Nous retrouvons ainsi un AI SNMP : un AI CMIP , un AI DSAC/AEP et un AI GMP qui est un protocole propriétaire de Bull. Chaque AI peut supporter plusieurs agents et agit comme un multiplexeur. Comme nous le verrons à la section 5.5, d'autres intégrateurs d'agents peuvent être ajoutés selon les besoins du client permettant ainsi d'utiliser des protocoles aussi divers que SNA d'IBM, X25, etc.

5.3.2 ISM/OpenMaster Agent

Sur chaque machine du système distribué tourne au moins un agent responsable du contrôle de cette machine. Les agents, qui constituent en fait les représentants de l'ISM/Manager sur la machine gérée, doivent rendre les objets gérés visibles pour l'ISM/Manager. Ce sont eux qui gèrent, localement, les différents objets composant la MIB. Ils doivent également effectuer toutes les opérations de gestion demandées par

l'ISM/Manager. Il existe donc différents types d'agents implémentés dans ISM/OpenMaster :

- les agents SNMP,
- les agents CMIP,
- les agents DSAC/AEP et
- les agents GMP.

5.4 La MIB d'ISM/OpenMaster.

Toutes les classes d'objets gérés de la MIB d'ISM/OpenMaster sont définies selon le standard contenant les spécifications des objets de l'architecture d'ISM/OpenMaster, ISM/GDMO. Cette architecture est basée sur la norme [ISO10165-4].
La MIB ISM/OpenMaster contient toutes les classes d'objets nécessaires à la gestion de l'ensemble du système distribué. Les classes d'objets vont donc représenter :

- des composantes locales (logiciels, périphériques, etc.)
- des utilisateurs
- des composantes de communication (interfaces, circuits, etc.)
- des abstractions du système distribué (vendeurs, localisation de composants, etc.)
- des objets de gestion (agents, fichiers de journalisation , etc.)

Comme nous l'avons vu précédemment, ISM/OpenMaster supporte plusieurs protocoles de gestion. La MIB ISM/OpenMaster devra dès lors contenir les classes d'objets gérés conformes aux protocoles SNMP, DSAC, CMIP, etc.

La MIB ISM/OpenMaster est représentée par un ensemble de sous-MIB appelées *MIBlets*. Chacune des MIBlets est un sous-arbre de la MIB ISM/OpenMaster et est attachée directement à la racine (comme illustré à la Figure 25). L'instance de l'objet géré attachée à la racine porte le nom de ***Rootlet***.

Chaque MIBlet est instanciée auprès de l'ISM/Manager afin qu'elle soit connue par le système. Les éléments d'ISM/OpenMaster qui se chargent de cette instanciation

s 'appellent les ***gestionnaires d'objets***[33]. Les intégrateurs d'agents, qui sont des gestionnaires d'objets, ont pour fonction de rendre visibles les éléments avec lesquels ils communiquent. Ainsi, l'intégrateur d'agent SNMP va instancier la MIBlet d'un agent SNMP qui contient les classes d'objets dérivées des objets SNMP (par exemple : SNMP System, TCP group, etc.).

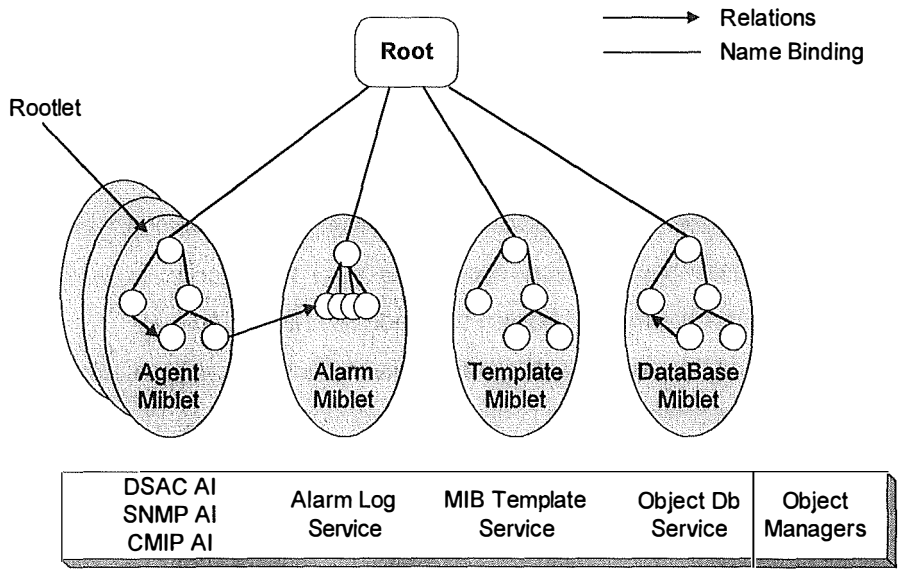

Figure 25 : La MIB d'ISM/OpenMaster

Malgré cette division de la MIB en MIBlets, la MIB d'ISM/OpenMaster est vue comme une seule entité où toutes les MIBlets partagent la même racine. Une application peut effectuer des opérations sur des instances d'objets gérés au sein de la MIB sans en connaître le gestionnaire d'objet qui s'en occupe. L'application mentionne simplement le DN de l'instance d'objet géré et c'est le CMIS Request Broker qui se charge d'orienter la requête vers le gestionnaire d'objet adéquat.

Nous retrouvons deux types d'objets au sein de la MIB. Les objets statiques et les objets dynamiques.
Les objets *statiques* sont crées et supprimés peu fréquemment. Comme ils sont présents plus longtemps dans le système, les utilisateurs et les opérateurs en ont connaissance. Ils représentent :

[33] Les services ainsi que les intégrateurs d'agents sont des gestionnaires d'objets.

- des composantes matérielles : une imprimante, un modem, ...
- des configurations logicielles : une version d'un système d'exploitation par exemple.

Les objets *dynamiques* sont des objets qui seront créés et supprimés de manière régulière. Comme ils ne sont pas tout le temps présents dans le système, ils n'est pas utile de leur attribuer un nom explicite ou de les représenter sous forme d'icône. Les objets qui répondent à cette caractéristique sont par exemple :

- des connexions : des connexions TCP, ...
- des enregistrements d'événements dans des journaux : les alarmes, ...

Notons que la MIB d'ISM/OpenMaster est extensible et ce pour plusieurs raisons :

- pour prendre en considération des nouveaux appareils qui sont gérés par les agents
- pour développer de nouveaux agents
- pour ajouter de nouvelles classes d'objets à la base de données CMIS (CMIS-DB).

5.5 Les outils de développement d'ISM/OpenMaster.

Les administrateurs, soucieux de trouver une solution intégrée pour la gestion de leur système distribué, sont bien souvent contraints d'utiliser les seules applications offertes par le système de gestion qu'ils vont choisir. En plus des applications intégrées dans ISM/OpenMaster, Bull offre aux administrateurs des outils de développement leur permettant de développer leurs propres applications de gestion. Ce développement se fera grâce au langage propriétaire ISM Management Language, mieux connu sous le nom de SML. Il s'agit d'un langage interprété de haut niveau, proche du LISP. SML permet par exemple d'afficher une valeur représentée graphiquement en une seule ligne de code. Il réagit à des événements dans la même philosophie que X-Windows.

En plus de ce langage de développement, Bull offre une série de boîtes à outils. Les deux principales boîtes à outils sont l'*AI Toolkit*[34] et le *SNMP Agent Toolkit* (SAT). La première permet à chaque administrateur de créer son propre AI, au cas où il utiliserait un protocole autre que CMIP, AEP ou SNMP. Quant à la seconde boîte à outils, elle facilite le développement de nouveaux agents SNMP. Le SNMP Agent Toolkit comprend des librairies de fonctions C qui s'occupent de l'implémentation des opérations SNMP. Il existe en outre des supports à la compilation (*scripts*, *makefiles*), au debogage et aux tests de fonctionnalités de l'agent.

5.6 Les modèles de gestion offerts par ISM/OpenMaster

5.6.1 Le modèle de gestion centralisée

Le concept manager-agent est à la base d'un outil tel qu'ISM/OpenMaster. Les ISM/Agents tournent sur la machine administrée et agissent pour le compte d'un ISM/Manager : ils collectent des données sur base des requêtes émises par l'ISM/Manager dont ils dépendent.

5.6.2 Le modèle de gestion partiellement distribuée

Une hiérarchie à deux niveaux ne suffit en général pas lorsque l'on développe une solution de gestion pour de grands systèmes distribués. Dès lors, la solution de gestion doit s'adapter à la hiérarchie de ces systèmes. C'est pourquoi, ISM/OpenMaster peut être mis en cascade.

Un ISM/Manager peut alors fédérer plusieurs autres managers ISM. Cet ISM/Manager est appelé supra-manager.

L'architecture est alors à trois niveaux :

1 Supra-ISM/Managers

[34] Cette boîte à outils est appelée *ADK CMIP* et permet de développer à la fois des agents, des intégrateurs d'agents ainsi que des services.

2 ISM/Managers

3 agents.

Les managers sont dès lors des relais de gestion pour le supra-manager.

ISM/OpenMaster offre de nombreuses autres possibilités de gestion partiellement distribuée au moyen de l'ISM/Concentrator.

ISM/Concentrator est un ISM/Manager sans les applications. Seuls les intégrateurs d'agents ainsi que les services d'ISM/OpenMaster (performance, alarm,...) sont fournis. Un concentrateur n'est pas autonome et demande à être configuré par un ISM/Manager.

On peut dès lors arriver à des hiérarchies du type :

1 ISM/Managers

2 ISM/Concentrators

3 agents

voire même :

1 Supra-ISM/Managers

2 ISM/Managers

3 ISM/Concentrators

4 agents.

5.6.3 Le modèle de gestion distribuée

Le chapitre suivant, dans lequel nous allons décrire *generix*, s'inscrit dans la perspective d'un modèle de gestion distribuée tel que nous l'avons vu, à savoir : la délégation d'applications de gestion vers les agents. Même s'il n'existe pas beaucoup d'exemples de ce type au sein d'ISM/OpenMaster, on peut s'attendre au développement croissant d'agent intelligent de ce type.

5.7 Conclusion

Qu'il s'agisse de gérer des serveurs, des systèmes Unix propriétaires ou ouverts voire même des réseaux d'ordinateurs individuels, ISM/OpenMaster est sans conteste la

solution intégrée la plus adéquate. Elle agit en effet dans un milieu tout à fait hétérogène, respecte la plupart des standards émis par l'ISO et est compatible avec de nouvelles technologies telles que CORBA de l'OMG ou encore XMP de X/open. Pour satisfaire de telles contraintes, cette plate-forme logicielle fournit aux utilisateurs des applications intégrées de gestion.

Tout comme nous l'avons vu, ISM/OpenMaster possède également un ensemble d'outils de développement permettant à ses utilisateurs de construire leurs propres solutions de gestion.

6. Generix : un agent générique

6.1 Introduction

Comme nous l'avons vu dans les chapitres précédents, les agents jouent un rôle important dans la gestion des systèmes distribués. Actuellement, le développement d'un agent de gestion est assez difficile, même si l'on possède une boîte à outils tel que le SNMP Agent Toolkit, et demande beaucoup de temps. Ce chapitre présente une solution réalisée lors du stage effectué chez Bull. Cette solution porte le nom de *generix*, pour définir un agent générique ou extensible. Nous étudierons les détails de cet agent dans les premières sections de ce chapitre. En plus d'offrir cette possibilité, l'agent s'inscrit également dans la perspective d'un agent intelligent puisqu'il répond aux caractéristiques du modèle de gestion distribuée vu auparavant. Avant de conclure ce sixième chapitre, nous parlerons des perspectives futures afin de voir quels pourraient être les améliorations ainsi que les nouveaux développements propres à *generix*.

6.2 L'environnement de développement

L'agent a été développé sur un DPX/20[35] par le biais d'un terminal X. Comme système d'exploitation, nous avons utilisé une version UNIX, le *Base Operating System AIX (BOSX)*.

Le langage de programmation C ainsi que le SNMP Agent Toolkit[36] nous ont servi pour le développement de l'agent. Avant de continuer, il nous semblait nécessaire de donner quelques informations supplémentaires sur le SNMP Agent Toolkit. Comme nous le verrons, cette boîte à outils, fort utile selon nous, présente cependant des inconvénients qui pourraient être résolus au moyen du *package* fourni avec *generix*.

---

[35] Le DPX/20 est une station propriétaire de Bull équivalente à un *IBM Risc System 6000*

[36] Nous avons déjà mentionné cette boîte à outils au chapitre précédent. Pour rappel, cette boîte à outils est un outil de développement d'agents SNMP existante avec ISM/OpenMaster.

6.2.1 Buts de SAT

Le SNMP Agent Toolkit est un ensemble de librairies et d'utilitaires qui facilitent le développement d'un agent SNMP.

Cette boîte à outils permet au développeur de se concentrer uniquement sur le code pertinent aux méthodes qu'offre l'agent. Toutes les autres tâches que l'on peut retrouver habituellement dans tout agent sont traitées par SAT. Des exemples de ces tâches peuvent être :

- la réception et l'interprétation des requêtes *get, next* et *set*,
- l'encodage et le décodage des OIDs,
- le multiplexage entre les différentes méthodes propres à l'agent : lorsque SAT reçoit une requête pour un OID particulier, il dirige celle-ci vers la méthode appropriée (préalablement encodée par le développeur).

SAT propose également un certain nombre d'utilitaires permettant de tester l'agent. Ces utilitaires correspondent en fait à des simulations de requêtes telles que des *get, getnext,...*

6.2.2 Avantages et inconvénients de SAT

Les principaux avantages d'une telle boîte à outils ont été évoqués dans la sous-section précédente.

Quant aux inconvénients, on pourrait les qualifier de *temporels*. En effet, même si le développeur ne doit pas se préoccuper de la manière dont les requêtes vont être reçues et analysées, le temps d'apprentissage du SNMP Agent Toolkit est assez long. Ce qui se répercute directement sur le temps de développement de l'agent.

De plus, lorsque l'on veut développer un nouvel agent, on doit obligatoirement suivre la même démarche :

1°) conception de l'agent,

2°) apprentissage de SAT,

3°) implémentation de l'agent,

4°) tests.

Il serait plus facile de ne pas devoir suivre cette liste pour chaque nouvel agent que l'on désire développer même s'il est vrai que l'apprentissage de SAT est unique. Nous verrons dans la suite, comment *generix* pourrait réduire le temps de développement d'un agent.

6.3 Rôle et fonctionnement de l'agent

Jusqu'à présent, lorsque le client veut remonter ses propres informations, il ne peut passer directement par un agent d'ISM/OpenMaster. Il doit suivre un chemin bien précis :

- écrire une nouvelle MIB correspondant aux objets qu'il désire gérer,
- écrire l'agent qui réalisera ses objectifs,
- introduire ces deux composantes au sein d'ISM/OpenMaster.

Une alternative à cette façon de faire serait de permettre au client d'utiliser ses propres scripts tout en lui offrant la possibilité de visualiser le résultat de ceux-ci. C'est dans cette optique principale que l'agent générique *generix* a été développé, car, grâce aux particularités intrinsèques de sa MIB, il va pouvoir s'adapter à de nouvelles contraintes.

Exemple de fonctionnement :

Une personne, travaillant chez Bull, reprochait à un agent SNMP existant de ne pas donner la visibilité des filesystems montés par NFS[37]. Dans tous les cas, il faut écrire une MIB, la compiler et l'installer sur ISM/OpenMaster. Comme dans ce cas, il n'y a qu'une table avec quelques attributs, il faut environ une à deux heures. Ensuite, il faut un agent SNMP qui remonte l'information. Pour cela, il existe deux solutions :

- *développer un agent SNMP avec SAT : pour la formation sur SAT et écriture des méthodes, nous nous sommes rendu compte qu'il fallait entre un et demi à deux mois.*

[37] Grâce au produit NFS, *Network File Systems*, une machine sous UNIX peut mettre à disposition des autres machines une partie de ses fichiers sans que le concept de gestion de fichier ne nécessite de modification. Pour l'utilisateur final, cela signifie qu'il lui est impossible de savoir immédiatement si les fichiers auxquels il accède sont placés sur sa propre machine ou sur un poste quelconque du réseau sur lequel il est connecté.

- *utiliser generix au moyen de la commande **'lsfs -v nfs'** sur AIX et dès lors, la surveillance des filesystems montés par NFS pourra être assurée par ISM/OpenMaster. Cet exercice a été fait en quelques heures. On peut sans conteste affirmer que l'on permet de réduire les coûts[38] de l'entreprise.*

L'agent SNMP générique est un agent tournant sur des plates-formes UNIX

L'agent assure les fonctions suivantes :

- exécution de la commande définie dans le script,
- formatage du résultat de la commande,
- écriture de ce résultat en objets MIB.

Le résultat pourra être parcouru grâce, entre autre, à ISM/Monitor.

En plus d'être générique, *generix* s'intègre dans la perspective d'un agent intelligent. Dans de grands systèmes distribués, de part son paradigme de modèle de gestion centralisée, ISM/OpenMaster est quelque peu surchargé. Un agent, comme *mesurix*[39], permet de faire une gestion par délégation. En effet, l'ISM/Manager n'a pas besoin de faire du polling de ressources. Il doit avoir les fonctionnalités suivantes :

- collecte de données,
- calcul de nouvelles variables à partir des données de base,
- comparaison des données ou variables à des seuils,
- corrélation portant sur plusieurs variables et
- notification au manager si un problème est détecté ou lancement d'une action correctrice.

La collecte de données peut être divisée en deux parties :

1 interrogation des agents SNMP existants et

---

[38] Voir à ce sujet la section 2.3.2.

[39] *Mesurix* est un agent de mesures de performances développé chez Bull. Voir à ce sujet le mémoire de Marc Nosbusch dans [NOSBUS,95]

2 récupération de données à partir de commandes (➔ *generix*)

Generix devra donc posséder une interface offrant à un agent de log les fonctionnalités suivantes:

- activation d'un script,
- parcours de la table résultante,
- récupération des données.

Une sauvegarde automatique de la configuration de l'agent est effectuée à intervalle régulier ou lorsqu'un signal vient interrompre le processus. De cette manière, l'utilisateur peut facilement récupérer ses informations.

La configuration de *generix* est faite de la même manière que celle d'Alixd[40]. Elle est effectuée au travers de la MIB SNMP et plus précisément au travers de sa première table (voir infra pour la définition de cette table). Ainsi, la configuration du script traité par l'agent peut se faire en utilisant ISM/Monitor.

Il existe également un mécanisme de mot de passe sécurisant la configuration des deux premières tables (generixTableTable et generixAttributeTable) de la MIB de *generix*. Puisque l'utilisateur peut lancer un shell script quelconque au travers de l'agent, on peut très facilement se rendre compte de l'impact d'une commande tel qu'un shutdown qui serait effectuée par un utilisateur ne possédant pas les droits adéquats.

C'est pour éviter de tels désagréments que *generix* est doté d'un mécanisme de mot de passe.

Pour cela, deux attributs ont été définis dans la MIB de *generix* :

- generixOldPasswd et
- generixNewPasswd.

Ce mécanisme est identique à celui qui est utilisé dans Agix[41]. Ainsi, si ces deux attributs ne sont pas initialisés, leur valeur par défaut sera égale à "CMIS NULL". Dans ce cas,

---

[40] Alixd est un agent SNMP existant sur des plates-formes UNIX dont le but est de surveiller différentes sources d'alarmes et d'envoyer des notifications ou d'exécuter des actions locales lorsqu'un problème survient. La configuration de cet agent est faite par le biais de sa MIB de telle manière à pouvoir modifier son comportement en utilisant ISM/Monitor.

[41] Agix est un agent SNMP composé, entre autre, d'Alixd.

toute configuration de l'une des deux tables sera acceptée. Ce n'est que lorsque l'utilisateur aura rempli le champ **generixNewPasswd** que le mot de passe sera initialisé.

Après cette opération, toute configuration sera sécurisée. Si l'utilisateur décide de changer son mot de passe, il devra changer les deux champs définis par generixOldPasswd et generixNewPasswd. La figure 26 illustre cette possibilité de configuration.

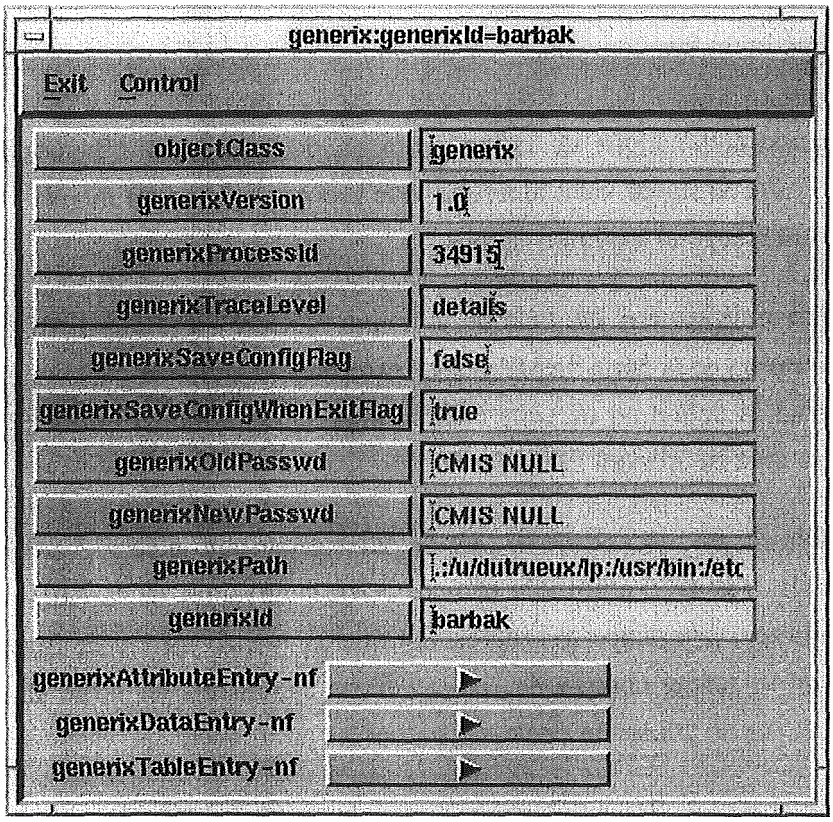

Figure 26 : Fenêtre d'ISM/Monitor permettant la configuration des attributs de generix

6.4 Architecture de l'agent générique

Le dispatcher SNMP s'occupe de la réception des requêtes (sur le port 161) provenant de l'AI SNMP et il redirige celles-ci vers l'agent destinateur (sur un port spécifique). Après avoir récolté les réponses, il retourne ces dernières vers l'AI SNMP (voir figure 27).

Generix reçoit des requêtes SNMP du dispatcher SNMP, il les exécute et il lui retourne ensuite les réponses à ces requêtes. Les traps sont envoyées sur le port 162 de l'AI SNMP. Les ports 161 et 162 sont les ports standards utilisés par SNMP.

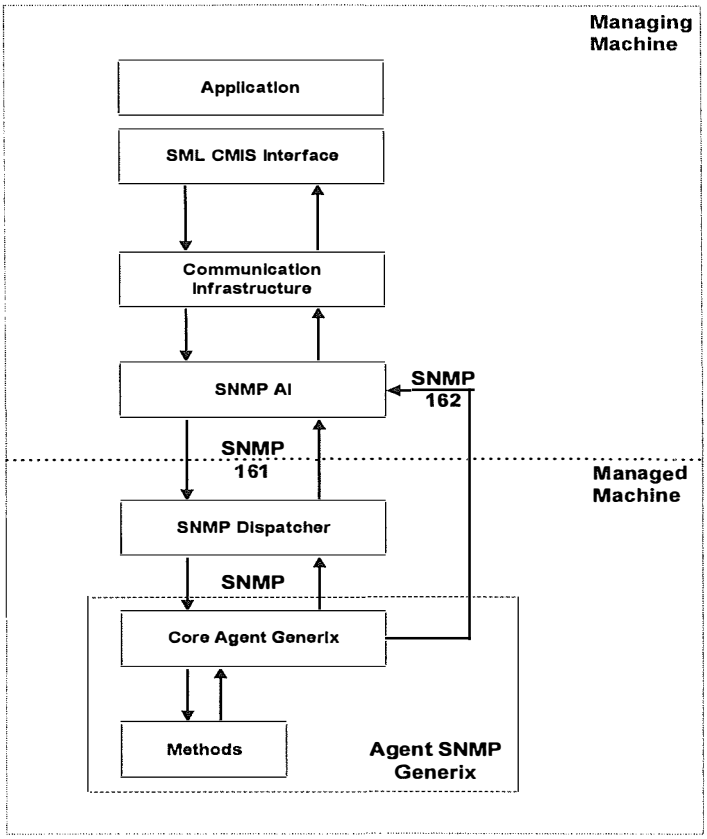

Figure 27 : Intégration de generix dans ISM/OpenMaster

On voit donc que *generix* possède deux types "d'utilisateurs" directs :

- le dispatcher SNMP, qui envoie les requêtes vers l'agent et qui reçoit les réponses, et
- l'AI SNMP, qui reçoit les traps.

6.5 Description de la MIB de *generix*

L'agent va être utilisé pour remonter les informations vers ISM Monitor au travers de la MIB. La description de celle-ci se trouve à l'annexe 1 de ce mémoire.

La MIB de *generix* est composée des trois tables dont la description succincte va être proposée dans les sections suivantes :

- GenerixTableTable

- GenerixAttributeTable,
- GenerixDataTable.

La combinaison de ces trois tables permet d'obtenir un agent générique puisque n'importe quel script peut être utilisé. Afin d'offrir une meilleure compréhension du fonctionnement de l'agent, nous allons illustrer la description des différentes tables au moyen d'un exemple.

6.5.1 La table GenerixTableTable

La table GenerixTableTable permet de formater la sortie standard du script. Elle va permettre de définir la commande que l'on va passer, la manière de séparer les différents champs ainsi que la manière d'analyser les lignes de la table.

Cette table est essentielle car elle définit non seulement ce que l'agent doit réaliser mais également la manière de le faire.

Exemple d'utilisation :

Tout au long de cette section 6.5, nous allons illustrer l'utilisation des différentes tables au moyen d'un exemple concret. Nous considérerons que l'utilisateur veut remonter les informations de la commande ***df***[42]*. Voici quel pourrait être le résultat*[43] *d'une telle commande :*

un attribut	**un attribut**					
Filesystem	1024-blocks	Used	Available	Capacity	Mounted on	
/dev/hd4	8192	6776	1416	83%	/	→ **instance n°1**
/dev/hd9var	12288	6436	5852	52%	/var	
/dev/hd2	651264	601664	49600	92%	/usr	
/dev/hd3	65536	24308	41228	37%	/tmp	
/dev/hd1	4096	2540	1556	62%	/home	
bison:/ISMCOMMON/4C_F/AIX	1048576	776412	272164	74%	/mnt3.51	

[42] La commande ***df*** (en anglais disk free) indique la place restante sur un support de données. Le résultat est exprimé le plus souvent en blocs (de 512 octets). En l'absence de tout paramètre, la commande affiche les espaces disponibles sous forme abrégée, nombre de blocs et d'en-têtes de fichiers libres.

[43] Nous tenons à faire remarquer que le résultat qui se trouve ici est différent du résultat présent dans les différentes figures illustrant les trois tables.

La première étape va être de remplir les champs de la première table afin de savoir quelle sera la commande à exécuter. La figure 28 nous permet de voir graphiquement la manière de remplir la première table.

Dans cet exemple, nous n'avons pas utilisé de filtre (generixTableFilterValue) mais si tel était le cas, nous aurions pu ne retenir que les instances possédant cette valeur. L'attribut generixTableValidityDuration permet de spécifier la durée pendant laquelle les données resteront dans le cache. Lors de l'implémentation, nous avons en effet été confrontés à un problème d'optimisation pour la troisième table. C'est pourquoi, nous avons utilisé des caches.

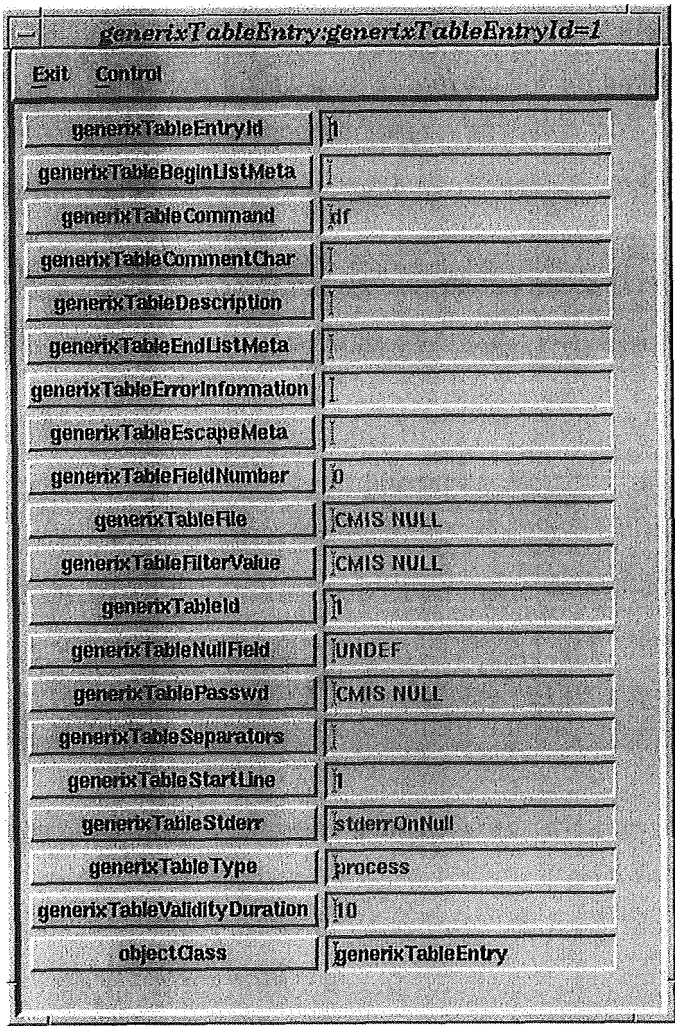

Figure 28 : Configuration de la table GenerixTableTable

6.5.2 La table GenerixAttributeTable

La seconde table va permettre d'associer une colonne de la sortie du script à un attribut de la MIB, soit un attribut décrit dans la troisième table.

Exemple d'utilisation :

Lorsque l'utilisateur va ouvrir cette deuxième table, il va être confronté à 21 valeurs, toutes initialisées. Toutes ces valeurs correspondent en fait aux 21 champs définis dans la table GenerixDataTable. Dans notre exemple, puisque nous n'avons pas utilisé de filtre, toutes les instances seront mappées en objets à l'intérieur de la MIB.

Grâce à cette seconde table, l'utilisateur peut lui-même définir le nom qu'il veut donner aux champs de la troisième table. Ainsi, s'il veut nommer le premier attribut "Système de fichiers" au lieu de FileSystem, rien ne s'y oppose. Par défaut toutefois, on retrouvera comme nom d'attribut le nom utilisé par la sortie standard. La figure 29 nous permet de voir comment on peut associer une colonne de la sortie du script à un attribut de la MIB.

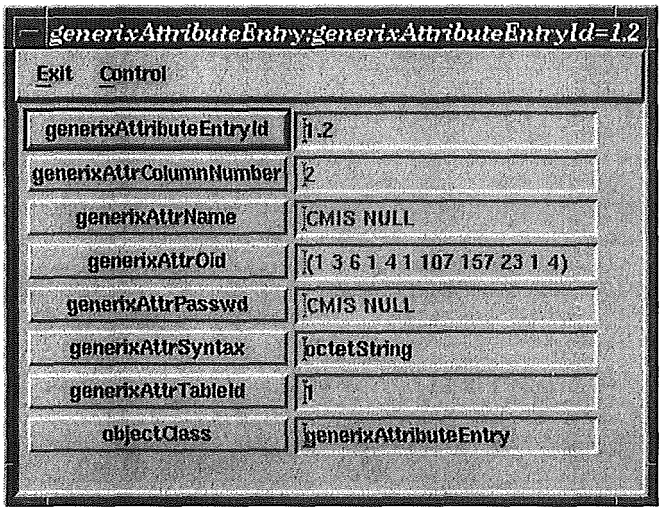

Figure 29 : Association d une colonne de la sortie du script à un attribut de la MIB

6.5.3 La table *GenerixDataTable*.

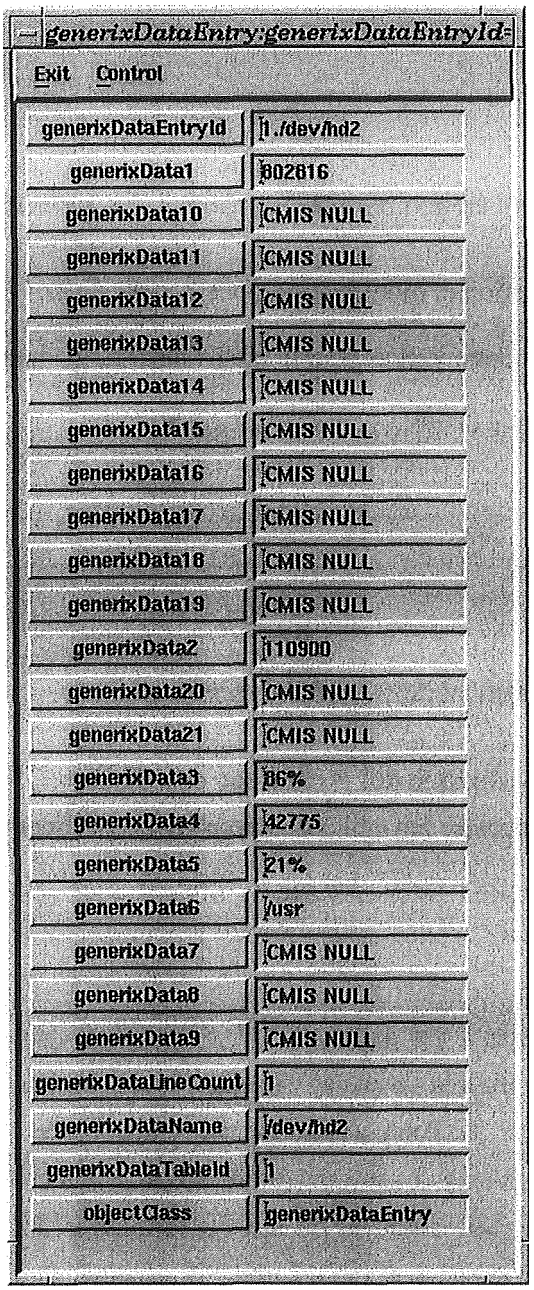

Figure 30 : Consultation d'une instance grâce à la table GenerixDataTable

Cette table va permettre de visualiser les informations à travers la MIB. C'est à partir de cette table que l'on va prendre les informations issues de la commande. Celle-ci avait été définie dans la table GenerixTableTable.

Exemple d'utilisation :

La figure 30 illustre la visualisation, toujours grâce à ISM/Monitor, d'une instance liée au résultat de la commande ***df****.*

De par leur définition, les tables présentées supportent les opérations SNMP suivantes :

- création, destruction et modification pour la table GenerixTableTable,
- modification et consultation pour la table GenerixAttributeTable,
- consultation pour la table GenerixDataTable.

6.6 Conclusion

Generix possède actuellement deux modes d'utilisation :

- visualisation dynamique des sorties des commandes UNIX ou de tout autre programme au moyen des tables que nous avons décrites,
- possibilité d'implémenter des agents SNMP pour remonter des informations vers le manager.

Le premier mode permet à cet agent SNMP d'être générique car tout utilisateur peut dorénavant utiliser ses propres scripts et visualiser leurs résultats. Suite à notre stage, les principales améliorations que l'on pourrait apporter sur ce mode d'utilisation relèvent de la programmation. Nous avions, à cette occasion, proposé des améliorations au niveau de l'allocation des caches mais aussi au niveau du formatage du résultat en utilisant des analyseurs syntaxiques (*Awk*[44], *Lex*, etc.). L'objectif principal était cependant atteint.

Dans le second mode d'utilisation, nous pouvons retrouver la notion de *package*. Nous entendons par là qu'il est possible d'utiliser *generix* pour implémenter un nouvel agent SNMP, tout simplement en utilisant un package comprenant :

[44] Awk, dont le nom est la contraction des initiales de ses concepteurs Aho, Weinberger et Kernighan, est un langage de programmation qui permet la manipulation aisée de données structurées ainsi que la génération de rapports formatés.

- un fichier de configuration,
- un fichier de correspondance entre un nom d'attribut et son OID et
- l'ensemble des programmes qui seront forkés par *generix*.

Les principaux avantages d'un tel mode se retrouvent principalement dans la réduction du temps de développement d'un agent[45] ainsi qu'au fait de ne pas devoir connaître SAT. Ce mode d'utilisation présente cependant une limitation par rapport à ce qu'il est possible de faire avec SAT ainsi qu'un inconvénient en ce qui concerne l'utilisation de la mémoire[46].

En terme de perspectives futures, nous envisagerions de continuer le développement de l'agent de performances pour lequel *generix* constitue une brique de base. Cela nous permettra ainsi d'avoir un agent de performances intelligent, s'inscrivant dans la perspective d'un modèle de gestion distribuée.

[45] L'utilisateur de *generix* concentre ses efforts sur les programmes qui permettent d'obtenir l'information qu'il souhaite remonter vers le manager.

[46] Principalement face à des tables de plus d'une centaine d'instances.

7. Conclusions

La gestion d'un système distribué est entraînée dans une spirale de complexité croissante. Les ressources, tant matérielles que logicielles, de ces systèmes informatiques se multiplient et proviennent de constructeurs différents. Parallèlement, les besoins des utilisateurs en terme de gestion sont énormes et exigeants.
En effet, face à la complexité et à l'hétérogénéité d'un système distribué, sa gestion est devenue de plus en plus complexe. D'autre part, si les entreprises ont opté pour la mise en place d'une architecture de gestion fortement centralisée, l'empilement des nombreuses applications sur une seule machine de gestion, le manager, entraîne des problèmes non négligeables de performances : il faut en effet envoyer des informations de gestion à travers le système et attendre une réponse, ce temps de délai étant sensiblement augmenté par la gestion du protocole de communication.

Face à de telles exigences, le rythme d'apparition sur le marché de plates-formes de gestion d'un système distribué semble s'être accéléré depuis quelques années. Les services qu'elles offrent se rapprochent des besoins à couvrir. Dans l'ensemble, les différentes plates-formes proposent des modèles de gestion autres que le modèle de gestion qui était jusqu'alors de mise, le modèle de gestion centralisée.

Nous avons décrit, tout au long de ce mémoire, le rôle et l'importance que peuvent jouer les agents de gestion. Nous avons présenté l'architecture et le fonctionnement de l'agent générique développé lors de notre stage. Outre ses capacités d'extension (de par son aspect configurable), *generix* se greffe dans la philosophie d'un agent de mesures de performances intelligent, ce qui lui permet de s'inscrire dans un modèle de gestion distribuée d'un système distribué.

Ce mémoire nous a permis d'établir, dans une première partie, les bases théoriques d'une gestion d'un système distribué. Pour y parvenir, nous avons choisi, dans un premier temps, d'isoler et de caractériser la gestion d'un système informatique. Nous avons ensuite décrit la notion de système distribué tout en montrant la complexité inhérente à la gestion d'un tel système. Pour clôturer la partie théorique, nous avons présenté les principaux modèles de gestion d'un système distribué : le modèle de gestion centralisée, partiellement distribuée et distribuée.
Dans la seconde partie, nous avons décrit un outil de gestion intégrée conforme au modèle théorique puisqu'il s'intègre parfaitement dans les caractéristiques énoncées. Cet

outil n'est autre qu'ISM/OpenMaster, une plate-forme logicielle propriétaire de Bull. Nous nous sommes ensuite attardé sur *generix*, l'agent générique, tout en évoquant ses perspectives futures de développement.

A terme, la complexité croissante des systèmes distribués entraînera un changement total dans le développement de plates-formes de gestion. Le confort d'utilisation des plates-formes de gestion de système distribué ne peut que croître, allant de pair avec l'adoption du modèle de gestion distribuée.

Liste des acronymes

ACSE	Association Control Service Element
AEP	Administrative Exchange Protocol
AI	Agent Integrator (Intégrateur d'Agents)
API	Application Programmatic Interfaces
ASN.1	Abstract Syntax Notation One
CMIP	Common Management Over TCP/IP
CMIS	Common Management Information Service
CMOT	Common Management Information Protocol
CORBA	Common Object Request Broker Architecture
DSAC	Distributed Systems Administration and Control
FDDI	Fiber Distributed Data Interface
GMP	GCOS Management Protocol
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISM	Integrated System Management
ISO	International Organization for Standardization
ITU-T	International Telecommunication Union - Telecommunication Standardization Sector
LAN	Local Area Network
MBD	Management By Delegation
MIB	Management Information Base
NFS	Network File Systems
NMS	Network Management Station
OID	Object Identifier
OMG	Object Management Group
OSI	Open Systems Interconnection
OSI/NMForum	OSI/Network Management Forum
POSIX	Portable Operating Systems Interface for Unix
RFC	Request For Comments
ROM	Root Object Manager
ROSE	Remote Operation Service Element
SAT	SNMP Agent Toolkit

SMI	Structure of Management Information
SML	System Management Language
SMT	Station Management
SNA	Systems Network Architecture
SNMP	Simple Network Management Protocol
SQL	Simple Query Language
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WAN	Wide Area Network
XMP	X/Open Management Protocol

Bibliographie

[AGENTW,95] AgentWorks : *"Technology for Distributed System Management."*, Technical White Paper, Legent Corporation, 1995

[ARPEGE,92] Arpège : *"Gestion de réseaux : concepts et outils"*, Préface de Louis Pouzin, Masson, 1992

[CF6CMI,93] CF6 : *"Le protocole CMIP."*, Conseil et Formation Système, Paris La Défense, 1992

[CF6SNM,92] CF6 : *"Le protocole SNMP."*, Conseil et Formation Système, Paris La Défense, 1992

[DEGHOR,92] Deghorain H. : *"L'administration des systèmes distribués. Approche de la gestion OSI"*, Facultés Universitaires Notre-Dame de la Paix, Institut d'Informatique, Namur, 1992

[GHLDD,94] Emsley I. : *"ISM3 : Global High Level Design Document"*, Document interne Bull, draft du 17.06.94

[GOLDSZ,94] Goldszmidt G. : *"On Distributed System Management."*, Distributed Computing and Communication Lab, Computer Science Department, Columbia University, 1994

[GOLDSZ,95] Goldszmidt G., Yemini Y. : *"Decentralizing Control and Intelligence in Network Management."*, Computer Science Department, Columbia University, 1995

[GOLYEM,95] Goldszmidt G., Yemini Y. : *"Distributed Management by Delegation"*, Proceedings of the 15th International Conference on Distributed Computing Systems, 1995

[HEYVAE,91] Heyvaert D. : *"Network Management Systems. The management of TCP/IP networks."*, Facultés Universitaires Notre-Dame de la Paix, Institut d'Informatique, Namur, 1991

[ISO7498-4] ISO/IEC 7498-4, *"Information Processing Systems - Open Systems Interconnection, Basic Référence Model - Part 4 : Management Framework"*, International Standards Organization, 15 novembre 1989

[ISO9596] ISO/IEC 9596 : *"Information technology -Open Systems Interconnection- Common Management Information Protocol*

Specification", International Standard Organisation, 1991

[LANGSF,93] Langsford A., Moffett J., *"Distributed Systems Management."*, Addison-Wesley, 1993

[NOSBUS,95] Nosbusch M. : *"Développement d'un agent de mesures de performances pour la gestion des systèmes distribués."* Facultés Universitaires Notre-Dame de la Paix, Institut d'Informatique, Namur, 1995

[PERKIN,93] Perkins D. : *"Understanding SNMP MIBs"*, September, 1993

[PRAS,95] Pras A. : *"Network Management Architectures"*, Thesis University of Twente, The Netherlands, 1995

[RFl156,90] RFC 1156 : *"Management Information Base for network management of TCP/IP-based internets"*, Rose M.T. : McCloghrie K., May 1990

[RFl157,90] RFC 1157 : *"Simple Network Management Protocol (SNMP)"*, Case J.D., Fedor M., Schoffstall M.L., Davin C., May 1990

[ROSEMA,94] Rose Marshall T. : *"The Simple Book - Second Edition"*, Prentice-Hall International Editions, 1994

[SLOMAN,94] Sloman M., *"Network and Distributed Systems Management"*, Addison-Wesley Publishing Company, 1994

[SNMPAG,91] *"Integrated System Management, Programmers's Guide. SNMP Agent Toolkit. Unix."*, Bull S.A., 1993, 1994

[STALLI,93] Stallings W. : *"SNMP, SNMPv2 and CMIP - The Practical Guide to Network Management Standards"*, Addison Wesley, 1993

[TANENB,93] Tanenbaum A. : *"Réseaux, architectures, protocoles, applications."*, InterEditions, 1993

[VANBAS,95] Van Bastelaer P. : *"Chapitre relatif à SNMP. Description d'un protocole de gestion de réseau : le protocole SNMP"*, Facultés Universitaires Notre-Dame de la Paix, Institut d'Informatique, Namur, 1995

[VANRYC,96] Van Ryckeghem F. : *"Etude de l'implémentation des standards ISO de performance au sein d'ISM/OpenMaster."*, Facultés Universitaires Notre-Dame de la Paix, Institut d'Informatique, Namur, 1996

[VINCEN,94] Vincent S. : *"La gestion et l'évolution des systèmes distribués."*, Facultés Universitaires Notre-Dame de la Paix, Institut d'Informatique, Namur, 1994

[WWWMIE,96] Mier E. : *"Divide and conquer : attacking distributed network management."*, Network World Fusion, 1996

[WWWNET,96] TechForum : *"Netrix. Distributed SNMP Management Whitepaper."*, http://www.netrix.com/techno/snmp.html, 1996

Description de la MIB de *generix*

```
GenerixASN1Module DEFINITIONS ::= BEGIN

--
-- the SNMP mib root
--

generix  OBJECT IDENTIFIER ::= { 1 3 6 1 4 1 107 157 }

--
--    Generix attributes
--

generixVersion OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..12))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            "Version of the generix daemon."
    ::= { generix 1 }


generixProcessId  OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
     DESCRIPTION
             "The generix process identifier"
     ::= { generix 2 }

 generixTraceLevel OBJECT-TYPE
     SYNTAX INTEGER {
                none(0),
                requests(1),
                functions(2),
                details(3)
            }
     ACCESS read-write
     STATUS mandatory
     DESCRIPTION
             "The generix trace level"
     ::= { generix 3 }

 generixSaveConfigFlag OBJECT-TYPE
     SYNTAX INTEGER {
      true(1),
```

```
        false(2)
        }
        ACCESS read-write
        STATUS mandatory
        DESCRIPTION
                "Flag used to indicate if the current generixd
              configuration should be saved. This flag is set to
              'true' with a snmp request and is automatically
              reset to 'false' by alixd when the operation
              succeeds."
        ::= { generix 4 }

    generixSaveConfigWhenExitFlag OBJECT-TYPE
        SYNTAX INTEGER {
        true(1),
        false(2)
        }
        ACCESS read-write
        STATUS mandatory
        DESCRIPTION
                "Generixd behaviour when exiting : should the
              configuration be saved ?"
        ::= { generix 5 }

    generixOldPasswd OBJECT-TYPE
        SYNTAX OCTET STRING (SIZE (0..64))
        ACCESS read-write
        STATUS mandatory
        DESCRIPTION
                "Previous password value.
              This attribute is mandatory to change the current
              password if this one has already been set."
        ::= { generix 6 }

    generixNewPasswd OBJECT-TYPE
        SYNTAX OCTET STRING (SIZE (0..64))
        ACCESS read-write
        STATUS mandatory
        DESCRIPTION
                "New password value.
              This password is used to secure the configuration
              of 'generixTable' and 'generixAttribute' tables"
        ::= { generix 7 }

    generixPath OBJECT-TYPE
        SYNTAX OCTET STRING (SIZE (0..128))
        ACCESS read-write
        STATUS mandatory
        DESCRIPTION
                "The generix PATH environment variable"
```

```
    ::= { generix 8 }

--
--   The generixPackage table
--

generixPackageTable OBJECT-TYPE
    SYNTAX SEQUENCE OF GenerixPackageEntry
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION ""
    ::= { generix 20 }

generixPackageEntry OBJECT-TYPE
    SYNTAX GenerixPackageEntry
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION
            "Package description"
    INDEX { generixPackageName }
    ::= { generixPackageTable 1 }

GenerixPackageEntry ::=
    SEQUENCE {
     generixPackageName
         OCTET STRING (SIZE (0..64)),
     generixPackageStatus
         INTEGER,
     generixPackageErrorInformation
         OCTET STRING (SIZE (0..64))
    }

generixPackageName OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..64))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            "Package name"
    ::= { generixPackageEntry 1 }

generixPackageStatus OBJECT-TYPE
    SYNTAX INTEGER {
     ok(1),
     notAvailable(2),
     inError(3)
    }
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            "Package status:
```

```
            - ok: the package is available and the
            configuration has been restored with success
            - notAvailable: the package is not available
            - inError: the package is available but the
            configuration restoration failed"
    ::= { generixPackageEntry 2 }

generixPackageErrorInformation OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..64))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            "String which gives precision about the error"
    ::= { generixPackageEntry 3 }


--
--   The generixTable table
--

generixTableTable OBJECT-TYPE
    SYNTAX SEQUENCE OF GenerixTableEntry
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION ""
    ::= { generix 21 }

generixTableEntry OBJECT-TYPE
    SYNTAX GenerixTableEntry
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION
            "Description for a new table"
    INDEX { generixTableId }
    ::= { generixTableTable 1 }

GenerixTableEntry ::=
    SEQUENCE {
    generixTableId
        INTEGER,
    generixTableDescription
        OCTET STRING (SIZE (0..64)),
    generixTableType
        INTEGER,
    generixTableCommand
        OCTET STRING (SIZE (0..64)),
    generixTableStderr
        INTEGER,
    generixTableFile
        OCTET STRING (SIZE (0..64)),
```

```
    generixTableValidityDuration
        INTEGER,
    generixTableErrorInformation
        OCTET STRING (SIZE (0..64)),
    generixTablePasswd
        OCTET STRING (SIZE (0..64)),
    generixTableStartLine
        INTEGER,
    generixTableBeginListMeta
        OCTET STRING (SIZE (0..1)),
    generixTableEndListMeta
        OCTET STRING (SIZE (0..1)),
    generixTableEscapeMeta
        OCTET STRING (SIZE (0..1)),
    generixTableCommentChar
        OCTET STRING (SIZE (0..1)),
    generixTableSeparators
        OCTET STRING (SIZE (0..10)),
    generixTableNullField
        OCTET STRING (SIZE (0..10)),
    generixTableFieldNumber
        INTEGER,
    generixTableFilterValue
        OCTET STRING (SIZE (0..64))
     }

generixTableId OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            "Identify an entry in the generixTable table"
    ::= { generixTableEntry 1 }

generixTableDescription OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..64))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
            "User defined description for this table"
    ::= { generixTableEntry 2 }

generixTableType OBJECT-TYPE
    SYNTAX INTEGER {
    process(1),
    file(2)
    }
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
```

```
            "How to retrieve informations:
          - process: fork a process and format the standart
          output
          - file: format data read from a file"
    ::= { generixTableEntry 3 }

generixTableCommand OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..64))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
            "Command to execute. The output is used to fill
          the new table.
          This attribute is used when 'generixTableType' is
          equal to 'process'"
    ::= { generixTableEntry 4 }

generixTableStderr  OBJECT-TYPE
    SYNTAX INTEGER {
     stderrOnNull(1),
     stderrOnStdout(2)
    }
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
            "Redirect the standart error:
          - stderrOnNull: redirect it on /dev/null
          - stderrOnStdout: redirect stderr on stdout
          This attribute is used when 'generixTableType' is
          equal to 'process'"
    ::= { generixTableEntry 5 }

generixTableFile OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..64))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
            "File name read to retrieve informations
          This attribute is used when 'generixTableType' is
          equal to
          'file'"
    ::= { generixTableEntry 6 }

generixTableValidityDuration OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
            "Valid data period for data in the cache, in
          seconds"
```

```
    ::= { generixTableEntry 7 }

generixTableErrorInformation OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..64))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            "String which gives precision about the error"
    ::= { generixTableEntry 8 }


generixTablePasswd OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..64))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
            "Password used to secure creations, deletions and
          modifications in the generixTable table"
    ::= { generixTableEntry 9 }

generixTableStartLine OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
            "Line number where to start analyse"
    ::= { generixTableEntry 10 }

generixTableBeginListMeta OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
            "Start list meta character"
    ::= { generixTableEntry 11 }

generixTableEndListMeta OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
            "End list meta character"
    ::= { generixTableEntry 12 }

generixTableEscapeMeta OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
            "Escape character for meta characters"
```

```
    ::= { generixTableEntry 13 }

generixTableCommentChar OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..1))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
            "Character that identify the start of a comment"
    ::= { generixTableEntry 14 }

generixTableSeparators OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..10))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
            "List of field separators"
    ::= { generixTableEntry 15 }

generixTableNullField OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..10))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
            "TBD"
    ::= { generixTableEntry 16 }

generixTableFieldNumber OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
            "Column number used by the filter. A null value
          means the the filtering feature is not used"
    ::= { generixTableEntry 17 }

generixTableFilterValue  OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..64))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
            "Filter value: the value of the field identified
          by 'generixTableFieldNumber' must have this value
          to be displayed in the output table"
    ::= { generixTableEntry 18 }

--
-- The generixAttribute table
--

generixAttributeTable OBJECT-TYPE
```

```
    SYNTAX SEQUENCE OF GenerixAttributeEntry
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION ""
    ::= { generix 22 }

generixAttributeEntry OBJECT-TYPE
    SYNTAX GenerixAttributeEntry
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION
            "Description for a new table"
    INDEX { generixAttrTableId, generixAttrColumnNumber }
    ::= { generixAttributeTable 1 }

GenerixAttributeEntry ::=
    SEQUENCE {
     generixAttrTableId
         INTEGER,
     generixAttrColumnNumber
         INTEGER,
     generixAttrOid
         OBJECT IDENTIFIER,
     generixAttrSyntax
         INTEGER,
     generixAttrPasswd
         OCTET STRING (SIZE (0..64)),
     generixAttrName
         OCTET STRING (SIZE (0..20))
    }

generixAttrTableId  OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
             "Identifier of the new table, i.e. a
           'generixTableId' attribute value. Used with the
           'generixAttrColumnNumber' attribute to identify an
           entry in the generixAttribute table"
    ::= { generixAttributeEntry 1 }

generixAttrColumnNumber OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
            "Column number in the command output.
          Used with the 'generixAttrTableId' attribute to
          identify an entry in the generixAttribute table"
```

```
    ::= { generixAttributeEntry 2 }

 generixAttrOid OBJECT-TYPE
    SYNTAX OBJECT IDENTIFIER
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
           "An object identifier for an attribute in the
          'generixDataTable' table."
    ::= { generixAttributeEntry 3 }

 generixAttrSyntax OBJECT-TYPE
    SYNTAX INTEGER {
     integer(1),
     octetString(2),
     ipAddress(3),
     counter(4),
     gauge(5),
     timeTicks(6),
     objectId(7)
    }
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            "Syntax of the attribute"
    ::= { generixAttributeEntry 4 }

 generixAttrPasswd OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..64))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
            "Password used to secure the configuration of new
          tables"
    ::= { generixAttributeEntry 5 }

 generixAttrName OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..20))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
            "Name of the attribute"
    ::= { generixAttributeEntry 6 }


 --
 --   The generixData table
 --

 generixDataTable OBJECT-TYPE
```

```
    SYNTAX SEQUENCE OF GenerixDataEntry
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION ""
    ::= { generix 23 }

generixDataEntry OBJECT-TYPE
    SYNTAX GenerixDataEntry
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION
            "Display new tables"
    INDEX { generixDataTableId, generixDataName }
    ::= { generixDataTable 1 }

GenerixDataEntry ::=
    SEQUENCE {
     generixDataTableId
         INTEGER,
     generixDataName
         OCTET STRING (SIZE (0..64)),
     generixDataLineCount
         INTEGER,
     generixData1
         OCTET STRING (SIZE (0..64)),
     generixData2
         OCTET STRING (SIZE (0..64)),
     generixData3
         OCTET STRING (SIZE (0..64)),
     generixData4
         OCTET STRING (SIZE (0..64)),
     generixData5
         OCTET STRING (SIZE (0..64)),
     generixData6
         OCTET STRING (SIZE (0..64)),
     generixData7
         OCTET STRING (SIZE (0..64)),
     generixData8
         OCTET STRING (SIZE (0..64)),
     generixData9
         OCTET STRING (SIZE (0..64)),
     generixData10
         INTEGER,
     generixData11
         INTEGER,
     generixData12
         INTEGER,
     generixData13
         Gauge,
     generixData14
```

```
        Gauge,
    generixData15
        Gauge,
    generixData16
        Counter,
    generixData17
        Counter,
    generixData18
        Counter,
    generixData19
        TimeTicks,
    generixData20
        IpAddress,
    generixData21
        OBJECT IDENTIFIER
    }

generixDataTableId OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            "Identifier of the new table, i.e. a
          'generixTableId' attribute value. Used with the
          'generixDataName' attribute to identify an entry in
          the generixData table"
    ::= { generixDataEntry 1 }

generixDataName OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..64))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            "Identifier of an instance in a new table.
          Used with the 'generixDataTableId' attribute to
          identify an entry in the generixData table"
    ::= { generixDataEntry 2 }

generixDataLineCount OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            "Number of lines in the command output used to
          fill a 'generixData' entry"
    ::= { generixDataEntry 3 }

generixData1 OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..64))
    ACCESS read-only
```

```
    STATUS mandatory
    DESCRIPTION
        ""
    ::= { generixDataEntry 4 }

generixData2 OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..64))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            ""
    ::= { generixDataEntry 5 }

generixData3 OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..64))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            ""
    ::= { generixDataEntry 6 }

generixData4 OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..64))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            ""
    ::= { generixDataEntry 7 }

generixData5 OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..64))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            ""
    ::= { generixDataEntry 8 }

generixData6 OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..64))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            ""
    ::= { generixDataEntry 9 }

generixData7 OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..64))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
```

```
            ""
    ::= { generixDataEntry 10 }

generixData8 OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..64))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            ""
    ::= { generixDataEntry 11 }

generixData9 OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..64))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            ""
    ::= { generixDataEntry 12 }

generixData10 OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            ""
    ::= { generixDataEntry 13 }

generixData11 OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            ""
    ::= { generixDataEntry 14 }

generixData12 OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            ""
    ::= { generixDataEntry 15 }

generixData13 OBJECT-TYPE
    SYNTAX Gauge
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            ""
    ::= { generixDataEntry 16 }
```

```
generixData14 OBJECT-TYPE
    SYNTAX Gauge
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            ""
    ::= { generixDataEntry 17 }

generixData15 OBJECT-TYPE
    SYNTAX Gauge
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            ""
    ::= { generixDataEntry 18 }

generixData16 OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            ""
    ::= { generixDataEntry 19 }

generixData17 OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            ""
    ::= { generixDataEntry 20 }

generixData18 OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            ""
    ::= { generixDataEntry 21 }

generixData19 OBJECT-TYPE
    SYNTAX TimeTicks
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            ""
    ::= { generixDataEntry 22 }

generixData20 OBJECT-TYPE
```

```
    SYNTAX IpAddress
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            ""
    ::= { generixDataEntry 23 }

generixData21 OBJECT-TYPE
    SYNTAX OBJECT IDENTIFIER
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
            ""
    ::= { generixDataEntry 24 }

--
--                  TRAP DEFINITIONS
--


--
--   Traps sent when generix starts or stops
--

generix-start TRAP-TYPE
     ENTERPRISE generix
     DESCRIPTION
          "The generix daemon is started"
     ::= 1

generix-panic  TRAP-TYPE
     ENTERPRISE generix
     DESCRIPTION
          "The generix daemon has been killed by a signal, or
          it has exited due to abnormal conditions."
     ::= 2

--
--   Traps sent during the configuration
--

generix-generix-setFailed TRAP-TYPE
     ENTERPRISE generix
     DESCRIPTION
          "A set request failed for a generix attribute"
     ::= 3

generix-package-setFailed TRAP-TYPE
     ENTERPRISE generix
     VARIABLES { generixPackageName }
     DESCRIPTION
```

```
          "A set request failed for the 'generixPackage'
          table"
     ::= 4

generix-table-setFailed TRAP-TYPE
     ENTERPRISE generix
     VARIABLES { generixTableId }
     DESCRIPTION
          "A set request failed for the 'generixTable' table"
     ::= 5

generix-attribute-setFailed TRAP-TYPE
     ENTERPRISE generix
     VARIABLES { generixAttrTableId, generixAttrOid }
     DESCRIPTION
          "A set request failed for the generixAttribute'
          table"
     ::= 6

generix-data-setFailed TRAP-TYPE
     ENTERPRISE generix
     VARIABLES { generixDataTableId, generixDataName }
     DESCRIPTION
          "A set request failed for the 'generixData' table"
     ::= 7


--
--   Traps sent by the extensible part of the agent
--

generix-package-problem TRAP-TYPE
     ENTERPRISE generix
     VARIABLES { generixPackageName, generixTableId }
     DESCRIPTION
          "Cannot retrieve informations to build a cache.
          A program cannot be forked, or a file cannot be
          read"
     ::= 8

generix-package-clear TRAP-TYPE
     ENTERPRISE generix
     VARIABLES { generixPackageName, generixTableId }
     DESCRIPTION
          "Trap sent when the status of a new table comes
          back to normal"
     ::= 9

END
```

Architecture d'ISM/OpenMaster

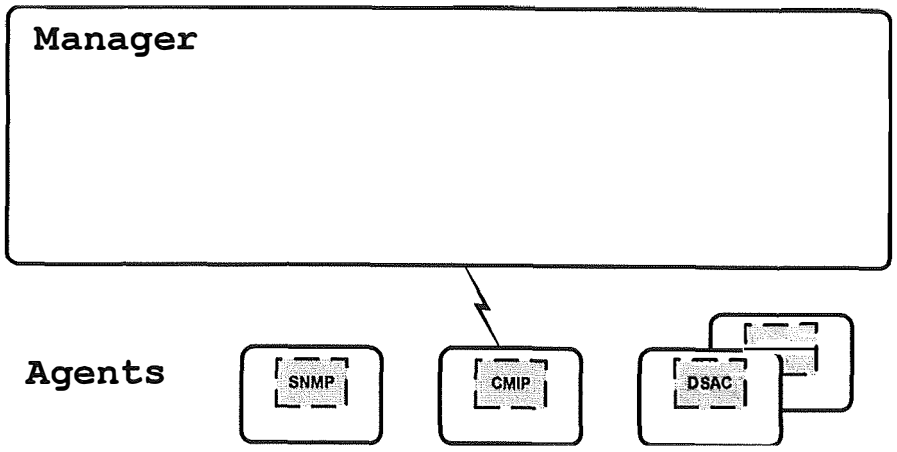

Figure A2- 1 : Gérer le monde réel

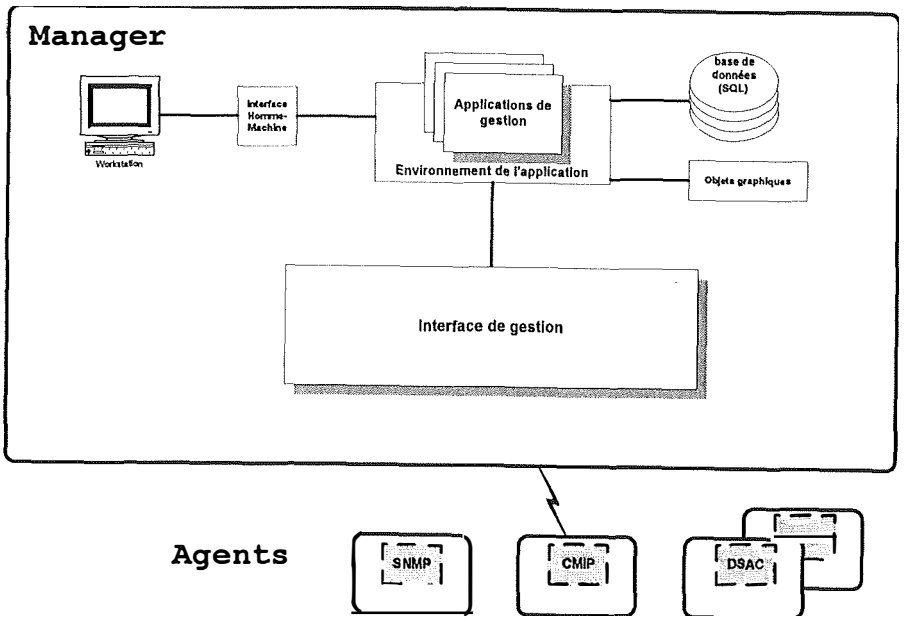

Figure A2- 2 : Les applications de gestion