Numéro d'identification :

ACADÉMIE DE MONTPELLIER

UNIVERSITÉ MONTPELLIER II

— SCIENCES ET TECHNIQUE DU LANGUEDOC —

THÈSE

présentée à l'Université des Sciences et Techniques du Languedoc
pour obtenir le diplôme de DOCTORAT

SPÉCIALITÉ	: **INFORMATIQUE**
Formation Doctorale	: **Informatique**
École Doctorale	: **Sciences Pour l'Ingénieur**

Application de techniques orientées agent en gestion et supervision de réseaux

par

Babak Esfandiari

Soutenue le 8 janvier 1997 devant le Jury composé de :

M. Jacques FERBER, Professeur, Université Montpellier II, ... Président
M. Guy PUJOLLE, Professeur, Université de Versailles, ... Rapporteur
M. Yves DEMAZEAU, Chargé de recherche, IMAG, ... Examinateur
M. Michel DAO, Ingénieur, CNET-France Télécom, ... Examinateur
M. Claude BOKSENBAUM, Professeur, Université Montpellier II, ... Examinateur
M. Philippe REITZ, Maître de conférences, Université Montpellier II, ... Examinateur
M. Ehoud AHRONOVITZ, Maître de conférences, Université Montpellier II, ... Examinateur
M. Joël QUINQUETON, Directeur de recherche, Université Montpellier II, ... Directeur de Thèse

Sed quis custodiet ipsos custodies ?
(Juvénal, 130-60 av. JC)

Le travail présenté dans ce rapport a été réalisé au Laboratoire d'Informatique, de Robotique et de Micro-éléctronique de Montpellier (LIRMM) au sein du département Acquisition et Représentation de Connaissances (ARC) dans le cadre du projet "Réseau futé" financé par le CNET-France Télécom.

Je tiens tout d'abord à remercier Monsieur Gaston CAMBON, directeur du LIRMM, Monsieur Michel CHEIN, responsable de l'Ecole doctorale et Monsieur Michel HABIB, responsable de la Formation doctorale pour m'avoir permis d'effectuer mon travail de thèse au LIRMM.

Je remercie chaleureusement Monsieur Joël QUINQUETON, Directeur de Recherche INRIA, d'avoir dirigé ma thèse avec une disponibilité et une sympathie incommensurables.

Je remercie les membres du jury qui m'ont fait l'honneur de juger ce travail de thèse :

- Monsieur Jacques FERBER, Professeur à l'Université Montpellier II, Président de mon jury de thèse,
- Messieurs Philippe JORRAND et Yves DEMAZEAU, respectivement Professeur et Chargé de Recherche à l'IMAG, d'une part, et
- Messieurs Guy PUJOLLE, Professeur à l'Université de Versailles, et Michel DAO, Ingénieur au CNET Issy-les-Moulineaux, d'autre part, pour avoir accepté de rapporter et juger ma thèse. Merci pour leurs critiques constructives,
- Monsieur Claude BOKSENBAUM, Professeur à l'Université Montpellier II, en qualité d'examinateur,
- Messieurs Ehoud AHRONOVITZ et Philippe REITZ, Maîtres de Conférence à l'Université Montpellier II, en qualité d'examinateurs, et pour les corrections pertinentes qu'ils ont apportées aux premières versions de ma thèse.

Merci à Albert Féroldi et Gilles Deflandre du CNET Lannion pour avoir supervisé avec rigueur le projet "Réseau Futé". Merci aussi à Michel Plu pour ses conseils capitaux, son enthousiasme et ses encouragements.

Merci aux autres membres du projet, Christophe DONY et Richard TERRAT, pour leurs conseils éclairés. N'oublions pas non plus Richard Nock, Laurence Grauby, Nicolas Philippe et Luc Yriarte pour leur participation active et précieuse.

Merci aux thésards du laboratoire, notamment Christophe Vial, Vincent Berry, Christian Capelle, Youssef Hamadi et Alain Serrano pour m'avoir aidé, conseillé et accordé leur amitié (et avec quelle patience !).

Enfin, je remercie Azar, Andy, Ghodsi, Nizar et surtout ma grand-mère adorée.

Table des matières

Chapitre 1

Introduction

Les réseaux représentent aujourd'hui un champ d'application idéal pour les techniques orientées “agent” : des agents “mobiles” sont utilisés pour la recherche d'information sur internet [EW94], le caractère par essence distribué des réseaux se prête à l'expérimentation de systèmes multi-agents, et des architectures à base d'agents sont utilisées pour le diagnostic de pannes dans le réseau.

Notre propos sera de tâcher de trouver des applications dans les domaines particuliers de la gestion et la supervision de réseaux. On présentera ainsi une utilisation des agents d'interface pour l'assistance à l'opérateur de supervision de réseau. On profitera également du réseau pour faire collaborer plusieurs agents d'interface pour améliorer leur performance. Mais on apprendra également à utiliser des concepts et des méthodes qui préexistaient dans les réseaux, telles les normes de gestion de réseau, qui nous serviront à mettre au point notre système multi-agent, qui au départ n'etait rien d'autre qu'une simulation d'une plateforme de gestion de réseau !

Nous pensons ainsi que les techniques et concepts utilisés dans les réseaux peuvent à leur tour constituer une bonne source d'inspiration pour les chercheurs en Intelligence Artificielle Distribuée.

Ce document débutera par un état de l'art dans les différents domaines abordés : les réseaux (et plus particulièrement la gestion de réseaux), la supervision en général, les agents... Nous utiliserons ensuite les enseignements tirés de cet état de l'art pour présenter un système d'assistance collaborative à la supervision de réseaux. Nous terminerons enfin par la description des expérimentations menées pour tester et valider notre système, et les résultats obtenus.

Le chapitre 2 a pour but de recenser toutes les notions qui seront utilisées dans les chapitres suivants. Il présente tout d'abord des généralités sur les réseaux, en s'attardant sur les normes de gestion de réseau. Il aborde ensuite la notion générale de supervision et ses formalismes adéquats, dont notamment celui de *chronique*. On s'intéressera ensuite aux agents, en commençant par la présentation de systèmes multi-agents utilisés en gestion et supervision de réseaux, et en abordant enfin des concepts qui nous ont semblé utiles dans la perspective de leur utilisation dans le domaine des réseaux. On peut citer notamment la notion d'*agent d'interface* qui consiste en un programme qui observe le comportement d'un utilisateur d'un système informatique quelconque afin d'y détecter des régularités qu'il pourra transformer en tâches automatiques après accord de l'utilisateur.

Les chapitres 3 et 4 constituent le coeur de cette thèse. Le chapitre 3 présente en effet notre proposition d'un système d'agent d'interface pour la supervision de réseau. Cet agent procède en apprenant le comportement de l'opérateur de supervision sous la forme de chroniques. Deux algorithmes d'apprentissage de chronique sont présentés : on utilisera l'un ou l'autre en fonction des contraintes imposées par l'environnement (pannes simultanées, multiplicité des causes d'une même panne,...). Enfin, pour améliorer et surtout accélérer la tâche d'apprentissage, nous montrerons comment un agent peut consulter ses "collègues" et tenir compte de leur avis.

Le chapitre 4 présente notre implémentation d'un prototype de simulation d'une plateforme de gestion de réseau, MAGENTA, au sein duquel nous avons intégré notre agent d'interface. MAGENTA utilise en partie les normes de gestion de réseau présentées dans le chapitre 2. La généricité de cet outil nous a même permis de l'utiliser comme une plateforme simplifiée de développement de systèmes multi-agents.

Enfin, le chapitre 5 présente les expérimentations menées et les résultats obtenus.

Chapitre 2

Etat de l'art

Une étude bibliographique assez détaillée est toujours intéressante à lire dans une thèse parce qu'elle donne une vue générale d'un domaine tout en rentrant suffisamment dans les détails pour montrer réellement l'intérêt des différents travaux abordés.

Plutôt donc que de faire un simple survol des travaux, on préfèrera dans ce qui suit les évoquer avec suffisamment de précision pour que le lecteur “ne reste pas sur sa faim”, et qu'il retire déjà, avant de consulter éventuellement lesdits travaux, une connaissance substantielle.

Le lecteur pressé pourra néanmoins faire l'économie de cette lecture et se reporter, à chaque fin de section, à un paragraphe récapitulatif qui tirera les conclusions des différents résumés et qui pointera le doigt sur les notions qui seront réellement utilisées par la suite dans ce document.

Notre étude bibliographique se décompose en plusieurs parties :

Nous commencerons par des généralités sur les réseaux [Tan91] [Puj]. Nous y verrons la décomposition en couches de l'ISO, et isolerons les couches susceptibles d'accueillir des techniques orientées “agent”. Ainsi nous étudierons les techniques de routage et surtout les normes de gestion et de supervision de réseaux.

Or le terme même de supervision est à définir avec précision. Qui plus est, la supervision soulève un certain nombre de problèmes dont notamment la prise en compte du temps. L'étude bibliographique se poursuivra donc avec une étude du concept général de supervision, et les méthodes et outils généralement employés.

Des travaux ont déjà été menés dans l'intégration de techniques d'Intelligence Artificielle Distribuée dans la gestion et la supervision de réseaux. Nous en verrons un certain nombre et nous discuterons de l'opportunité effective de l'utilisation de telles techniques suivant les cas, ce qui nous conduira naturellement à faire un état des lieux des techniques orientées agents qui nous semblent réellement appropriées à notre tâche.

2.1 Généralités sur les réseaux

2.1.1 Couches ISO

Pour réduire la complexité de la conception, les réseaux sont organisés en niveaux ou *couches*. Une couche offre des *services* aux couches plus hautes en leur épargnant les détails de la mise en oeuvre. La couche n d'une machine gère la communication avec la couche n d'une autre machine. Les règles et conventions utilisées pour cette communication sont appelées le *protocole* de couche n. Entre chaque paire de couches adjacentes, on trouve une *interface*. L'interface définit des opérateurs élémentaires et les services que la couche inférieure offre à la couche supérieure.

L'ISO (*International Standard Organization*) a élaboré une architecture de réseau à 7 couches connu sous le nom de *modèle de référence OSI (Open Systems Interconnection)*. Ce modèle permet de représenter de manière standard la connexion entre systèmes ouverts (fig. 2.1).

Voici brièvement le rôle de chaque couche :

1. la couche physique : elle s'occupe de la transmission des bits de façon brute sur un circuit de communication,
2. la couche liaison de données : son rôle est de masquer les caractéristiques physiques des entités du réseau et de détecter et corriger les erreurs,
3. la couche réseau : elle gère essentiellement le routage,
4. la couche transport : elle permet la communication entre deux applications, alors que les protocoles des couches inférieures gèrent les dialogues entre machines,

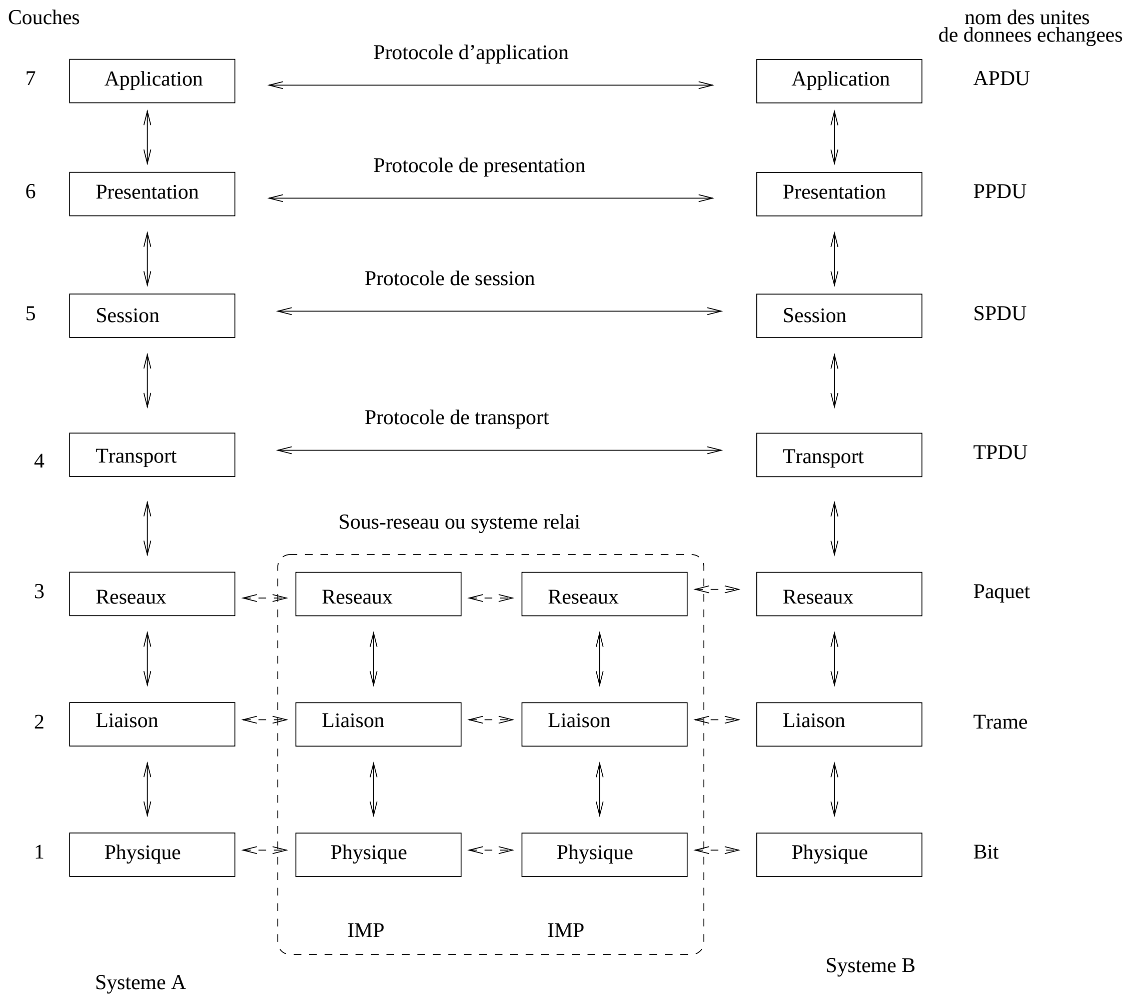


FIG. 2.1 - *Le modèle OSI*

5. la couche session : première des couches à offrir des services orientés utilisateur, elle offre notamment le service de session (login) et la gestion du dialogue,

6. la couche présentation : elle s'intéresse à la sémantique des informations. Elle fournit notamment des représentations standards des objets, et prend en charge la compression et la cryptographie,

7. la couche application : elle contient un ensemble de protocoles correspondant à des applications communes (messagerie, transfert de fichier, terminal virtuel...).

Les couches 1 et 2 étant trop proches du matériel, et les couches 5 et 6 n'étant pas implantées dans tous les réseaux, nous avons décidé de nous intéresser surtout au routage (implanté dans la couche 3 et dans une moindre mesure la couche 4), qui pose de véritables problèmes d'algorithmique distribué, et bien sûr la gestion de réseau, qui utilise des services de la couche 7.

2.1.2 Routage

Le problème du routage est celui du choix de la route optimale (en termes de délai d'acheminement, de la bonne répartition des messages dans le réseau, de nombre de "sauts" à effectuer...) pour acheminer un message de l'émetteur vers le récepteur. Les algorithmes sont implantés dans les noeuds de commutation (aussi appelés *IMP*). Ces algorithmes sont de deux types :

- non adaptatifs : ils sont mis en place dans les tables de routage des IMPs une fois pour toute. C'est la solution la plus simple, mais elle n'est possible que si le trafic est relativement stable, et si le réseau n'évolue pas souvent ;
- adaptatifs : les décisions de routage peuvent être modifiées en fonction de la variation de la topologie et du trafic réel du réseau.

2.1.2.1 Les algorithmes non adaptatifs

2.1.2.1.1 L'algorithme du plus court chemin On suppose qu'on dispose de la topologie complète du réseau, que l'on modélise par un graphe où les noeuds sont les IMPs et les arcs sont les lignes de communication. Les arcs sont valués grâce à une fonction de calcul de distance (distance géographique, délais de transmission, ...). Il s'agit alors de calculer le plus court chemin entre deux noeuds du graphe. L'algorithme classiquement utilisé est celui de Dijkstra [Dij].

2.1.2.1.2 Le routage multichemin Au lieu de ne proposer qu'un seul chemin, on peut proposer, pour chaque destination, plusieurs chemins possibles, pondérés selon leur qualité. Le choix du chemin se fera selon une probabilité proportionnelle à la pondération. Ainsi on pourra mieux répartir la charge du réseau. Cela permet également d'augmenter la tolérance aux pannes, à condition que les différents chemins proposés soient disjoints. Une façon simple de calculer des chemins disjoints est de calculer d'abord le plus court chemin, puis d'enlever du graphe les noeuds et les arcs correspondant et de réitérer le calcul.

2.1.2.2 Les algorithmes adaptatifs

2.1.2.2.1 Le routage centralisé On suppose qu'on dispose quelque part dans le réseau d'un ordinateur central qui reçoit périodiquement des informations sur le réseau (trafic, longueur des files d'attentes dans les commutateurs, ...) et qui calcule les chemins optimaux en fonction de ces informations, afin de mettre à jour dynamiquement les tables de routage.

Cela pose un certain nombre de problèmes :

- la tolérance aux pannes de l'ordinateur central (il faudrait utiliser un ordinateur de secours) ;
- la mise à jour "simultanée" des tables de routage : il ne faudrait pas engendrer temporairement des incohérences ;
- le trafic accru à la proximité de l'ordinateur central

Dans le réseau TYMNET, les IMPs envoient périodiquement les informations à l'ordinateur central. A chaque demande de connexion, l'hôte demande la meilleure route à l'ordinateur, qui calcule cette route et envoie un "paquet" d'aiguillage" pour permettre la mise à jour des tables de routage à l'initialisation de la connexion.

2.1.2.2.2 Le routage local Ici les IMPs décident localement, en fonction des informations qu'ils possèdent, du routage sans consulter les autres IMPs. Par exemple dans l'algorithme *hot potato* l'IMP se "débarrasse" d'un paquet entrant en le plaçant dans la ligne de sortie ayant la file d'attente la plus courte, sans se préoccuper de l'endroit où elle conduit.

Une variante consiste à combiner le routage statique avec le *hot potato* : on utilise le meilleur choix statique, sauf si la file d'attente est trop longue. Il existe d'autres variantes encore...

Dans le cas du *backward learning*, chaque paquet contient l'identification de l'IMP source, et à chaque saut on incrémente un compteur. Chaque IMP peut ainsi savoir quelle distance le sépare de la source, et si cette distance est inférieure à celle du chemin connu, l'IMP modifie sa table de routage à destination de la source. Au bout d'un certain temps, on espère découvrir le meilleur chemin pour relier tout IMP. Le problème de cette approche est qu'en cas de panne d'une ligne, il n'y a aucun moyen d'en tenir compte pour mettre à jour les tables. Une solution consiste à remttre à zéro de temps en temps ce que les IMPs savent.

Le *routage delta* est un panachage possible du routage centralisé et du routage local : les IMPs envoient périodiquement le coût de chaque ligne à un ordinateur central, et celui-ci décide des k meilleurs chemins dont la différence de coût par rapport à l'optimum n'excède pas une certaine valeur δ. L'IMP peut alors choisir n'importe lequel de ces chemins. Si δ tend vers 0, on se rapproche d'un choix unique, donc du routage centralisé. Si δ tend vers l'infini, tous les chemins deviennent équivalents, donc la décision de routage devient purement locale. Or en paramétrant correctement δ et k, on peut obtenir des compromis très satisfaisants.

2.1.2.2.3 L'inondation Dans cette méthode, chaque paquet entrant est réémis sur chaque ligne de sortie, exceptée la ligne d'arrivée. Pour éviter de générer un nombre illimité de paquets, une méthode consiste à inclure un compteur de sauts dans le paquet. Ce compteur a pour valeur initiale la longueur du chemin ou par défaut le "diamètre" du réseau (par exemple le nombre total de noeuds). A chaque saut le compteur est décrémenté. Si le compteur atteint zéro, on élimine le paquet. La méthode d'inondation, a priori très coûteuse, est intéressante surtout pour sa très grande tolérance aux pannes, et son utilisation pour effectuer de la diffusion de messages.

2.1.2.2.4 Le routage distribué Dans cette méthode, chaque IMP envoie à ses voisins la distance (nombre de sauts, taille des files d'attentes, ...) qui le sépare de toutes les autres IMPs. Chaque IMP, connaissant la distance qui le sépare de ses voisins, peut alors calculer pour chaque IMP la distance optimale, et l'IMP par lequel il faudra passer.

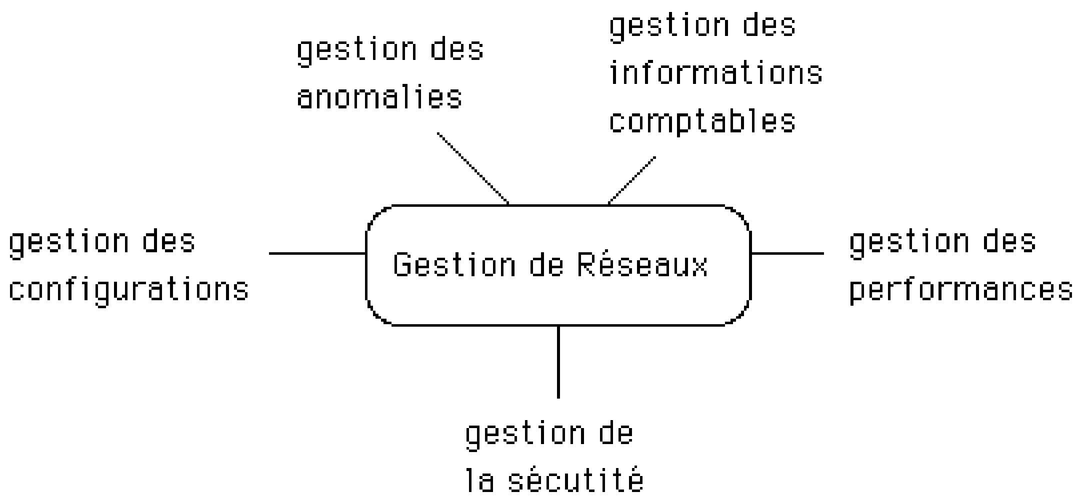


FIG. 2.2 - *Les fonctions de la gestion de réseaux selon l'ISO*

2.1.3 La gestion de réseaux

Pourquoi gérer un réseau ? L'objet de la gestion de réseaux est de rendre le système opérationnel tout en offrant la meilleure qualité de service aux utilisateurs et de le faire évoluer au mieux. Selon l'ISO [LC92] [CC93], gérer un réseau nécessite cinq fonctionnalités (fig.2.2):

- La Gestion des Anomalies permet de surveiller le fonctionnement du réseau en permanence, de détecter les problèmes, les identifier et les corriger.

- La Gestion de Configurations permet de paramétrer, initialiser et rendre opérationnel le réseau. Elle surveille les différents changements d'état et décrit de façon non ambiguë tous les éléments constitutifs de son architecture et de son mode de fonctionnement.

- La Gestion des Performances permet de collecter les informations sur le fonctionnement du système, de les journaliser, de quantifier la qualité de service et de réguler le trafic en se basant sur les données précédentes.

- La Gestion des Informations Comptables permet de surveiller pour chaque ressource ses charges et limites d'utilisation, de comptabiliser les ressources utilisées et leur coût.

- La Gestion de la Sécurité permet la mise en oeuvre de mécanismes de sécurité (contrôle d'accés, integrité...) et de confidentialité.

De nouvelles normes ISO visent à créer de meilleures conditions de gestion des réseaux, en adoptant notamment une approche orientée objet dans la description des différents composants (formalisme G.D.M.O.), et en normalisant les protocoles de communication entre entités de gestion de réseau (gestionnaires et agent). Cette approche sera détaillée dans le paragraphe 2.1.3.1

Or la plupart des réseaux internationaux ont déjà développé leurs propres protocoles et outils de gestion. Nous verrons dans le paragraphe 2.1.3.2 comment se situe Internet par rapport aux normes.

L'hétérogénéité des réseaux de télécommunication a conduit le CCITT à définir un Réseau de Gestion des Télécommunications (R.G.T.) qui permet la mise en relation de réseaux et le développement de services indépendamment du matériel (2.1.3.3).

2.1.3.1 La normalisation de la gestion de réseau OSI : la gestion système

2.1.3.1.1 La notion d'objet L'ISO a défini un formalisme orienté objet de description : G.D.M.O. (Guideline for the Definition of Managed Objects) [ISO90b].

GDMO permet de décrire les informations de gestion en terme d'objets gérés. On peut définir un objet géré comme une abstraction sur une ou plusieurs ressources physiques ou logiques du système géré.

Chaque objet représente les propriétés et les caractéristiques d'une ressource manipulable dans le but de la gérer. Les objets ayant les mêmes propriétés sont regroupés dans des classes appartenant à un arbre d'héritage.

Chaque objet géré posséde :

- des attributs : propriétés sur les valeurs desquels on peut exécuter des opérations de consultation ou de modification.
- des actions : opérations qui peuvent être exécutées sur ces attributs.
- des notifications : messages que l'objet peut émettre pour signaler qu'un événement extérieur s'est produit.

– des comportements : comportement de l'objet face à un événement extérieur.

Pour décrire ces formats, G.D.M.O utilise des "templates" (formats prédéfinis) qui s'appuient sur le langage de description de type ASN1 (Abstract Syntax Notation 1). Les templates permettent uniquement de décrire la structure des objets et non leur comportement ; ce dernier est laissé au programmeur. Dans le paragraphe 4.1.3, on peut voir un exemple de description au format GDMO.

2.1.3.1.2 Le nommage des objets (*name-binding*) Chaque objet se situe de plus dans une hiérarchie d'objet, appelée arbre de nommage, par rapport à un objet supérieur dont il est le subordonné. La racine de cet arbre est l'objet NULL.

Les objets sont organisés en arborescence, ce qui permet de les nommer de façon unique, de la même manière que les fichiers d'un système arborescent.

Ainsi, un objet possède un nom relatif qui l'identifie des autres subordonnés de son objet supérieur et un nom absolu qui le distingue des autres objets de l'arbre. Ce nom absolu est obtenu par concaténation de tous les noms relatifs des objets rencontrés lors du parcours de l'arborescence de la racine jusqu'à l'objet.

2.1.3.1.3 La base de gestion des informations (MIB) On définit la MIB comme la base conceptuelle des informations de gestion [ISO90c]. Elle représente le modèle des objets du système géré et permet de stocker les instances des classes décrites par G.D.M.O.

On peut notamment accéder à cette base par des services de gestion définis au niveau de la couche application de l'ISO (dont C.M.I.S.E. (Common Management Information Service Element)).

2.1.3.1.4 Le modèle gestionnaire - agent Entre deux systèmes ouverts, deux processus de gestion-système interagissent selon le modèle client - serveur.

On définit comme processus gestionnaire (client) le processus qui utilise le service et comme processus agent (serveur) le processus qui fournit le service.

Les services de gestion correspondent à des services d'exécution d'opérations sur les objets de l'agent ou à des services d'émission de notifications d'événements sur un objet.

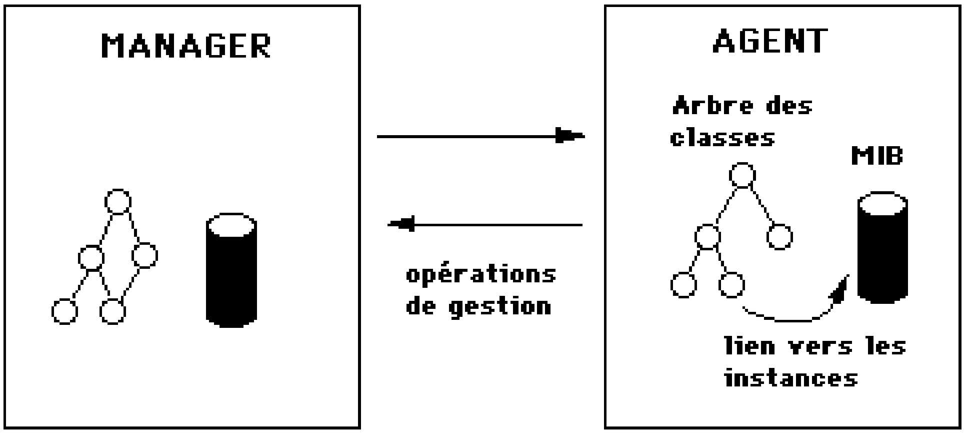


FIG. 2.3 - *Le modèle gestionnaire (manager) - agent*

Nous pouvons résumer l'interaction entre ces deux processus par la figure 2.3.

Il est évident que cette terminologie n'est pas fixe, un agent peut devenir un gestionnaire s'il veut envoyer des messages à d'autres objets que les siens.

2.1.3.1.5 La communication gestionnaire - agent La communication de gestion entre un gestionnaire et un agent s'effectue grâce aux services CMISE [ISO90a]. Ces éléments de service offrent au processus gestionnaire de réaliser sur un ou plusieurs objets gérés des opérations de gestion (création et destruction d'objets (primitives CREATE et DELETE), consultation et modification d'attributs et déclenchement d'actions (GET, SET et ACTION)) et de rapporter des notifications (primitive EVENT-REPORT).

La sélection d'un objet géré (sélection (ou *scope*) et filtrage sur la présence et la valeur d'attributs) présente une analogie avec la requête SELECT de SQL. Ainsi on peut par exemple sélectionner tous les objets ayant le même père (voire l'ancêtre) dans l'arbre de contenance, et dont la valeur d'un certain attribut est supérieure, inférieure ou égale à une certaine valeur.

Le protocole de communication associé à ce groupe de service est le Common Management Information Protocol (CMIP). Il fournit un échange synchrone (avec connexion) sous forme de requêtes et/ou de requêtes/réponses.

CMISE fournit uniquement des services de gestion et fait appel à des éléments de services sous-jacents : ACSE pour assurer l'établissement et la libération de l'association entre le gestionnaire et l'agent et ROSE pour le transfert des données à distance.

2.1.3.1.6 Les fonctions de gestion-système (*SMF*) Les fonctions de gestion-système (ou *System Management Functions*) définissent un ensemble de services permettant de satisfaire un besoin particulier. Celles-ci sont modélisées en terme de propriété, de classes d'objets GDMO et ont leur projection sur CMISE. Les fonctions normalisées actuellement sont :

- La fonction de gestion d'objet :

  Cette fonction définit l'ensemble des services de manipulation et de notification pour les objets gérés. Elle permet de créer, de supprimer des objets, de lire, modifier des valeurs d'attributs conformément aux requêtes CMISE reçues.

- La fonction de gestion de relation :

  Les objets gérés ne sont pas toujours indépendants les uns des autres. Le comportement d'un objet (manière dont l'objet va exécuter ses opérations) peut être influencé par un autre objet. On parle dans ce cas de relation entre ces deux objets. Cette fonction permet d'exprimer, modifier les relations entre objets.

  Afin de développer la gestion des relations, l'ISO a décidé de créer un modèle général de relations qui définirait un formalisme pour la spécification de relations entre objets. Ce modèle est en cours d'élaboration [ISO95].

- La fonction de gestion des états :

  Cette fonction permet de contrôler l'état d'un objet concernant notamment son utilisation, sa disponibilité. Un objet dispose de plusieurs attributs d'état ayant différentes valeurs possibles. Selon les opérations exécutées sur l'objet son état pourra être modifié. On définit alors des types de notification pour informer le gestionnaire des changements d'état intervenus.

- La fonction de rapport d'alarme :

  Cette fonction normalise les comptes-rendus d'erreur (déviation du fonctionnement normal d'un système) et de faute (cause d'un mauvais fonctionnement). Elle permet de gérer l'envoi et le transfert des informations d'alarme.

Il peut s'agir d'alarmes de communication, de qualité de service, d'équipement de traitement (logiciel) ou d'environnement. Le compte-rendu comporte des informations sur la cause probable de l'erreur et le degré de sévérité associé. Eventuellement, il peut comporter des indications sur la réparation.

- La fonction de rapport d'événement :

  Cette fonction permet à un gestionnaire de contrôler la transmission de notifications d'événements indépendamment de la définition des objets qui les émettent. Les notifications d'événement sont transmises à des discriminateurs de transmission d'événement. Ceux-ci les analysent et déterminent si elles doivent donner lieu à des comptes-rendus d'événement pour une certaine destination.

- La fonction de rapport d'alarme de sécurité :

  Cette fonction définit un service de compte-rendu d'alarme relative à la sécurité c'est-à-dire concernant les mauvais fonctionnements, attaques, violations des services et mécanismes de sécurité. Le compte-rendu d'alarme fournit également le degré de sévérité de l'alarme.

- La fonction de gestion de journaux :

  Cette fonction permet de définir des services de gestion de ressources servant à mémoriser des notifications d'événement. Ces ressources sont modélisées par les classes d'objets journal (log) et enregistrement (log record). Ces services doivent permettre d'initialiser et supprimer des journaux, d'en modéliser les caractéristiques, de choisir les informations à mémoriser et leur ordonnancement, consulter et détruire des enregistrements, veiller à la qualité de mémorisation.

- La fonction de test :

  Cette fonction définit des services de test pour le diagnostic de défaut et pour obtenir des données de performance. L'initialisation, la suppression, la reprise et la terminaison d'un test doivent pouvoir être déclenchées selon le cas par un gestionnaire distant ou un événement spécifié.

  Un test peut impliquer plusieurs systèmes. Le module de test définit un conducteur de test (gestionnaire) et un exécuteur de test (agent).

L'exécuteur contient des objets à tester (MOT) et des objets de test (OT) qui effectueront le contrôle et la supervision du test.

Le conducteur active le test en envoyant des commandes de test et de supervision.

Les catégories de test sont nombreuses; elles sont définies dans la norme CCITT x737. On peut trouver notamment des tests de connectivité, d'intégrité de données, de protocole ...

2.1.3.2 La gestion des réseaux TCP/IP

La communauté Internet a normalisé et diffuse actuellement deux protocoles de gestion de réseau. Le premier, SNMP, est un protocole simple ; le second, CMOT, tend à conformer le système de gestion TCP/IP aux normes ISO.

La gestion OSI et celle des réseaux TCP/IP utilisent le même vocabulaire mais y affectent souvent une sémantique différente.

Ces deux types de gestion diffèrent d'une part quant à la structure des informations traitées, et d'autre part par le type de communication liant deux entités de gestion.

2.1.3.2.1 Architecture L'architecture de la gestion des réseaux TCP/IP repose sur un ensemble de stations de gestion (stations de travail dédiées à la gestion) et des composants du réseau.

Les stations de gestion supportent le protocole et les opérations de gestion ; elles surveillent et contrôlent les composants du réseau en demandant périodiquement à chaque noeud des informations.

Les composants ou noeuds du réseau peuvent être considérés comme des agents (exécuteur de l'opération de gestion). Les composants représentent l'ensemble des objets du réseau à gérer : cela peut être des machines-hôtes, des stations, des passerelles, des ponts, des modems...

L'interaction entre une station et un composant (noeud) se fait sur le modèle client - serveur (fig. 2.4).

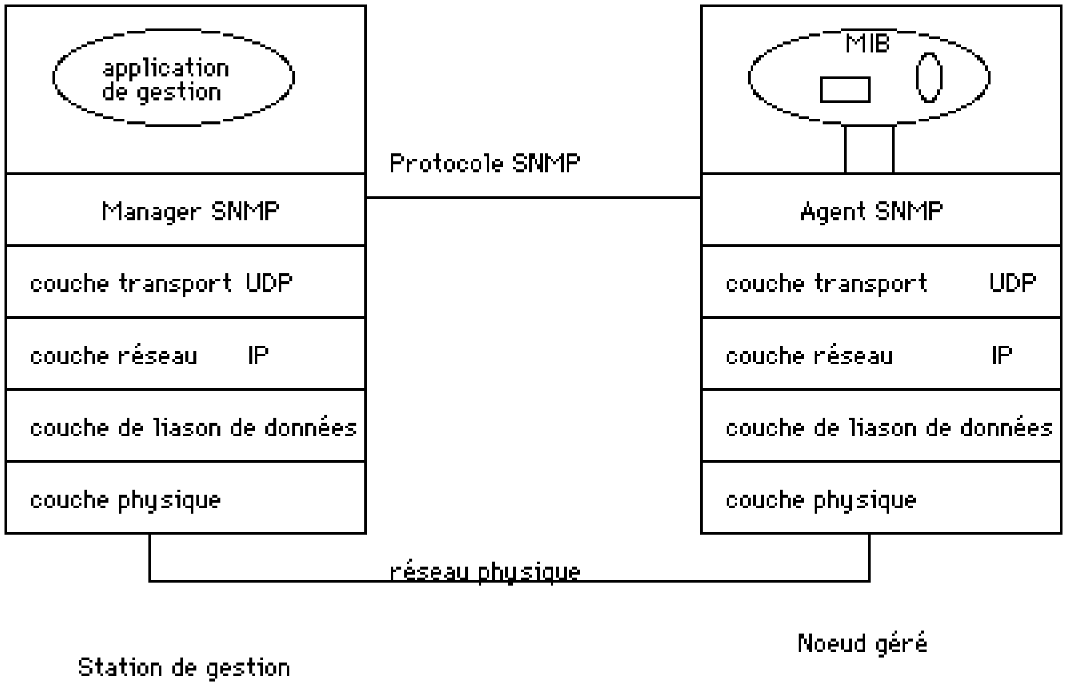


FIG. 2.4 - *La gestion de Réseaux dans Internet*

2.1.3.2.2 La notion d'objet La notion d'objet est totalement différente de celle de la gestion de la gestion OSI. Elle n'est pas liée au paradigme objet. Un objet est une variable informatique classique.

Le nom des objets, défini de façon unique, indique la position de l'objet dans l'arbre d'enregistrement global (arbre où sont hierarchisés l'ensemble des types d'information normalisés). En ce qui concerne les objets de gestion TCP/IP, ils se trouvent sous root/iso/org/dod/internet/mgnt/mib2.

Les instances de ces objets sont nommées en ajoutant au nom de l'objet un sous-identifiant d'instance.

2.1.3.2.3 La Base d'Informations de Gestion La base d'informations de gestion ou MIB est une zone virtuelle de stockage des objets gérés.

La MIB publique contient l'ensemble des objets normalisés par la communauté internet (171 objets regroupés en 10 sous-ensembles).

Les autre objets doivent être définis dans la MIB propriétaire qui se trouve dans l'arbre d'enregistrement global sous root/iso/org/dod/internet/private/entreprise. Cette MIB est accessible par le protocole SNMP.

2.1.3.2.4 Le protocole SNMP Ce protocole fournit une communication asynchrone sous la forme requêtes et/ou requêtes-réponses. Ce protocole propose 6 services. Il est développé sous le protocole UDP et utilise le mode datagramme.

Ses services sont la lecture, la lecture incrémentale, l'incrémentation, l'envoi de réponses et l'envoi de notifications.

Ce protocole est simple ; s'il n'est pas conforme à la normalisation ISO, il a l'avantage d'être opérationnel. Il n'offre néanmoins pas la richesse de CMIP et GDMO du fait de la représentation “à plat” des objets gérés.

2.1.3.2.5 Le protocole CMOT Ce protocole a pour objet de conformer la gestion des réseaux TCP/IP aux normes ISO. L'architecture envisagée pour le CMOT est de rajouter au-dessus de TCP et UDP une couche présentation allégée : Lightweight Presentation Protocole, les services ACSE, ROSE , les services CMISE et le protocole CMIP.

Les objets gérés devront alors s'appuyer sur le paradigme objet.

2.1.3.3 Le Réseau de Gestion de Télécommunication

Le concept de Réseau de Gestion de Télécommunication (RGT) a été élaboré par le CCITT (Recommandation M30) pour définir un réseau de gestion de réseaux de télécommunications. La notion de réseau s'applique non seulement à décrire le système à gérer (réseau de communication) mais également le système gérant (réseau de gestion).

L'objectif du RGT est de fournir un moyen de dialoguer avec des équipements, de structurer les applications de gestion de réseau et de spécifier les interfaces entre ces applications. Il englobe donc tous les aspects de gestion de réseaux.

Le Réseau de gestion de réseaux de télécommunication peut être représenté par la figure 2.5.

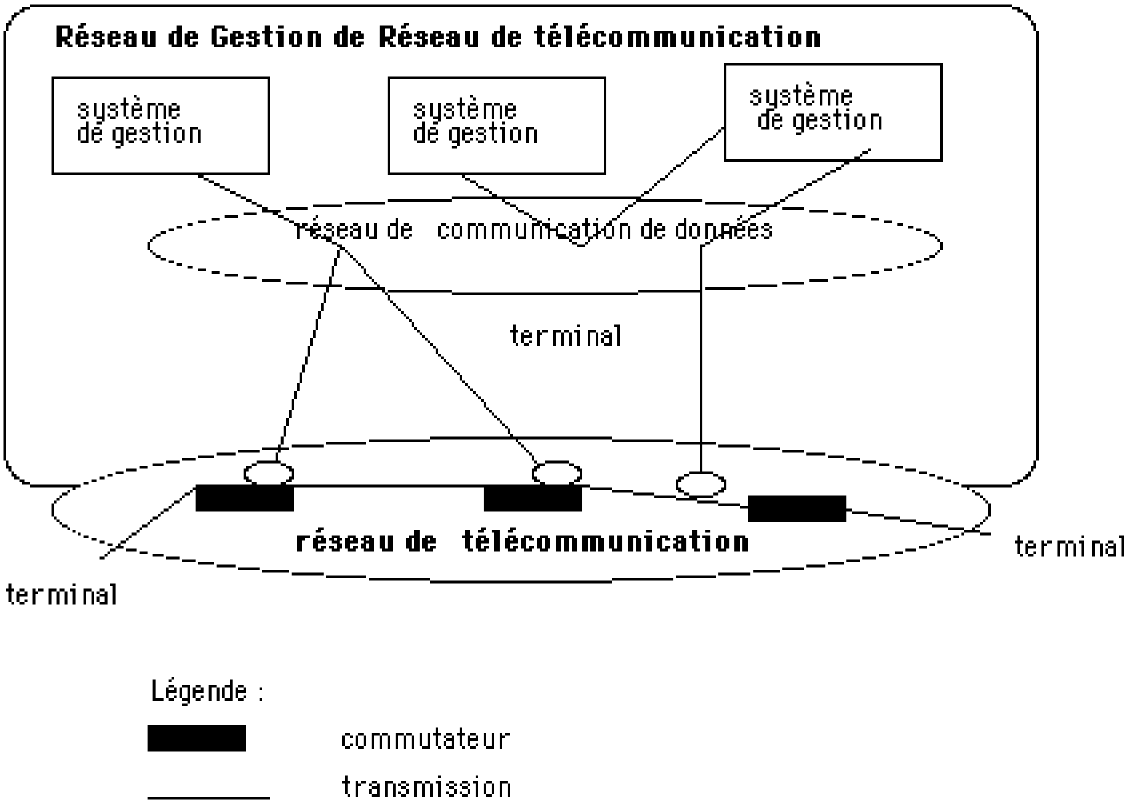


FIG. 2.5 - *Le RGT*

Le RGT vise à gérer de façon transparente des réseaux différents tels que les réseaux téléphoniques (RT, RNIS), les réseaux de transmission (multiplexeur...) et les réseaux privés (PABX, RLE, RGD).

2.1.3.3.1 Architecture fonctionnelle Le RGT identifie six types de blocs fonctionnels : les blocs de systèmes de gestion (OSF), de communication (DCF), de médiation (MF), d'élément de réseau (NEF), d'accès et d'adaptation d'interfaces non normalisées (QAF).

Entre deux blocs, se trouve un point de référence. Le point Q3 est l'interface entre OSF et les points OSF, MF, NEF, QAF, le point Qx entre le MF et les NEF, QAF, MF, le point X entre deux RGT. Ces deux interfaces Q et X sont définies à l'aide des normes de gestion OSI décrites précédemment.

La figure 2.6 montre l'architecture d'un RGT.

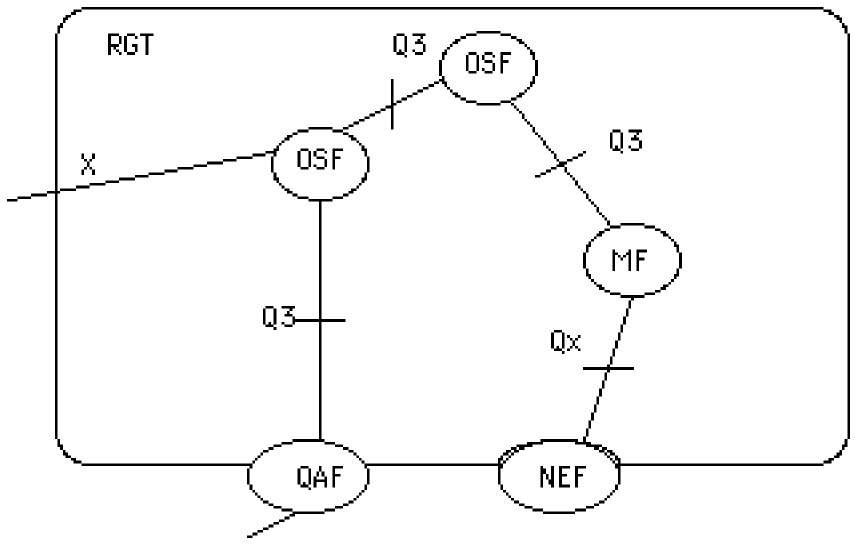


FIG. 2.6 - *L'architecture d'un RGT*

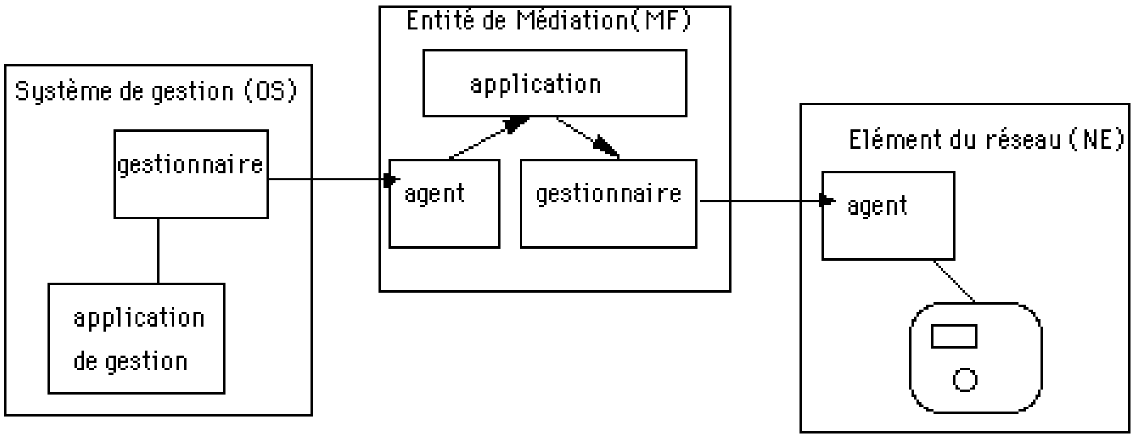


FIG. 2.7 - *Le modèle gestionnaire-agent dans le RGT*

2.1.3.3.2 Architecture de communication Les notions de gestionnaire et agent peuvent être appliquées au RGT. La figure 2.7 montre le modèle gestionnaire - agent appliqué au RGT.

Le système de gestion envoie des messages de gestion. L'entité de médiation adapte, filtre, convertit les informations et les protocoles entre systèmes de gestion et éléments du réseau.

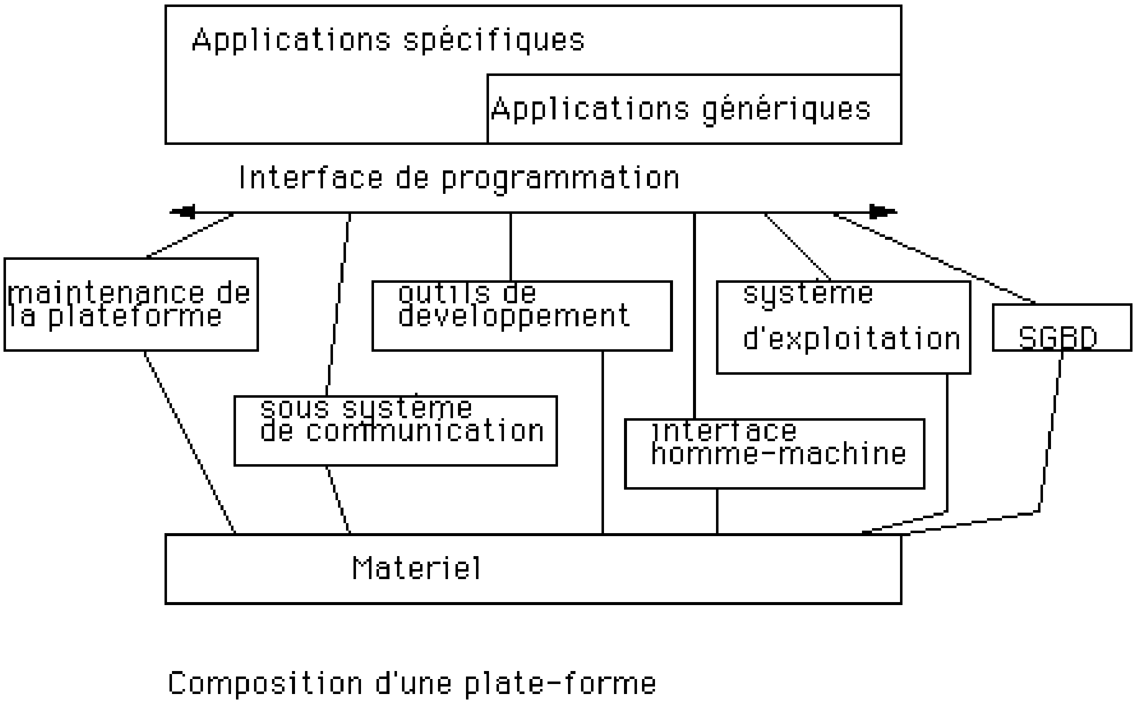


Fig. 2.8 -

Les échanges se font à l'aide des services CMISE (Common Management Information Service Element) si le volume est faible ou à l'aide de FTAM (File Transfer, Access Management) s'il s'agit de transfert de fichier.

2.1.3.4 Les plateformes de gestion de réseaux

([ARP92]) : L'appellation de plateforme de gestion de réseaux désigne un environnement logiciel et matériel de support d'applications de gestion de réseaux grâce à des interfaces définies. Une vraie plateforme se compose d'une gamme de machines physiques mono ou multi-constructeurs, d'un système d'exploitation autorisant la portabilité sur diverses machines physiques, d'un système de gestion de bases de données (idéalement orienté objet), d'un sous-système de communication (protocoles OSI), d'une interface Homme-Machine, d'un système de maintenance de plate-forme, d'un environnement de développement avec des interfaces de programmation normalisées, et d'un ensemble d'applications génériques. La figure 2.8 résume cela.

2.1.4 Que retenir ?

- Routage et agents réactifs : on constate que des techniques distribuées sont déjà implantées dans les routeurs. Ces techniques peuvent par ailleurs être des sources d'inspiration pour les concepteurs de systèmes multi-agents réactifs.
- On dispose d'un langage de représentation des ressources à gérer (GDMO) ainsi que d'un langage d'interaction (CMIS) très élaborés. Pourquoi ne pas les utiliser dans notre futur système multi-agent ? SNMP par contre, s'il s'avère suffisant pour des tâches élémentaires de gestion, ne peut pas prétendre à un statut de langage d'interaction (ce n'est de toute façon pas son but).
- La mise en place du RGT est la preuve qu'on ressent le besoin de mettre en contact plusieurs stations de gestion de réseau. Le RGT n'est pas encore en fonction, et c'est à nous concepteurs d'en trouver les futures applications.

La section suivante présente le concept général de supervision afin de le définir, de montrer les problèmes qu'il pose et de recenser les formalismes qu'il utilise.

2.2 La supervision

Le projet inter-PRC "gestion de l'évolutif et de l'incertain dans une base de connaissances" s'est intéressé à une étude formelle de la problématique de la supervision en général [cdpiP95]. Il a notamment décomposé la supervision en une série de sous-problèmes, et a étudié des formalismes de représentation de connaissances adéquates pour tous ces sous-problèmes, qui ont pour point commun la nécessité de gérer des aspects temporels dûs à l'évolutivité du système supervisé. Tous les formalismes étudiés reposent sur le principe d'un graphe de transitions entre états, mais avec des niveaux d'abstraction ou des capacités d'expression différents.

Tout d'abord il convient d'adopter un modèle pour décrire le système physique qui est supervisé (à ne pas confondre avec le modèle décisionnel de l'opérateur de supervision). On peut adopter différents types de modèles : un modèle peut être structurel (décrire une certaine topographie), fonctionnel, comportemental, causal,... On peut aussi classer les modèles suivant les techniques mathématiques de représentation auxquelles ils font appel. Les modèles peuvent alors être **continus** (le temps et les variables sont continus), **échantillonnés** (variables continues, temps discret), **discrétisés** (variables discrètes, temps continu), ou **à événements discrets** (temps et variables discrets) suivant la méthode d'observation.

Dans le cas des modèles à événements discrets, l'outil mathématique le plus utilisé est l'automate d'états finis. Le système étudié peut être représenté par un graphe de tous les états possibles, les arcs représentant les événements (ou ensembles d'événements) permettant de passer d'un état à un autre. Notons que comme en général on n'observe pas tous les événements et qu'un état observé peut correspondre à des états réels différents, le graphe des **observables** est un sous-graphe relativement restreint du graphe d'états "réels". Un chemin dans le graphe d'états est appelé **scénario**.

En général il est impossible de représenter entièrement le graphe d'états. Il est plus satisfaisant d'utiliser des représentations d'un plus haut degré d'abstraction qui permettent une expressivité et une compacité plus grandes. La contrepartie en est que certains traitements tels que la mise à jour sont rendus plus difficiles : on se heurte notamment au "frame problem" (persistance du décor). Il s'agit de savoir répercuter les changements et seulement les changements dûs à l'occurence d'un événement. Ce problème se pose par exemple dans le cas d'une description en logique du premier ordre. L'expression suivante :

```
peindre-en-rouge (taxi (X) ) -> rouge ((taxi (X))
```

si elle permet de décrire le changement indépendamment de la couleur initiale du taxi, elle implique par contre d'enlever la couleur précédente du taxi de la base de connaissances.

Une description en logique linéaire ou par réseaux de Petri de haut niveau de type :

```
|- !((bleu(X) + rouge(X) + noir(X)) -o rouge(X))
```

nécessite en revanche de décrire de façon extensive tous les changements possibles, ce qui possède l'avantage de simplifier la mise à jour, puisque les faits sont considérés comme des "ressources consommables", et sont donc "consommés" lors du changement, donnant lieu à l'apparition de nouveaux faits (la partie droite de l'implication).

Bien entendu une description par réseau de Petri ordinaire ou directement par une machine à ensemble fini d'états est encore plus proche du graphe d'états sous-jacent, mais nécessite du coup une énumération exhaustive de tous les états et changements possibles.

Une fois le formalisme adéquat adopté, le système de supervision l'utilise dans ses différents modules. En général un tel système est doté de :

– un module de suivi d'évolution ; il peut s'agir d'un programme de reconnaissance de scénarios, correspondant soit au comportement nominal du système (on cherchera alors d'éventuels écarts par rapport à ce comportement), soit aux comportements perturbés, qu'il faut détecter.

– un module de diagnostic, qui doit déterminer les causes des anomalies.

– un module de prédiction, qui s'appuie sur sa connaissance du modèle et de l'état courant pour prévoir l'évolution du système.

– un module de décision, qui choisit, en interaction avec l'opérateur et avec les autres modules, des actions à entreprendre pour remettre le système dans un mode de fonctionnement normal.

Un programme de reconnaissance de scénario a le mérite d'être non seulement capable d'effectuer un suivi d'évolution, mais aussi dans une moindre mesure de faire du diagnostic et de la décision : à un scénario de comportements perturbés on peut associer une étiquette décrivant le type de panne et l'action corrective à effectuer. La prédiction est également possible grâce à la consultation d'instances encore incomplètes de scénarios en cours.

Une technique possible de reconnaissance de scénario consiste à se ramener au problème de reconnaissance de motifs dans une chaîne de caractères : les modèles et l'évolution du système physique sont alors des séquences linéaires d'événements. L'algorithme de Knuth, Morris et Pratt [KMP77] permet la reconnaissance "à la volée" de *facteurs* (sous-chaînes à caractères contigus) et donc de scénarios *exclusifs* (incompatibles avec d'autres événements de même type durant son déroulement), totalement *ordonnés*, et avec des contraintes temporelles portant uniquement sur des événements contigus.

Le paragraphe suivant expose la notion de chronique et détaille la reconnaissance de scénarios par suivi de chroniques.

2.2.1 Supervision et raisonnement sur le temps : la notion de chronique

La représentation explicite du temps [Gha94] s'avère nécessaire dès qu'il s'agit de raisonner sur des événements, des actions et des changements. Ainsi, la supervision et la planification sont des domaines qui doivent adopter des formalismes qui permettent de tels raisonnements.

Les approches de type "numérique" pour le raisonnement temporel utilisent des techniques de recherche opérationnelle, où les contraintes temporelles sont traduites en termes d'équations ou d'inéquations.

Il existe plusieurs approches de type "logique", dont notamment les logiques modales, qui peuvent avoir une sémantique temporelle. On peut syntaxiquement créer des modalités temporelles *ad hoc* à partir des modalités de base. Mais ces types de logiques posent des problèmes théoriques (problème de complétude de la logique temporelle du 1er ordre) et de complexité algorithmique.

La logique réifiée est une autre approche logique possible : la réification consiste à transformer un prédicat en un terme d'un prédicat plus général. En logique temporelle réifiée du 1er ordre, on dispose de deux prédicats particuliers :

- l'assertion (*hold*) permet d'exprimer la persistance de la valeur d'un attribut dans un intervalle de temps. Ex:

  hold (position (robot1, docking-site), (t5, t6))

- l'événement (*event*) exprime le changement de la valeur d'un attribut à un instant donné. Ex:

  event (state (switch) : (off, on), t8)

  la valeur de *state (switch)* était "off" avant t8, et devient "on" après t8.

Le système IxTeT est un "raisonneur temporel" : il permet de recevoir et de stocker des données temporelles, de maintenir la cohérence de la base de données temporelle ainsi constituée, de répondre à des requêtes concernant cette base et enfin de prédire l'évolution de l'environnement concerné. A ces fins, IxTeT utilise la logique réifiée, et obtient une complexité algorithmique raisonnable dans le maintien de cohérence de la base de données temporelle. Le module de supervision d'IxteT utilise quant à lui la notion de chronique qui repose également sur la logique temporelle réifiée. Un modèle de chronique est un ensemble de motifs d'événements, de contraintes de temps entre ces événements, dans un contexte spécifié par des assertions. Si les événements observés correspondent aux motifs et respectent le contexte et les contraintes, alors une instance de chronique se produit et une action correspondant à la détection de la chronique est déclenchée.

Exemple :

Chronicle `RobotLoadMachine` {
event `(Robot : (outRoom, inRoom), e1);`
event `(Robot : (inRoom, outRoom), e4);`
event `(MachineInput : (Unloaded, Loaded), e2);`
event `(Machine : (Stopped, Running), e3);`
`e1 < e2;`
`1'<=e3 - e2 <= 6';`
`3'<=e4 - e2 <= 5';`
hold `(Machine : Running, (e2,e2));`
hold `(SafetyCondition : true, (e1, e4));`
when recognized {**report** `"Successful load";` }}

Le suivi et la reconnaissance en temps réel des chroniques [Dou94] sont effectués en plusieurs étapes :

- transformation des assertions en événements "interdits" (c'est-à-dire qu'un événement ne doit pas changer la valeur d'attribut de l'assertion pendant la durée de validité de l'assertion.)

- la mise à jour (en fait la restriction) incrémentale des fenêtres temporelles (intervalles d'instants acceptables pour les événements attendus pour compléter une instance de chronique) à l'arrivée de chaque événement.

- la détection d'échéances de "deadlines" et de violation d'assertions.

Il est tenu compte des délais éventuels de transmission des événements. Le traitement des chroniques possibles se fait par la gestion d'un arbre d'instances de chroniques.

2.2.2 Que retenir ?

La supervision est une notion complexe. Elle se décompose elle-même en plusieurs tâches : diagnostic, prédiction, suivi d'évolution,... Ces tâches ont pour point commun la nécessité de choisir un formalisme de représentation des objets de supervision et des mécanismes qui tiennent explicitement compte du temps.

Le modèle des chroniques nous paraît le formalisme qui représente le meilleur compromis expressivité-faible complexité. On dispose de plus avec IxTeT d'un outil qui implémente déjà les mécanismes de suivi d'évolution, de diagnostic et de mise à jour. On supposera par la suite qu'on dispose d'un tel système, et en tout cas on adoptera le modèle des chroniques comme formalisme pour la supervision de réseaux.

La section suivante traite d'architectures intelligents pour la supervision de réseaux.

2.3 Supervision de Réseau et Intelligence Artificielle

Tout d'abord il convient de faire une distinction entre "Réseaux Intelligents", et "Réseaux utilisant des techniques d'Intelligence Artificielle" [GP92]. Le premier terme désigne généralement des réseaux de télécommunication qui offrent des services évolués tels que le numéro vert, les téléconférences etc... La mise en place de ces réseaux nécessite des protocoles particuliers de transport de l'information de commande qu'on appelle communément la *signalisation*. Notre propos concerne plutôt le second terme, c'est-à-dire les réseaux qui, pour leur bonne gestion, font appel à des techniques d'Intelligence Artificielle.

Les systèmes experts ont souvent été expérimentés en gestion de réseau. On peut notamment citer SIMNETMAN [ZTD88] qui utilise une base d'une centaine de règles réparties sur des systèmes experts locaux, et qui a pour but de contrôler le trafic dans le réseau.

Data Pack Advisor est un système d'aide à la décision qui propose des outils de filtrage d'alarmes [RRCS88].

Or certains systèmes de supervision de réseaux ont une architecture multi-agent. Le paragraphe suivant en présente quelques-uns.

2.3.1 Supervision Intelligente

Dans le cadre du projet DOMAINS, D.Gaïti propose un noyau intelligent [Gai91] [Gai92] de contrôle des ressources d'un réseau à travers 2 modes de gestion :

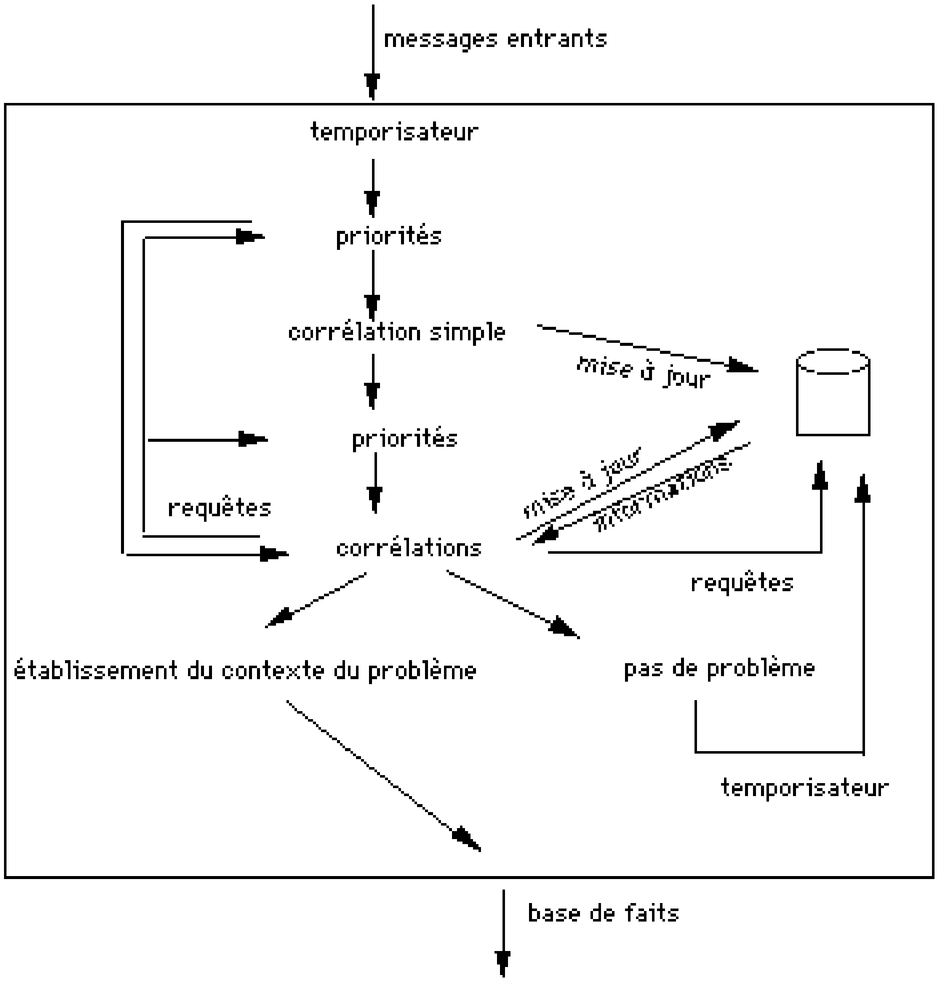
1. le mode "prévisionnel": un filtre détecte les événements significatifs pour déclencher des actions prévisionnelles. En cas de situations inattendues, le noyau passe en mode "opérateur".

2. le mode "opérateur" : les actions sont effectuées par l'opérateur mais néanmoins stockées dans un historique pour une utilisation future.

Le système de filtrage (fig. 2.9) mis en place permet notamment d'écarter les événements qui découlent d'autres événements, tout en temporisant afin de ne pas corriger des situations qui peuvent se rétablir d'elles-mêmes. Le pré-filtrage laisse de côté des données a priori non pertinentes qui sont cependant stockées pour permettre des requêtes au cas elles auraient été écartées trop tôt. Ensuite une corrélation est effectuée entre les événements jugés significatifs pour déterminer le contexte du problème. Cette corrélation est faite dans le temps (les événements sont répartis sur un intervalle de temps) et dans l'espace (un événement peut induire d'autres événements via le réseau). Le temps est considéré comme un événement du réseau: la fréquence d'un événement et les durées sont des facteurs non négligeables.

S'inspirant entre autres des travaux de Gaïti, D.Siboni propose une fonction intelligente de gestion globale des incidents pour un hyperviseur (superviseur de superviseurs) [SARJ94]. Le système décrit utilise, outre l'arbre d'héritage de GDMO et l'arbre de contenance, deux modèles de représentation supplémentaires :

- un modèle de dépendances fonctionnelles entre les différents éléments de réseau ;
- un graphe de causalité d'événements qui matérialise l'expertise concernant les dépendances entre éléments du réseau.



Fonctionnement du filtre intelligent

FIG. 2.9 -

Trois étapes sont distinguées dans la gestion des incidents (La figure 2.10 présente le processus complet de gestion des incidents) :

1. *Le filtrage et la corrélation passive des événements :* ceci permet d'éliminer les événements jugés secondaires en utilisant la technique "système expert basé sur des modèles". Le système prédit le comportement des composants du réseau, et :

   – soit les événements attendus expliquent le problème courant ;

  - soit ils sont dérivés du problème en cours, et sont donc filtrés.

  Or l'environnement est dynamique : un événement peut par exemple être modifié ou contredit par un autre, d'où une gestion d'hypothèses de problèmes qui intègre un processus de temporisation du diagnostic. Une hypothèse peut :

  - être confirmée par l'occurence d'événements prévus ;
  - être induite par un autre problème : elle est onc expliquée ou réduite par l'hypothèse correspondante ;
  - disparaître si le problème disparaît.

2. *Le diagnostic actif :* il s'agit d'une recherche de causalité approfondie en effectuant des tests pour confirmer les hypothèses. Or des événements survenant ultérieurement peuvent aussi confirmer (ou infirmer) ces hypothèses courantes ; il faut donc que la phase de diagnostic soit interruptible. Ceci d'autant plus qu'il faut pouvoir tenir compte en temps réel de pannes de gravités et de priorités différentes.

3. *La gestion du problème :* elle est fondée sur l'utilisation de tickets d'incidents qui sont des fiches qui contiennent tous les renseignements nécessaires à l'analyse de l'incident. Ces tickets d'incidents sont ensuite stockés pour permettre une analyse statistique des défaillances du réseau.

La figure 2.10 présente le processus complet de gestion des incidents :

Le système expert MAITE [GH92] propose une aide aux opérateurs qui maintiennent le réseau IBERCOM. Etant donné le nombre d'alarmes différentes (environ 120) et leur fréquence (80/h), ainsi que la difficulté de les interpréter et de les corriger (il existe des manuels de plus de 2000 pages !), MAITE guide et conseille l'opérateur pendant tout le processus de traitement des pannes, qui se décompose en quatre étapes :

- le tri des alarmes : les alarmes sont classées en fonction de leur type et de leur degré d'urgence. Une alarme peut simplement indiquer une panne provisoire. Enfin quand une alarme contredit ses prédécesseurs, il devient nécessaire de décider des actions de vérification de consistance des alarmes;

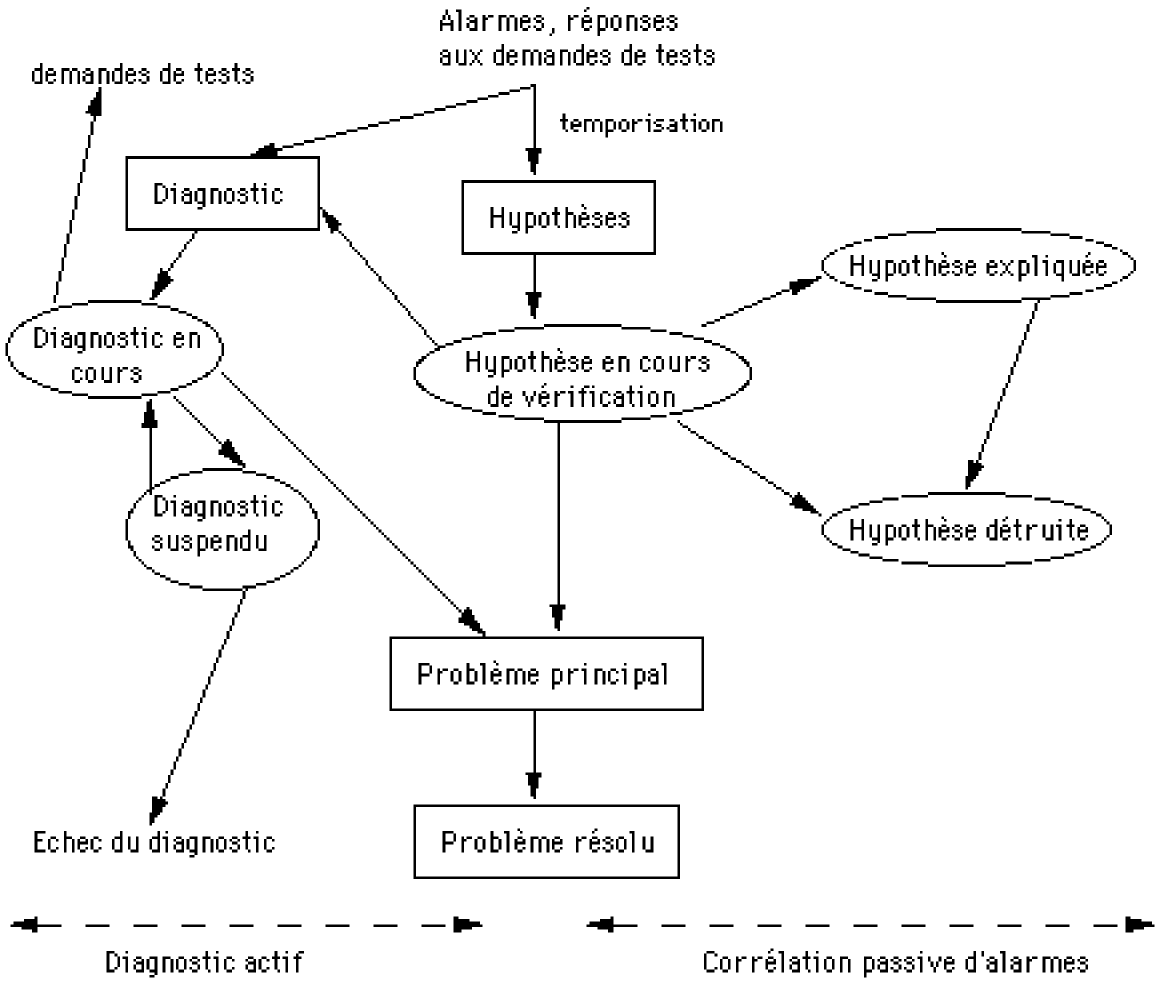


FIG. 2.10 - *La gestion des incidents dans le système de Siboni*

- le diagnostic de pannes : les alarmes donnent souvent des indices explicites sur la localisation de la panne. Les procédures de tests d'hypothèses de pannes sont indiquées dans MAITE, qui suggère les actions à effectuer, et en effectue certaines après une demande de validation de l'opérateur ;

- la réparation des pannes : MAITE offre les mêmes fonctionnalités que pour le diagnostic de pannes ;

- le stockage des problèmes rencontrés : à des fins de statistique.

Le paragraphe suivant détaille les architectures multi-agents des systèmes que nous venons de décrire. A chaque fois, nous décrirons d'abord les différents types d'agents qui constituent le système, puis la structure interne d'un agent et enfin comment s'organisent ces agents.

2.3.2 Architectures Multi-agents pour la supervision

2.3.2.1 Architecture du système de Gaïti

Le système conçu par D. Gaïti repose sur une technique d'Intelligence Artificielle Distribuée appliquée à une collection d'agents semi-autonomes et intelligents. Chaque agent constitue un résolveur de problème et peut faire appel à d'autres agents.

Une telle architecture multi-agents s'organise en une collection d'agents, une communauté d'agents système, et une base de données de description.

2.3.2.1.1 Les différents agents Dans une collection d'agents, chaque agent contient la connaissance des autres agents de son environnement : identité, localisation, et capacités. Ils peuvent ainsi s'organiser en groupes ou "sous unités" afin de répondre à des problèmes spécifiques.

La communauté d'agents système fournit une interprétation des commandes, une interface utilisateur, des agents constructeurs, et un gestionnaire d'erreurs. Elle peut ainsi prendre en charge la gestion des agents (construction et édition de comportement), la communication entre agents et processeur, et l'ordonnancement des agents à l'exécution.

La base de données de description est gérée par les agents système qui vérifient les descriptions, en construisent de nouvelles. Ils construiront les agents à partir de ces descriptions.

2.3.2.1.2 La structure d'un agent L'agent possède une structure (fig. 2.11) constituée de trois entités :

- un module qui comprend une partie qui gère les entrées-sorties (requêtes et réponses) et une autre qui donne l'accès à la connaissance,

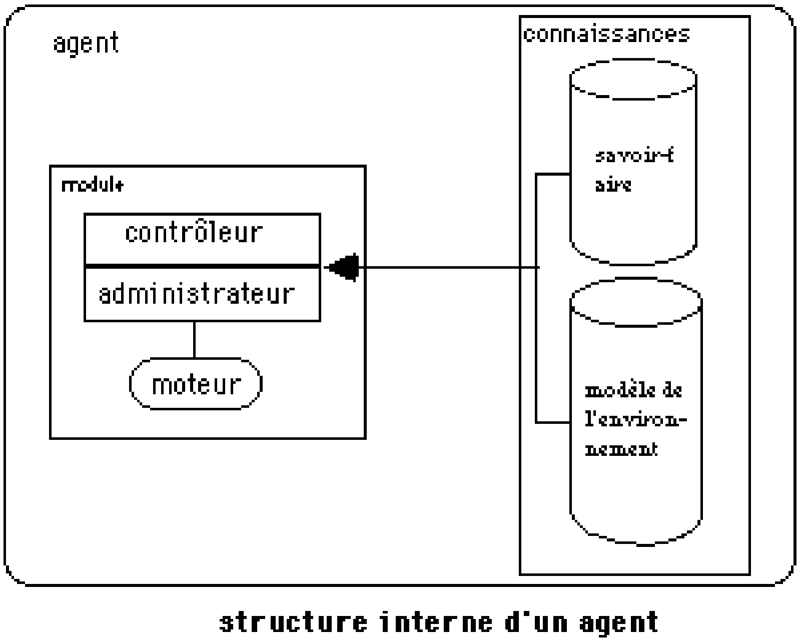


FIG. 2.11 -

- la connnaissance proprement dite de l'agent : ses capacités et l'ensemble des agents qui peuvent coopérer avec lui,
- le moteur d'inférence qui exécute les connaissance de l'agent.

2.3.2.1.3 L'organisation des agents Lors de la résolution d'un problème spécifique, les agents peuvent s'organiser selon une répartition des tâches ou une répartition des connaissances. Cette répartition est toujours gérée par un agent, dit gestionnaire, chargé d'envoyer des requêtes, ou allouer des tâches. Ce processus est itératif. On arrive donc à une architecture hiérarchisée (fig. 2.12).

2.3.2.2 Le système MAITE de Garijo

Le système MAITE se décompose en trois entités : une interface utilisateur chargée de récupérer des données entrées par l'opérateur, une interface de communication qui assure le lien entre le module de raisonnement et le système IBERCOM, et enfin le module de raisonnement proprement dit qui est un résolveur de problèmes.

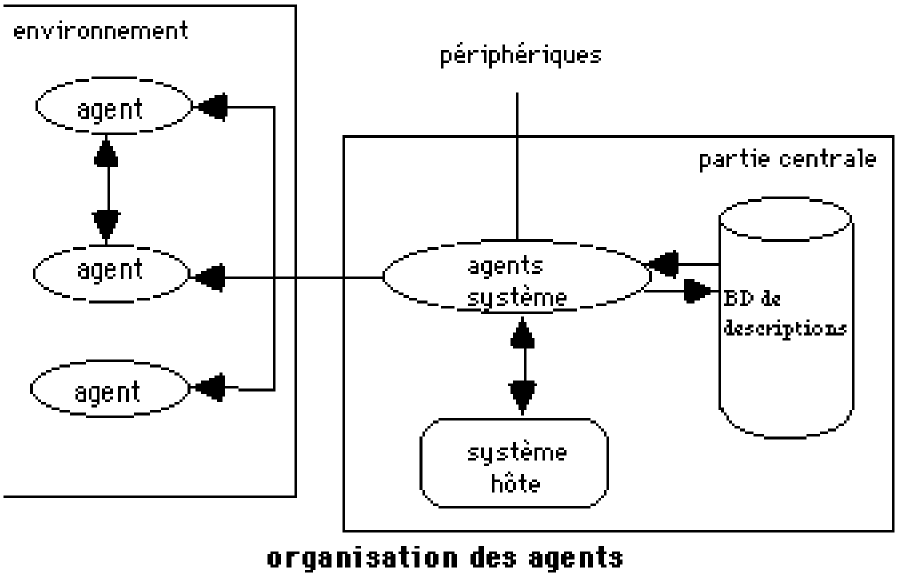
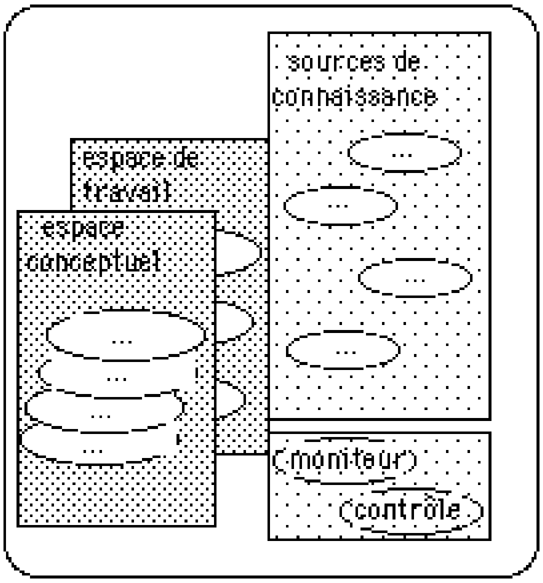


FIG. 2.12 -

Le module de raisonnement s'organise autour de deux sortes d'agents : les agents génériques, et les agents spécialisés.

2.3.2.2.1 Les différents agents

- Les agents génériques :
  - l'agent d'interprétation du modèle possède la connaissance des composantes du système, la description du réseau dans lequel s'intègre son environnement, une description des actions. Il pourra localiser les sources d'un problème, fournir une vision de l'état des composantes du réseau, créer de nouvelles hypothèses plus précises.
  - l'agent de contrôle représente un module de communication avec l'opérateur, fournit le diagnostic final, et permet d'exécuter les processus de réparation. Il exploite différentes sources de connaissance pour obtenir : une interface cohérente avec l'opérateur, un contrôle des communications pour éviter des redondances, un filtrage des analyses et des actions des agents spécialisés, et enfin un gestionnaire d'erreurs.



structure interne d'un agent du système MAITE

FIG. 2.13 -

- Les agents spécialisés :
  - un groupe d'agents se charge de gérer les alarmes. Ceux-ci détectent les occurrences répondant à leur compétences et proposent des hypothèses. L'activité passe ensuite à un autre groupe.
  - les agents qui s'occupent du traitement des composantes du système. Chacun d'entre eux représente une composante. Il peut créer et valider des hypothèses, proposer des diagnostics, provoquer des actions de réparation.

2.3.2.2.2 La structure d'un agent Chaque agent (fig. 2.13) est un "blackboard" constitué d'un espace dit "conceptuel" regroupant la connaissance des composantes du système et les données relatives à la résolution du problème, et d'un espace de travail qui contient les informations reçues par d'autres agents ou par lui-même pendant un processus de résolution.

Les sources de connaissances répondent aux informations de l'espace de travail, et peuvent en générer de nouvelles. Elles sont de deux types : relatives au domaine,

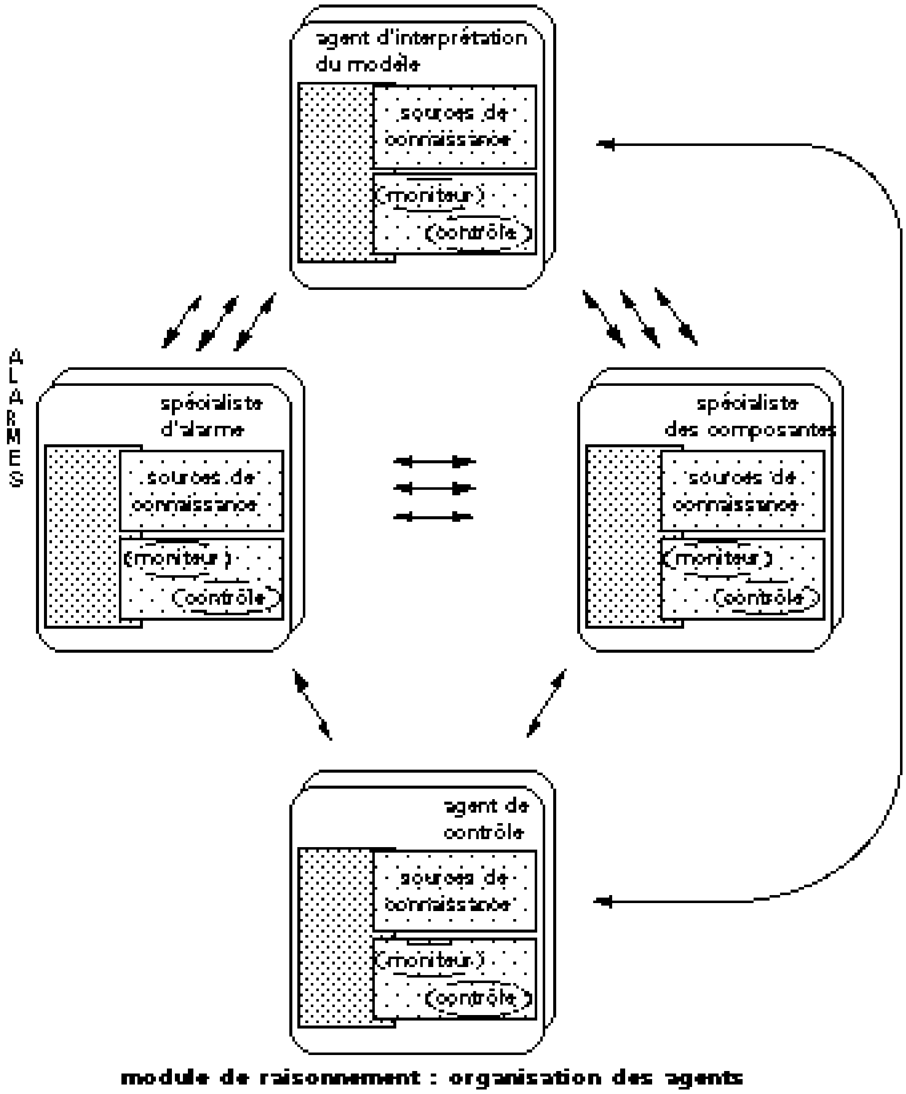


FIG. 2.14 -

pour interpréter des actions ou valider des hypothèses; relatives au contrôle, pour sélectionner des procédures en se basant sur des règles de production.

2.3.2.2.3 L'organisation des agents Les agents s'organisent (fig. 2.14) selon un schéma classique pour la résolution de problèmes : un agent qui déduit l'état des composantes du système, un agent qui interprète et classe les alarmes, et un agent qui génère des hypothèses et provoque des actions de réparation.

2.3.2.3 Architecture du système de Siboni & al

Dans son système d'hyperviseur, D. Siboni envisage deux architectures possibles : une architecture centralisée, ou une architecture décentralisée.

Si on veut calquer son système sur une architecture multi-agents, il apparaît deux sortes d'agents : un agent central, des agents locaux.

2.3.2.3.1 Les différents agents L'agent central est en fait un hyperviseur. Il regroupe les trois fonctions suivantes : filtrage, corrélation, diagnostic actif. Il réagit aux événements envoyés par les agents locaux.

Les agents locaux sont des superviseurs. Ils envoient un ensemble d'événements relatifs à l'état du système. Ils peuvent assurer les mêmes fonctions que l'hyperviseur, en fonction de l'architecture choisie, mais à un niveau local.

2.3.2.3.2 La structure d'un agent Chaque agent est un système expert, qu'il soit central ou local. Seul le champ d'application du système expert varie en fonction du type d'agent.

2.3.2.3.3 L'organisation des agents Dans le cas d'une architecture centralisée, on retrouve une structure qui contient un agent central autour duquel s'organisent des agents locaux. Ce sera l'agent central qui assurera la partie active du processus de gestion des incidents.

L'architecture décentralisée fournit un ensemble de superviseurs. Chacun d'entre eux assure filtrage, corrélation, et diagnostic, mais seulement à un niveau local de leur environnement. Les corrélations appelleront à une coopération des superviseurs. Un agent central est toujours présent, mais il n'assure que la corrélation finale, déchargé des autres tâches par les agents locaux.

2.3.3 Que retenir ?

La supervision "intelligente" consiste essentiellement à filtrer les alarmes, à diagnostiquer les incidents et éventuellement prescrire des actions correctrices. Or finalement, alors que le domaine des réseaux semble bien s'y prêter, on n'a pas compté un grand nombre d'applications multi-agents. En général, au lieu de trouver de vrais cas de distribution d'"intelligence" dans le réseau, on a surtout recensé des architectures d'IA distribuée au sein d'une même plateforme de gestion ou de supervision. Or, bien qu'on puisse considérer qu'il s'agisse déjà là d'une première évolution par rapport aux "simples" systèmes experts, l'adoption d'architectures multi-agents n'est pas toujours suffisamment argumentée, et l'apport de l'approche multi-agents n'a pas été évalué. En effet, on n'y montre pas ce qu'on gagne dans l'utilisation de telles techniques : du temps ? De l' "émergence" ? De la simplicité de conception ? D'autre part certaines architectures sont centralisées et les agents s'apparentent plutôt à des "modules" : sont-ils autonomes ? Est-ce un système ouvert ? Le système est-il tolérant à l'ajout ou la perte d'un "agent" ?

Pour conclure :

- **"Tout ce qui brille n'est pas Agent"** : il faut se méfier des dénominations abusives ;
- **"L'Agent ne fait pas** *toujours* **le bonheur"** : l'approche multi-agents ne se justifie pas tout le temps : la programmation parallèle nous apprend par exemple que la communication entre processus est toujours coûteuse.

On semble donc encore à la recherche de réelles applications. On peut penser qu'avant d'arriver à de véritables systèmes coopératifs, il a déjà fallu bâtir des systèmes "isolés", et on peut prévoir que la prochaine génération de travaux consistera justement à établir un dialogue entre ces systèmes.

Un autre point semble intéressant : dans la plupart des systèmes présentés, un besoin se fait ressentir au niveau de l'entrée des règles d'expertise. La plupart des auteurs mentionnent l'apprentissage automatique comme une possibilité.

Récapitulons :

- Si application "intelligente" il faut, c'est surtout dans le diagnostic ;
- Si nous voulons étudier la possibilité de coopération entre systèmes, partons de systèmes existants au lieu de nous attarder à en concevoir d'autres ;
- L'apprentissage automatique n'est évoqué dans les travaux présenté que comme une perspective prometteuse. A nous donc de transformer ces perspectives en réalité !

Dans le paragraphe suivant nous présentons des concepts "Agents" qui nous semblent intéressants dans la perspective de leur intégration dans la supervision de réseaux.

2.4 Des concepts orientés “agent” pour la supervision

Schématiquement, selon Newell, le "Knowledge level" serait la couche finale de l'empilement des niveaux de langage, se situant juste au-dessus de la couche symbolique, et définissant la connaissance indépendamment de tout langage de programmation ou de représentation, tout en permettant de l'exprimer à l'aide d'instructions de langages de programmation. Son comportement obéira à un principe de rationalité selon lequel des actions sont sélectionnées pour atteindre les buts d'un *agent rationnel* [Gan].

Il semble intéressant de trouver une correspondance entre ces niveaux de langage et les couches ISO, en particulier de définir un "niveau connaissance" au-dessus des services de la couche "application". Ce niveau permettrait ainsi par exemple d'introduire et de spécifier des services “intelligents” en termes de croyances (*Beliefs*), de Désirs, d'Intentions (on parlera alors d'*agents BDI*)... Or beaucoup de travaux ont été menés pour spécifier de telles notions [CL90] [WJ95]. Parmi eux, les travaux de Shoham ont le mérite d'allier la spécification poussée à l'opérationnalisation des concepts présentés. Le paragraphe 2.4.1 présente donc son langage de programmation “orienté agent” qui utilise justement des notions mentales telles que les croyances, le choix...

Or étant donné le caractére incomplet des données de chaque site ou de chaque superviseur local, la résolution de certains problèmes nécessite la coopération entre les différents agents. Selon Bouron [Bou92], la coopération se compose de l'organisation et de la communication. Dans la communication en particulier, il est nécessaire de transmettre les intentions des agents. Il faut alors enrichir le langage de communication et le transformer en un véritable *langage d'interaction* [DBK94] [KDEQ95]. En tant que langage d'interaction, KQML semble s'imposer comme un standard.

Dans le paragraphe 2.4.2, nous présentons KQML, et le cadre général dans lequel il s'insère.

L'étape suivante consiste à utiliser les primitives de ce langage d'interaction dans des *protocoles d'interactions* (paragraphe 2.4.3) qui constituent un véritable début d'algorithmique de la communication.

Pour finir, nous présentons le concept d'*agent d'interface* (paragraphe 2.4.4) et son application dans des domaines concrets. Certains de ces agents utilisent justement des langages et protocoles d'interaction pour collaborer entre eux.

2.4.1 Des agents BDI (opé)rationnels : l'exemple de l'Agent-Oriented Programming (Yoav Shoham)

Shoham [Sho90] se propose de décrire un nouveau paradigme de programmation, dérivé de la programmation orientée objet, et qui s'intitule *Agent-Oriented Programming*. Pour Shoham, les agents sont : "des entités possédant une représentation formelle d'un état mental, et en particulier des représentations formelles de la connaissance, des croyances, des capacités, du choix, des engagements..." En ce sens, ses agents sont de "haut niveau", par opposition aux agents de Brooks, pour qui il n'est pas nécessaire que les agents possèdent une représentation explicite de leur environnement. Se pose alors la question de la légitimité d'utilisation de notions anthropomorphiques dans le cadre d'entités informatiques. Shoham cite McCarthy :

"To ascribe certain *beliefs, free will, intentions, consciousness, abilities* or *wants* to a machine or computer program is <u>legitimate</u> when such an ascription expresses the same information about the machine that it expresses about a person. It is <u>useful</u> when the ascription helps us understand the structure of the machine, its past or future behavior, or how to repair or improve it. It is perhaps never <u>logically required</u> even for humans, but expressing reasonably briefly what is actually known about the state of the machine in a particular situation may require mental qualities or qualities isomorphic to them. Theories of belief, knowledge and wanting can be constructed for machines in a simpler setting than for humans, and later applied to humans. Ascription of mental qualities is <u>most straightforward</u> for machines of known structure such as thermostats and computer operating systems, but is <u>most useful</u> when applied to entities whose structure is very incompletely known."

La programmation orientée agent peut être considérée comme une spécialisation de la programmation orientée objet au sens où l'état d'un agent est décrit par des paramètres tels que la connaissance, les croyances, les engagements, les capacités, etc... D'autre part les messages échangés sont typés à l'aide d'actes de langages tels que la requête ou la promesse.

AOP est composé de 3 parties :

- Un langage formel pour décrire l'état mental
- Un langage de programmation interprété, avec des primitives telles que REQUEST et INFORM
- Un compilateur.

Par la suite nous décrirons les deux premières parties, sachant que Shoham lui-même ne donne que quelques indications imprécises sur le compilateur.

2.4.1.1 La définition de l'état mental

La spécification de l'état mental d'un agent est réalisée grâce à une logique temporelle à "instants explicites" (explicit-time point-based logic) où les prédicats sont de la forme:

`p(trm1,...,trm n)[t]`

Les formules bien formées (fbf) de cette logique sont construites de façon classique. Il en est de même pour la sémantique. La plupart des notions mentales seront décrites à l'aide de modalités :

2.4.1.1.1 La croyance (Belief) et la connaissance (Knowledge) B est un opérateur de KD45, ce qui permet donc d'exprimer (f1 et f2 étant des fbf) :

```
(B(f1) et B(f1-> f2) ) -> B(f2)
B(f1) <-> B(B(f1))
Non (B(f1)) <-> B (Non (B(f1)))
Non (B(f1 et Non(f1)))
```

B[t]`(a, fbf)` est une fbf, et exprime qu'un agent 'a' croit que la fbf est vraie à la date t.

exemple: `B[3](a,B[10](b, like(a,b)[7]))`

La connaissance (Knowledge) : Shoham suppose pour simplifier que les croyances et les connaissances d'un agent ne sont pas distinctes.

2.4.1.1.2 L'engagement (Commitment) CMT est un opérateur ternaire de KD4 : `CMT[t](a,b,fbf)` est une fbf. Ainsi, un agent s'engage auprès d'un autre agent sur la véracité d'une proposition à un instant t.

L'engagement et la croyance doivent être mutuellement consistants :

- un agent est conscient de ses engagements :

  `CMT[t](a,b,fbf) <-> B[t] (CMT[t](a,b,fbf))`

- un agent a foi dans ses engagements :

  `CMT[t](a,b,fbf) -> B[t] (a,fbf)`

Le choix (Choice) est l'engagement vis-a-vis de soi-meme :

`CH[t](a,fbf) <-> CMT[t](a,a,fbf)`

La capacité (Capability) : ”être capable d'une chose, c'est avoir le pouvoir de rendre vrai le choix de cette chose”

`CAN[t](a,fbf) <-> ( CH[t](a,fbf) -> fbf )`

Certaines propriétés qui découlent de cette définition sont gênantes, comme par exemple :

`Non ( CH[t](a,fbf) ) -> CAN[t](a,fbf)`

Mais d'autres propriétés s'avèrent très intuitives :

`CMT[t](a,b,fbf) -> B[t] (a, CAN[t](a,fbf) )`

2.4.1.1.3 La persistance de l'état mental On stabilise la croyance dansle temps :

- persistance des croyances par défaut (mémoire et foi) :

  `(B[t] (a, fbf) et Non (LEARN[t ](a, Non(fbf)))) -> B[t+1] (a, fbf)`

- l'ignorance (l'absence d'une croyance) persiste jusqu'à ce qu'on apprenne cette croyance :

`Non (B[t] (a, fbf)) et Non (LEARN[t ](a, fbf))) -> Non (B[t+1] (a, fbf))`

– persistance des engagements.

2.4.1.2 La programmation des agents

Les définitions et propriétés énoncées précédemment donnent un cadre général à la programmation d'agents. Cependant dans un programme d'un agent particulier, il est nécessaire de spécifier plus précisément son comportement. Ceci se fait grâce à AGENT0, dont on va voir la syntaxe ainsi que son interprétation, qui se fait bien entendu conformément aux définitions données sur l'état mental.

2.4.1.2.1 Le principe d'AGENT0 :

– s'engager pour le futur ;

– accomplir les engagements qui arrivent à échéance.

2.4.1.2.2 La syntaxe d'AGENT0

– les faits : éléments de base d'AGENT0, ce sont des fbf qui peuvent décrire le monde ou l'état mental de l'agent.

– les actions : de deux types : privées (DO) ou communicantes (INFORM, REQUEST, UNREQUEST, REFRAIN).

  Les actions conditionnelles sont des actions qui impliquent un test au moment de leur exécution (à distinguer des conditions que l'on pose avant de s'engager à quelque chose) : IF mentalcondition THEN action, par exemple :

  `IF (B, employee(smith,acme)) THEN INFORM(t,a,employee(smith,acme))`

– les variables : permettent l'unification au sens de la programmation logique. ex:

  `IF NOT ((CMT,?x),REFRAIN action) THEN action`

– Les déclarations d'engagement (Commitment statements) :

Un programme en AGENT0 est une séquence de conditions qui déterminent l'engagement à des actions (étant données un ensemble de capacités ainsi qu'un ensemble de croyances initiales) :

```
IF MSGCOND: msgcond AND MNTLCOND: mntlcond THEN COMMIT TO agent ABOUT
action
```

ou plus simplement :

```
COMMIT (msgcond, mntlcond, agent, action)
```

exemple :

```
COMMIT( (?a, REQUEST, ?action),
              (B, myfriend(?a)),
              ?a,
              ?action)}
```

2.4.1.2.3 L'interpréteur d'AGENT0 Il effectue itérativement les deux étapes suivantes :

1. traiter les messages reçus, qui conduisent à la mise à jour des croyances et engagements
2. traiter les engagements de l'instant courant, ce qui conduit éventuellement à des changements de croyance.

Shoham fait l'hypothèse lourde selon laquelle une boucle d'itération de l'interpréteur s'effectue en un temps inférieur à l'intervalle entre deux ticks d'horloge. Voyons maintenant comment s'effectuent les différentes mises à jour et exécutions effectuées par l'interpréteur :

- La mise à jour des croyances : se fait lors du déclenchement d'une action DO ou à la réception de messages INFORM. Dans le deuxième cas :
  - ajout du couple (a, fact) à la base de croyances
  - vérification et correction des éventuelles inconsistances générées. On supposera avoir fait suffisamment de restrictions sur la syntaxe des croyances pour éviter que cette vérification s'avère trop coûteuse.

- La mise à jour des engagements :
  - pour chaque UNREQUEST, enlever la requête correspondante ;
  - dans le programme, pour chaque COMMIT, vérifier les conditions et l'absence de REFRAIN correspondant, et s'engager alors.
- L'exécution des engagements :

  Pour chaque engagement correspondant à l'instant courant, exécuter l'action :
  - envoi de message à un autre agent ;
  - exécution d'une action privée (DO) ;
  - faire les mises à jours nécessaires.

2.4.1.3 Discussions sur AOP

La modélisation du comportement d'un agent en terme d'état mental peut s'avérer intéressante pour améliorer la compréhension qu'un usager peut avoir d'un système. Dans [EPQD96], Michel Plu explique comment on pourrait par exemple modéliser le système de reconnaissance de chroniques pour rendre compte à l'usager de l'état d'avancement de la reconnaissance, ainsi que des contraintes limitant les capacités du système (charge du réseau, ...).

Shoham reconnaît lui-même le besoin d'une "compilation" de son langage en un langage de plus bas niveau qui serait plus efficace en terme de coût de calcul. En attendant, on peut très bien utiliser (voire étendre) ce langage pour faire de la spécification de systèmes complexes, la traduction en un langage de programmation efficace se faisant "manuellement". Nous verrons dans la suite une ébauche de spécification de notre système avec ce type de langage.

2.4.2 L'interaction entre agents

Le DARPA Knowledge Sharing Effort [GK94] a pour but de :

- Faire communiquer des logiciels existants et hétérogènes : les programmes peuvent en effet être écrits dans des langages différents et ne pas avoir des entrées-sorties qui se correspondent. On aimerait réutiliser ces logiciels sans avoir à les réécrire pour les rendre compatibles.

– Faciliter la mise en place d'applications en IAD (Bases de connaissances distribuées, systèmes multi-agents...), en réutilisant des spécifications que l'on pourrait éventuellement spécialiser suivant le domaine, en rendant la communication transparente au programmeur, et en permettant l'intégration facile de nouveaux systèmes (principe du "plug and play").

Le moyen préconisé est de transformer ces programmes en "Software Agents", qui interagissent à l'aide d'un langage de communication normalisé : ACL (Agent Communication Language). Ce langage se compose de trois parties :

– Le vocabulaire : les *ontologies* ;

– Un langage interne : KIF ;

– Un langage externe : KQML.

2.4.2.1 Le langage KIF

KIF est un langage de partage des connaissance entre agents. Ceux ci ne sont pas sensés avoir le même univers du discours, et seuls les concepts de base employés en KIF sont supposés connus: les ensembles, les listes, les nombres, l'élément bottom et la syntaxe de KIF.

2.4.2.1.1 Formalisme Le formalisme de KIF est basé sur la logique du premier ordre, sans restrictions. Il est donc possible d'exprimer des propositions indécidables en KIF. On dispose ainsi de la négation, la disjonction, la conjonction, l'implication, des quantificateurs existenciel et universel, et de la conséquence logique. Les variables peuvent être des variables individuelles ou des listes.

2.4.2.1.2 Syntaxe La syntaxe de KIF est de la forme lisp, parenthésée et préfixée. On exprimera par exemple l'expression "Q(x,y) implique P(x) et P(y)".

```
(=> (Q?x?y) (and (P?x) (P?y)))
```

Pour décrire l'univers du discours, on peut définir des objets, des relations et des fonctions. Une relation est spécifiée en intension comme l'ensemble en extension de tous les tuples (représentés par des listes) qui vérifient cette relation. Une relation n'est pas sensée avoir une arité fixée (tous les tuples n'ont pas nécessairement la même longueur).

Par exemple ((4 8) (4 4 5) (3)) est une relation définie en extension.

Une fonction est une relation pour laquelle, en ayant fixé la valeur des n-1 variables d'un tuple de longueur n, il existe une unique valeur possible pour la dernière variable.

La fonction carré serait définie en extension sur N par: ((0 0) (1 1) (2 4) (3 9) ...)

KIF ne fait aucun calcul, de même qu'il ne fait aucune restriction sur les spécifications exprimées en KIF. C'est uniquement un format de représentation de la connaissance. Il sert de langage support pour Ontolingua.

2.4.2.2 Ontolingua

2.4.2.2.1 Notion d'ontologie D'après T.R.Gruber [Gru93], une ontologie est une "spécification d'une conceptualisation". Il s'agit d'abord de conceptualiser un domaine, i.e de définir un ensemble de concepts et de relations employés par un agent en prise avec le domaine. Cette conceptualisation est ensuite spécifiée dans un formalisme pouvant être traduit dans un langage calculable.

2.4.2.2.2 Construction d'ontologies Le but d'une ontologie est le partage et la réutilisation de connaissances entre des agents logiciels. Un agent qui souscrit à une ontologie en utilise le vocabulaire pour décrire le domaine, de manière consistante avec la théorie définie par l'ontologie. Les agents sont construits en prenant en compte les ontologies, et les ontologies sont construites pour pouvoir partager de la connaissance entre ces agents.

2.4.2.2.3 Caractéristiques d'Ontolingua Le logiciel Ontolingua est implémenté comme un ensemble de bibliothèques pour le CommonLisp. Il fait appel à Clos, la surcouche objet du CommonLisp. Ontolingua n'étant pas un langage destiné directement à inférer de la connaissance à partir des ontologies, son rôle est plutôt de les traduire dans des implémentations calculables, et de servir de bibliothèque d'ontologies.

2.4.2.2.4 L'outil Ontolingua Ontolingua remplit les fonctionnalités suivantes :

- Interprétation du langage KIF.
- Vérification d'ontologies, de leur syntaxe KIF et de la présence des concepts référencés.
- Traduction vers des systèmes de représentation de connaissances (notamment Loom, un dérivé de KL-ONE).
- Génération de documents WWW permettant de naviguer dans une ontologie par les termes et les relations.
- Format de Frame générique permettant de sauvegarder les ontologies sous un format standard ou de les manipuler en mémoire à partir du lisp.

Un éditeur graphique d'ontologies est accessible à Stanford sous WWW, permettant de réaliser graphiquement une ontologie et d'importer ou exporter le code source.

2.4.2.2.5 Sémantique d'Ontolingua Les définitions qui suivent sont extraites du noyau d'Ontolingua, c'est à dire des ontologies qui définissent Ontolingua. Ontolingua est donc décrit en ses propres termes :

- **Relation :** Ensemble de tuples.

```
(define-class RELATION (?relation)
"A relation is a set of tuples.''
  :iff-def  (and  (set ?relation)          ; ?relation is a set
            (=>   (member ?tuple ?relation)
                  (list ?tuple))))         ; ?tuple is a sequence
```

L'arité d'une relation n'est pas forcément définie.

```
(define-function  ARITY (?relation)
"Arity is the length of the tuples in a relation.  If the lengths of tuples in
the relation vary, then the arity is undefined.''
  :def (and   (relation ?relation)
           (not (empty ?relation))
           (or     (and    (integer (arity ?relation))
                           (=>   (member ?tuple ?relation)
                           (= (length ?tuple) (arity ?relation))))
                   (undefined (arity ?relation)))))
```

- **Classe :** Relation unaire, chaque tuple est une instance.

```
(define-class CLASS (?class)
"A class is a unary relation.''
        :iff-def (and   (relation ?class)
                        (= (arity ?class) 1)))
```

- **Fonction :** Relation fonctionnelle.

```
(define-class FUNCTION (?relation)
"A function is a mapping from a domain to a range
that associates a domain element with exactly one range element.''
  :iff-def  (and  (relation ?relation)
                  (=>  (member ?tuple1 ?relation)
                       (member ?tuple2 ?relation)
                       (= (but-last ?tuple1) (but-last ?tuple2))
                       (= (last ?tuple1) (last ?tuple2)))))
```

2.4.2.2.6 Bibliothèque d'ontologies Les ontologies sont représentées en treillis, les plus spécifiques incluant les plus générales (à gauche). L'inclusion d'une ontologie A par une ontologie B signifie que B pourra décrire ses classes comme des sous classes des classes définies en A (ou dans une ontologie incluse dans A), et utiliser tous les termes et relations définis en A.

```
Kif-Sets
        Kif-Extensions
                Frame-Ontology
                        Parametric-Constraints
                                Components-With-Constraints
                        Component-Assemblies
                                Components-With-Constraints
                        Basic-Matrix-Algebra
                        Abstract-Algebra
                        Slot-Constraint-Sugar
        Kif-Meta
                Parametric-Constraints ...
        Kif-Relations
                Frame-Ontology ...
                Kif-Extensions ...
Kif-Numbers
        Kif-Extensions ...
        Kif-Lists
                Kif-Extensions ...
                Kif-Meta ...
                Kif-Relations ...
User-Theory
```

Les ontologies constituant le ”noyau” d’Ontolingua sont incluses dans Frame-Ontology. Celles nommées KIF-* décrivent comme des ontologies les concepts manipulés par KIF, et KIF-Meta est une ontologie décrivant la syntaxe de KIF.

2.4.2.3 KQML : Knowledge Query And Manipulation Language

KQML est un langage d’échange d’informations et de connaissances entre agents. Etant donné une syntaxe interne commune à deux agents (le cas échéant on pourra utiliser KIF), il reste nécessaire de mettre en forme les messages échangés afin de préciser l'acte de discours que ces messages impliquent et les renseignements indispensables à l'acheminement du message. Tout cela est pris en charge par KQML, qui se compose de trois couches : "Content Layer", "Message Layer" et "Communication Layer".

2.4.2.3.1 Les couches du langage

1. "Content Layer" : elle contient la connaissance "brute" à transmettre. Elle peut a priori être exprimée en n'importe quel langage à partir du moment où il est commun à l'émetteur et au récepteur.

2. "Message Layer" : elle constitue la partie essentielle de KQML. Elle permet d'associer à chaque message l'acte du discours qu'il est sensé transmettre, afin que le récepteur sache se comporter en conséquence. On distingue deux types de messages : les "content messages" et les "declaration messages" :

   – les "content messages" sont les messages qui correspondent à des requêtes, des assertions... Ils commencent par le mot-clé MSG, et on y précise notamment le type du message (c'est-à-dire l'acte du discours employé), le langage dans lequel est décrit le contenu du message, l'ontologie employée et bien entendu le message lui-même.

     L'exemple suivant représente une requête pour obtenir la couleur de la neige :

     ```
     (MSG
             :TYPE query
             :QUALIFIERS ( :number-answers 1)
             :CONTENT-LANGUAGE kif
             :CONTENT-ONTOLOGY (blocksWorld)
             :CONTENT-TOPIC (physical-properties)
             :CONTENT (color snow ?C) )
     ```

   – les "declaration messages" sont en fait des déclarations d'intention d'émettre ou de recevoir des "content messages". Ils commencent par le mot-clé

DCL, et on y précise notamment s'il s'agit d'une "importation" ou d'une "exportation" et on décrit évidemment le message qu'on désire importer ou exporter.

L'exemple suivant représente une déclaration d'intention d'exporter des assertions concernant la couleur des objets du monde des blocks :

```
(DCL
        :TYPE assert
        :DIRECTION export
        :MSG (MSG
                        :TYPE assert
                        :CONTENT-LANGUAGE kif
                        :CONTENT-ONTOLOGY (blocksWorld)
                        :CONTENT-TOPIC (physical-properties)
                        :CONTENT (color ?X ?Y) ) )
```

3. "Communication Layer" : elle permet de constituer des packages contenant les renseignements indispensables à l'acheminement du message : le nom de l'expéditeur, celui du destinataire, l'identifiant du message... Exemple :

```
(PACKAGE
        :FROM ap001
        :TO ap002
        :ID DVL-f001-111791.10122291
        :COMM block
        :CONTENT
                (MSG
                        :TYPE query
                        :QUALIFIERS ( :number-answers 1)
                        :CONTENT-LANGUAGE kif
                        :CONTENT (color snow ?C) ) )
```

2.4.2.3.2 Les "performatives" Ce sont les actes de langages utilisés par KQML et qui correspondent au slot "type" d'un message. On peut bien sûr se contenter d'un sous-ensemble de ces actes, et on peut également en créer d'autres à condition de se conformer à une spécification stricte des slots associés. On peut regrouper les "performatives" en plusieurs parties distinctes :

1. "Basic informatives" : tell, deny, untel...
2. "Database performatives" : insert, delete...
3. "Basic responses" : error, sorry...
4. "Basic query performatives" : evaluate, reply, ask-if, ask-all...
5. "Multi-response query performatives" : stream-about, eos...
6. "Basic effector performatives" : achieve, unachieve...
7. "Generator performatives" : standby, ready, next...
8. "Capability-definition performatives" : advertise...
9. "Notification performatives" : subscribe...
10. "Networking performatives" : register, broadcast...
11. "Facilitation performatives" : broker-one, recommend-one, recruit-one...

Lors d'une interaction entre deux agents, ces "performatives" se combinent : à un "query" on répond par un "reply", à un "achieve" correspond la confirmation "tell", etc... Certaines "performatives" sont destinées à une utilisation particulière : les "databases performatives" pour interroger une base de données, les "facilitation performatives" pour dialoguer avec le "facilitator"...

2.4.2.3.3 Le "facilitator" Le "facilitator" est un outil qui a pour but de rendre transparente à une application la communication entre agents et la création des messages KQML. Il se décompose en deux parties distinctes : la "Facilitation Interface Library" (F.I.L. ou K.R.I.L. (KQML Router Interface Library) suivant les versions) et le "Facilitator" lui-même.

La "FIL" : elle a pour but de créer la "KQML message layer", c'est-à-dire qu'elle se charge entre autres d'attribuer une "performative" au message. L'application cliente doit simplement contenir une suite de déclarations qui décrivent la volonté d'importer ou d'exporter des connaissances sans avoir à préciser les agents qui sont concernés. La FIL se charge d'intercepter les importations et les exportations effectives et de les transformer en messages KQML, qu'elle adresse ensuite au "facilitator". La FIL, de par son intégration dans l'application, est partiellement écrite dans le langage de l'application (il existe à notre connaisance une FIL Prolog et une FIL en Common Lisp).

Le "facilitator" crée la "KQML communication layer" à partir de ce que lui envoie la FIL. C'est donc à lui de trouver les agents qui sont intéressés par les connaissances qui vont être exportées ou qui sont capables de fournir les connaissances à importer. Différentes méthodes sont envisageables suivant la taille du réseau d'agents : le broadcast, l'appel à un serveur de noms centralisé voire distribué. C'est dans ce dernier cas que des "facilitation performatives" sont utiles afin d'annoncer aux autres agents la disponibilité d'un nouvel agent avec ses besoins et capacités d'importation et d'exportation d'informations.

2.4.2.4 Conclusions sur KIF, Ontolingua et KQML

KIF, Ontolingua et KQML sont des outils récents, et les versions évoluent encore rapidement. Rappelons que leurs fonctions principales sont complémentaires, et la manière dont ils s'articulent en font un outil général pour le partage de connaissances. KIF est un format de description des connaissances, KQML un langage pour les partager, et Ontolingua contient les concepts génériques communs aux connaissances à partager.

Il est naturel de vouloir faire une comparaison entre KIF+KQML d'une part, et GDMO+CMIS de l'autre. Le reproche souvent fait à GDMO est qu'une bonne partie des spécifications est faite en langage naturel. Or les "performatives" de KQML ne sont pas non plus dénuées d'ambiguïté dans leur description [CL95].

Globalement néanmoins, KIF et KQML, munis de bibliothèques d'ontologies, offrent une richesse d'expression largement supérieure. Reste à savoir si ceci permet d'exprimer plus de choses que GDMO+CMIS, ou alors d'en exprimer autant en moins de messages. En fait de même qu'on dispose de modèles de calculabilité, il nous faudrait un modèle de "communicabilité" : de même qu'avec un jeu d'instructions minimum, on peut prétendre "tout faire", avec quels actes de langage minimum peut-on "tout dire", ou plutôt "tout faire faire" ? Ce débat dépasse notre champ de compétence et le propos de ce document, néanmoins nous constaterons dans la partie consacrée aux expérimentations, qu'en utilisant GDMO et CMIS, nous n'avons à aucun moment ressenti un manque d'expressivité. Peut-être l'usage de KQML nous aurait-il simplifié nos protocoles.

Le paragraphe suivant présente justement la notion de protocole d'interaction.

2.4.3 Les protocoles d'interaction

Nous avons vu que l'utilisation d'un langage d'interaction permet à chaque agent d'extraire explicitement, de la sémantique du message, de l'information qui serve à contrôler l'échange d'information et par la même de faciliter le contrôle de la société d'agents dans son ensemble. Or, selon Demazeau et Koning [DBK94] [EQKD95], le fait de découpler l'intention de l'expéditeur du message lui-même n'est pas suffisant. En effet, les agents doivent aussi savoir comment réagir à un message, ou bien ce qu'ils doivent attendre après avoir envoyé un message.

Il faut donc bâtir un cadre unique permettant ainsi de structurer les interactions entre agents, ce que Demazeau appelle le *protocole d'interaction*.

Un protocole d'interaction peut être représenté par un graphe d'état prédéterminé dans lequel l'agent évolue en fonction des types de messages reçus. Ceci conditionne son comportement, en particulier les types de messages qu'il doit à son tour émettre dans une situation donnée. On peut, grâce à ce type de graphes, modéliser des protocoles connus tels que celui des réseaux contractuels [Smi80].

Le paragraphe suivant présente les agents d'interface, qui peuvent être amenés à utiliser des protocoles d'interaction si une coopération s'avère nécessaire.

2.4.4 Des agents “spéciaux” : les ”agents d’interface”

Il s’agit d’agents dont le but est d’assister un opérateur en prenant en charge un certain nombre de tâches. On peut recenser plusieurs approches pour construire de tels agents :

- L’utilisateur programme lui-même l’agent, ce qui s’avère souvent complexe, trop figé (n’évaluant pas avec les habitudes de l’opérateur), et nécessite des connaissances et de la disponibilité de la part de l’opérateur ;

- L’approche ”basée sur les connaissances” consiste à bâtir un modèle de l’environnement et un modèle de l’utilisateur, ce qui s’avère souvent lourd et également peu évolutif ;

- La troisième approche, préconisée par Maes, consiste à “espionner” le comportement de l’utilisateur pour l’imiter ensuite. Nous allons détailler cette approche et son application par Maes et Shlimmer.

2.4.4.1 Assistance à l’utilisation de la messagerie électronique

La technique d’apprentissage utilisée par Maes [MK93] [Mae94] est celle de l’ ”apprentissage par coeur” où l’on mémorise des couples ”situation-action” en observant l’utilisateur. La prédiction d’une action se fait alors en trois étapes :

- Calcul de distance entre la situation actuelle et toutes les situations mémorisées ;

- Choix des actions les plus appropriées dans les N situations les plus proches ;

- Calcul du ”score” de l’action : c’est la somme, sur l’ensemble des situations prédisant la même action, de l’inverse de la distance entre la situation donnée et la situation actuelle.

On dispose de plusieurs ”niveaux de confiance”, suivant lesquels soit l’action est seulement suggérée, soit exécutée automatiquement sans consultation de l’utilisateur.

Il y a un oubli possible des situations trop anciennes.

D'autres méthodes d'apprentissage peuvent se greffer sur celle-ci :

– L'apprentissage par feedback : l'agent constate après coup si l'action qu'il a suggérée a été acceptée ou non ;

– L'apprentissage par entraînement : l'usager entre manuellement des couples "situation-action" ;

– L'apprentissage "multi-agents".

Parmi les différents modes d'apprentissage, l'auteur propose l'apprentissage "multi-agents" : quand une nouvelle situation est trop éloignée de celles déjà rencontrées, l'agent consulte ses "confrères" et tient compte de leurs recommandations [LMM94]. Cette collaboration se fait essentiellement en 2 modes :

– Le mode "désespéré" : quand un agent ne sait pas comment réagir face à un évènement, il se tourne vers ses confrères pour leur demander conseil.

– Le mode "exploratoire" permet à un agent de tester ses collaborateurs, pour savoir ce qu'eux feraient face à un type donné d'événements. L'agent peut ainsi repérer les confrères qui ont un comportement proche du sien, pour pouvoir s'adresser à eux lors du mode "désespéré".

Un agent qui reçoit une requête de collaboration ne sait pas a priori dans quel mode se trouve l'agent qui a émis la requête. En fonction des réponses aux requêtes, l'agent attribue à ses confrères une note de "confiance" dont il tiendra compte lors de ses requêtes futures.

2.4.4.2 Assistance à la prise de notes

Schlimmer et Hermens [SH93] ont mis au point un système d'assistance à la prise de notes sur des ordinateurs de poche à saisie au crayon optique (de type Newton). Le but du système est, en analysant les saisies antérieures de l'utilisateur, de lui proposer à tout moment une "complétion" possible de ce qu'il est en train de saisir. Il peut même apprendre et construire une interface constituée de menus pour offrir ainsi une alternative à la saisie "manuelle".

Les séquences de mots sont apprises sous la forme d'automates à états finis. Ces automates sont "compactés" et rendus aptes à la prédiction et à l'acceptation de nouvelles séquences grâce à des règles de fusion d'états qui leur confèrent la propriété de "k-réversibilité" [Ang]. D'autres règles permettent une généricité supplémentaire qui sera utile lors de la construction de l'interface à base de menus. Quand à partir d'un état, il y a plusieurs choix possibles ou une boucle représentant le caractère optionnel d'un mot, le système apprend un arbre de décision qui lui permet de faire le bon choix en fonction d'états antérieurs ou du fait que le mot optionnel a déjà été saisi ou non.

2.4.5 Que retenir ?

L'étude des travaux de Shoham nous a familiarisés avec la notion d'état mental, utile pour décrire et spécifier des systèmes à base d'agents. AGENT0 est par ailleurs un exemple de plateforme de développement de systèmes multi-agents, qui utilise des notions cognitives.

L'étude de KIF et KQML, si elle ne nous a pas conduit à adopter ces langages, nous a cependant confortés dans l'idée que GDMO associé à CMISE pouvaient constituer une base solide pour un langage d'interaction. Leur sémantique, plus pauvre, a en revanche le mérite d'être moins ambiguë.

Le concept d'agent d'interface nous paraît intéressant au point que nous l'adopterons par la suite et l'appliquerons à la supervision de réseau.

Nous retenons enfin la nécessité d'utiliser des protocoles d'interaction pour faciliter la coopération entre agents.

2.5 Bilan de l'état de l'art

- L'étude des normes de gestion de réseaux nous a permis de découvrir des formalismes de représentation des ressources gérées, ainsi qu'un langage d'interaction entre systèmes de gestion. Nous les adopterons.

- L'étude de la notion de supervision a mis à jour la nécessité de représenter explicitement le temps. Nous utiliserons à cette fin le modèle des chroniques manipulé dans le système IxTeT.
- Les différents systèmes "intelligents" utilisés dans les réseaux semblent se concentrer sur le diagnostic et le filtrage de pannes. Nous nous intéresserons également à ces problèmes en les abordant de la manière qui n'était jusque là évoquée que comme une perspective : l'apprentissage.
- Nous retenons de l'étude de travaux sur les agents la notion d' "agent d'interface", et la nécessité d'utiliser des protocoles d'interactions basés sur un langage d'interaction pour permettre la collaboration entre ces agents.

Le chapitre suivant décrira donc notre système d'assistance à la gestion et la supervision de réseaux [EQT95] [EDQD96] [EDQ96] sous la forme d'agents d'interface qui apprennent le comportement de l'opérateur sous la forme de chroniques, et qui collaborent pour améliorer leur performances en utilisant des protocoles d'interaction [KDEQ95] [EQKD95] s'appuyant sur les normes GDMO et CMIS.

Chapitre 3

Un assistant à la supervision de réseaux

3.1 Spécification et structure de l'assistant

On l'a vu (paragraphe 2.3.1), l'une des difficultés de la supervision de réseau provient du grand nombre d'alarmes à traiter. Beaucoup d'événements reçus par le superviseur sont insignifiants, et il faut les filtrer. D'autres événements ne prennent sens qu'en étant associés à leur contexte, c'est-à-dire entre autres aux événements qui les précèdent ou les suivent. Certaines pannes, décelables par une suite d'alarmes, peuvent apparaître simultanément, et il est nécessaire de les isoler pour les diagnostiquer correctement. Et enfin, les délais entre les événements attendus peuvent varier.

Il paraît donc intéressant d'utiliser le mécanisme de suivi et de reconnaissance de chroniques (cf paragraphe 2.2.1) pour la supervision : le superviseur pourra ainsi prendre à sa charge un certain nombre de tâches telles que le filtrage, l'archivage, ainsi que le diagnostic et la correction de certaines pannes pour soulager le travail de l'opérateur et lui permettre de se focaliser sur des problèmes graves ou inédits. On a déjà vu dans le paragraphe 2.3.1 des exemples de filtres intelligents et des outils de diagnostic qui s'appuient sur une description du modèle du réseau à gérer. La contrepartie est que ces filtres n'évoluent pas avec le temps. D'autre part on ne dispose pas nécessairement d'un modèle du réseau.

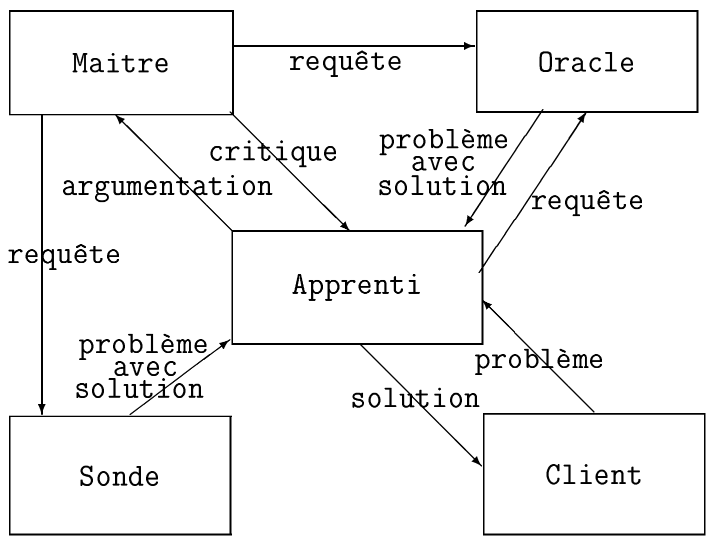


FIG. 3.1 - *Le protocole MOSCA*

L'apprentissage paraît donc une bonne solution pour acquérir les connaissances (décrites ici sous forme de chroniques) nécessaires à la supervision. Or un historique exhaustif de pannes n'existe pas toujours *a priori* , et on a toujours pour objectif de faire évoluer le système en même temps que le réseau. Les "agents d'interface" de Pattie Maes, vus dans le paragraphe 2.4.4, semblent donc être une bonne solution : il s'agira d'apprendre le comportement de l'opérateur en l'observant.

Toujours dans un souci d'évolutivité rapide du système (un nouveau type de panne peut par exemple survenir plusieurs fois dans la même journée), mais également pour que l'opérateur ait une bonne vision du système d'apprentissage, nous adopterons une méthode incrémentale et si possible en temps réel. Ainsi, si l'opérateur comprend bien le fonctionnement du système, il pourra interagir avec celui-ci pour en améliorer les performances, corriger certains biais, ainsi que ses propres erreurs de manipulation qui auraient pu être enregistrées, induisant ainsi le système en erreur. Il s'insère alors dans le cadre du protocole d'apprentissage MOSCA [RQ93]. Le protocole MOSCA est une analyse d'une situation d'apprentissage en termes de rôles et de messages typés. Le schéma 3.1 résume la nature des messages échangés selon les rôles tenus.

Dans notre cas le couple (réseau, opérateur) joue le rôle d'Oracle, puisque le réseau fournit les problèmes, et l'opérateur les solutions. Ce couple joue aussi le rôle de Sonde, puisqu'il fournit aussi les contre-exemples. Le rôle du Maître est joué par l'opérateur mais aussi éventuellement par d'autres agents d'interface. Le Client est le réseau.

L'association du système d'apprentissage et du système de suivi et de reconnaissance de chroniques peut donc constituer un véritable assistant à la supervision. La figure 3.2 présente la structure de l'assistant :

– Le système de reconnaissance de chroniques (SRC) reçoit les notifications d'événements, et tente de les apparier avec des chroniques stockées dans la *base de chroniques confirmées.* Si une chronique est reconnue, le système déclenche alors l'action correspondante (filtrage, action de réparation, diagnostic, enregistrement dans un journal,...). Si les événements reçus ne correspondent à aucune chronique, c'est alors à l'opérateur de supervision de prendre une décision.

– Le système d'apprentissage (SA) a pour but de trouver des corrélations entre les événements reçus et les décisions de l'opérateur, afin d'apprendre des chroniques qui puissent être utilisées par le SRC. Les nouvelles chroniques sont d'abord stockées dans la *base de chroniques non confirmées* avant d'être assez mûres pour être confirmées, c'est-à-dire transférées à la base confirmée.

Utiliser les notions d'état mental de Shoham (vues dans le paragraphe 2.4.1) peut être utile pour permettre à l'utilisateur de connaître l'état du système de reconnaissance de chroniques et ce qu'il a l'intention de faire. Ainsi, la chronique suivante[1] :

Chronicle `STATE-CHECK` {
event `(EV1(?connection), e1);`
event `(EV2(?connection), e2);`
`e2 = e1 + 2;`
`e3 = e1 + 7;`

1. Nous remarquerons au passage que nous avons légèrement modifié le modèle de chronique utilisé : nous datons également l'action, pour qu'éventuellement on puisse temporiser avant de prendre une décision.

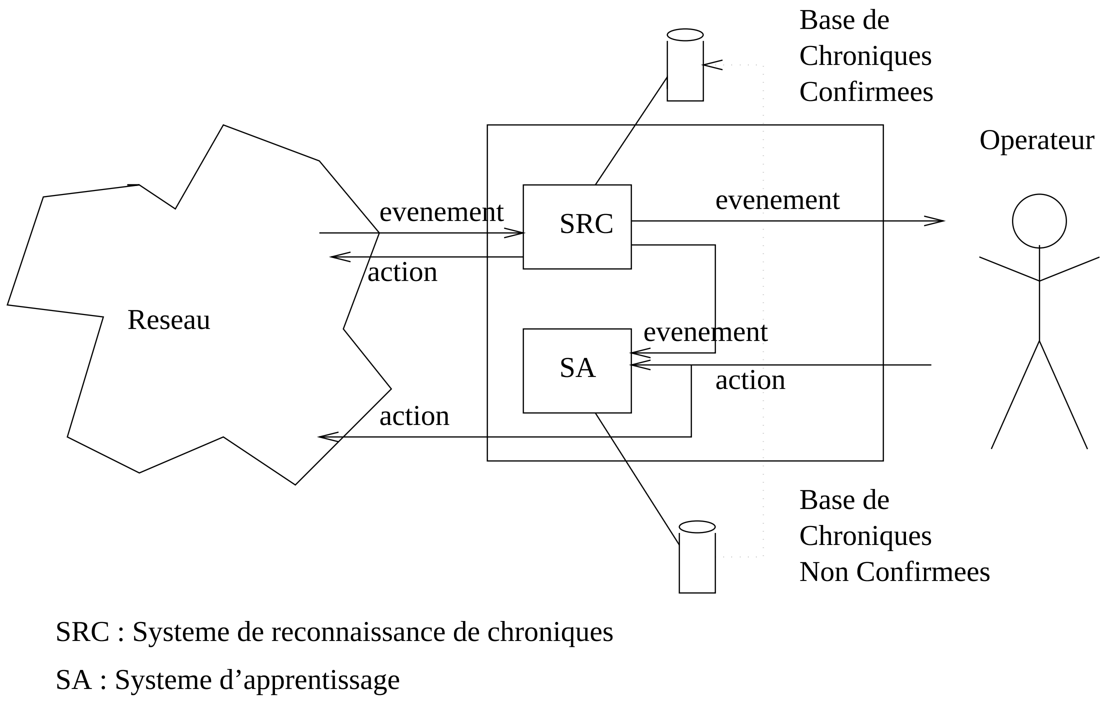


FIG. 3.2 - *La structure de l'assistant*

when recognized {**do** `STATE-CHECK(?connection, e3);` }}

pourrait se traduire en :

```
(If t1
    (Bel (t1 (RECEIVED-EV-REPORT EV1 ?connection)))
    (If (+ t1 2)
        (Bel ((+ t1 2) (RECEIVED-EV-REPORT EV2 ?connection)))
        (Do  (+ t1 7) (STATE-CHECK ?connection))
    )
)
```

L'opérateur, peut alors, en consultant l'état de l'assistant, notamment la liste de ses engagements, savoir ce que l'assistant va faire étant données les notifications reçues.

Quant au système d'apprentissage, utilisons les notions d'état mental pour spécifier notre système d'apprentissage et le différencier d'autres modèles tels que celui de

Valiant [Val84]. En effet dans ce dernier, à partir d'échantillons de cas non bruités, on prétend classer presque sûrement tous les cas, même ceux qui n'ont pas été déjà rencontrés. Dans notre système, à partir de cas bruités, on prétend classer presque sûrement des cas déjà rencontrés, identifiant ainsi la part de bruit dans l'échantillon d'apprentissage. A priori, quand le système rencontre un cas inédit, il ne doit surtout pas prendre de décision à la place de l'opérateur !

Reprenant alors à notre compte les définitions de Shoham pour les adapter à notre système, il devient alors nécessaire de distinguer croyances et connaissances (cette distinction n'étant pas faite dans le modèle de Valiant) : notre système doit acquérir des croyances et essayer de les transformer en connaissances pour pouvoir les utiliser. La spécificité de la modalité "connaissance" (notée K) par rapport à la croyance est la suivante, également signalée par Shoham :

```
K[t](a, fbf) -> fbf
```

Les fbf peuvent être la valeur de justesse d'une chronique. Ainsi, les chroniques de la base non confirmée constituent les (des) croyances de l'agent, et les chroniques de la base confirmée constituent (la plupart de) ses connaissances.

Comment modéliser le passage d'une croyance à l'état de connaissance? Pour cela nous allons introduire un opérateur ternaire, la Confiance (notée **Trusts**).

Trusts (**a**, **b**, **fbf**) signifie que l'agent **a** fait confiance à l'agent **b** à propos de la croyance de **b** à la **fbf** (pour alléger la syntaxe de cet opérateur et étant donné que la confiance n'affectera que des croyances, nous avons volontairement omis l'opérateur de croyance).Trusts est un opérateur ternaire de KD4. On a donc les propriétés suivantes :

```
Trusts (a, b, f1) et Trusts (a, b, f1 -> f2) -> Trusts (a, b, f2)

Trusts (a, b, f1) -> Trusts (a, b, Trusts (a, b, f1))

Non (Trusts (a, b, (f1 et Non (f1))))
```

Ajoutons alors l'axiome suivant :

```
(B (a, fbf) et Trusts (a, b, fbf) ) -> K (a, fbf)
```

Ainsi, si un agent **a** croit en une fbf, et s'il fait confiance à un agent **b** sur la même valeur de la fbf, il considère alors qu'il connaît la fbf. Trois cas particuliers sont alors intéressants à examiner :

- `Trusts (a, a, fbf)` : il s'agit du cas où l'agent a confiance en lui-même pour une fbf donnée. Cela correspond dans notre système à la confirmation automatique d'une chronique, quand celle-ci a atteint le seuil de confiance préréglé.

- `Trusts (a, opérateur, fbf)` : on considérera qu' on a :

  `B (a, Trusts (a, opérateur, fbf)) -> K (a, Trusts (a, opérateur, fbf)).`

  En effet, cela correspond à la confirmation "manuelle" d'une chronique par l'opérateur. La réception de cette action de la part du système peut être interprétée comme un témoignage de confiance et doit conduire systématiquement au passage à la base confirmée.

- `Trusts (a, b, fbf)` où (b <> a) et (b <> opérateur) : on rejoint ici le cadre des "collaborative learning interface agents" chers à Pattie Maes [LMM94]. L'acquisition de la confiance en un autre agent s'effectue par un processus de test des autres agents et d'évaluation de leurs réponses dans des cas connus. Une confiance en un agent peut ne pas être totale et dépendre du domaine d'expertise des deux agents concernés. Ainsi, un assistant à un opérateur de supervision de réseau pourra constater qu'un assistant d'un autre superviseur apprend des chroniques semblables dès que les deux superviseurs sont concernés par des problèmes provenant d'une infrastructure de réseau commune. Il n'accordera donc sa confiance que dans le cas de pannes concernant cette infrastructure.

On supposera par la suite qu'on dispose du système de reconnaissance de chroniques (en abrégé SRC), et on décrira le fonctionnement envisagé pour le système d'apprentissage (en abrégé SA).

3.2 Le système d'apprentissage

Il était relativement difficile d'utiliser les algorithmes proposés par exemple par Maes ou Schlimmer. En effet, Pattie Maes ne raisonnait pas sur la succession d'événements reçus et son algorithme ne tenait pas du tout compte d'aspects temporels. Quant à l'inférence grammaticale, technique employée par Schlimmer, elle permet effectivement de raisonner sur des séquences d'événements, mais il n'y a pas de distinction faite entre un événement et une action. Nous avons finalement décidé de proposer nos propres algorithmes en nous inspirant de travaux antérieurs. Nous sommes conscients qu'on peut encore améliorer ces algorithmes, ou qu'il aurait peut-être été judicieux de poursuivre une véritable étude bibliographique de travaux existants en apprentissage automatique et adaptables à notre problématique. Cela nous aurait néanmoins demandé beaucoup de temps et nous aurait éloigné de notre propos principal qui était de trouver des techniques orientées agent en supervision de réseaux. On constatera en tout cas dans la dernière partie de ce document que nos algorithmes se comportent de façon satisfaisante s'ils sont utilisés dans un contexte favorable, c'est-à-dire qui ne cumule pas par exemple la simultanéité de pannes et la multiplicité de causes possibles pour une même panne.

Comme on l'a déjà spécifié dans le paragraphe précédent, notre apprentissage ne rentre pas parfaitement dans le cadre classique défini par Valiant. Néanmoins le paragraphe 3.2.1 montre des résultats d'apprenabilité que nous avons obtenus, notamment grâce à la collaboration de Richard Nock.

3.2.1 Aspects théoriques de l'apprentissage de chroniques

Ce travail [EN96] a été effectué avec Richard Nock (qui poursuit actuellement une thèse au LIRMM sur les problèmes d'apprenabilité). Il s'agit pour nous d'étudier plus théoriquement la possibilité d'apprentissage des chroniques. Pour ce faire, nous nous basons sur le modèle d'apprenabilité de Valiant [Val84]. Selon lui, un concept est dit "apprenable", s'il est possible d'exhiber un algorithme polynomial, qui étant donné une probabilité de distribution des cas, est capable d'apprendre des instances de ce concept avec une bonne approximation et un faible taux d'erreur.

Nous montrons que dans le cas où il n'y a pas d'entremêlement d'événements (l'occurence simultanée de pannes pouvant conduire à la réception de notifications qu'il faut savoir attribuer à la bonne panne), les chroniques sont apprenables grâce à l'algorithme implémenté. La raison en est que sans entremêlement, il est possible de transformer une chronique en un monôme monotone, et l'apprentissage des monômes monotones est apprenable avec ce type d'algorithme [KLPV87a] [KLPV87b].

Par contre, en cas d'entremêlement, le premier algorithme présenté est insuffisant. Nous avons donc conçu un second algorithme qui gère plus spécifiquement le problème de l'entremêlement.

Une autre idée consiste à transformer l'oracle "passif", qui se contente d'émettre les événements et les actions qui y correspondent, en oracle "actif", qui répond aux interrogations de l'assistant. Nous montrons que la simple action de confirmation "manuelle" de chronique, bien que permettant d'accélérer le processus d'apprentissage, ou d'éviter par moments certains cas d'entremêlement, ne suffit pas pour résoudre ce problème. Nous montrons qu'avec un oracle, qui étant donné une suite d'événements et une action, répond "oui" ou "non" suivant que lesdits événements déclenchent ou non l'action, le problème d'apprentissage avec entremêlement devient apprenable au sens de Valient.

Or il ne s'agit pas, à des fins théoriques, de soumettre l'opérateur de supervision à un interrogatoire en règle ! C'est le contraire du but recherché ! C'est à ce moment qu'il devient intéressant de permettre la coopération entre plusieurs assistants, les questions n'étant donc pas adressées à l'opérateur, mais à un autre assistant. Nous verrons plus loin dans ce document, les protocoles envisagés pour permettre une telle coopération.

Nous montrons par ailleurs que même en disposant d'oracles puissants, il est difficile d'apprendre des chroniques quand plusieurs suites d'événements différents peuvent mener à une même action.

Nous constaterons (et nous tenterons de voir pourquoi) qu'en pratique il paraît difficile de lutter à la fois contre le problème d'entremêlement et celui de l'apprentissage de chroniques différentes déclenchant la même action. C'est pourquoi nous proposerons dans ce qui suit deux algorithmes dont chacun traite plus particulièrement un des problèmes. Il faudra alors savoir choisir l'algorithme adéquat en fonction des données à traiter (le premier algorithme traitera mieux les données où les actions peuvent avoir plusieurs prémisses ; le second sera plus adapté en cas d'entremêlement).

Les démonstrations effectuées par Richard Nock se trouvent en annexe de ce rapport, sous la forme d'un rapport de recherche du LIRMM.

3.2.2 Définitions préliminaires

Quelques définitions et hypothèses pour commencer :

- Soit E l'ensemble des événements : on suppose que cet ensemble est fini et "relativement restreint", c'est-à-dire qu'il est constitué d'un nombre fini d'attributs du domaine qui peuvent avoir un nombre fini de valeurs. Sans cette hypothèse, il sera a priori difficile d'obtenir des chroniques suffisamment génériques et on risque de générer un nombre trop grand de chroniques. On pourra par la suite, en fonction des résultats de l'expérimentation et des spécificités du domaine étudié, essayer de relaxer en partie cette contrainte.

- Soit A l'ensemble des actions. A contient un élément particulier : le "silence", qui ne correspond à aucune action "tangible" de l'opérateur de supervision, et qu'on utilisera pour créer des chroniques particulières : les filtres.

Une chronique est composée au départ d'une suite d'événements datés et d'une action. Nous verrons plus loin le traitement des assertions et l'ajout de contraintes temporelles.

On suppose qu'on dispose de deux bases de chroniques (éventuellement vides au départ) :

- la base dite "confirmée", qu'on notera BC : ce sont les chroniques qui sont suivies par le système de reconnaissance de chroniques (SRC). Elles sont supposées fiables et indiscutables ;

- la base dite "non confirmée", qu'on notera BNC : ce sont les chroniques engendrées par le système d'apprentissage (SA). Elles vont être "affinées" et généralisées au fur et à mesure de l'apprentissage, en étant débarrassées d'événements parasites et en ayant des contraintes temporelles moins restreintes. Une fois atteint un certain de degré de "maturité", elles pouront être "confirmées" (passage de la BNC à la BC).

Dans les paragraphes suivants, nous allons présenter deux algorithmes d'apprentissage de chroniques. Les principes des deux algorithmes sont semblables, mais le premier tentera d'apprendre même des chroniques qui déclenchent une même action, alors que le second sera plus adapté à traiter l'entremêlement d'événements. En tout cas l'apprentissage se déroulera toujours en trois étapes :

1. la création d'une chronique, à partir des événements reçus et des actions déclenchées par l'opérateur ;

2. l'évaluation de la chronique, qui consiste à comparer la chronique créée avec celles qui se trouvent déjà dans la BC et la BNC. Ceci aura pour conséquences éventuelles des modifications de chroniques de la BNC ;

3. la confirmation d'une chronique, qui consiste à transférer les chroniques "mûres" de la BNC vers la BC.

3.2.2.1 La première méthode

3.2.2.1.1 Création d'une chronique La SA enregistre le flot d'événements reçus par le système de supervision. Une chronique est générée :

– si l'opérateur de supervision décide d'une certaine action : la chronique sera constituée du flot d'événements reçus associé à l'action ;

– s'il ne se passe rien (ni réception d'événements, ni action de l'opérateur) pendant un certain temps. Ce seuil doit être paramétré au départ. On pourra éventuellement le faire dépendre du dernier type d'événement reçu (pour certains événements, la temporisation peut être longue, cette temporisation peut signifier l'attente de l'arrivée ou non d'un événement). La chronique générée est constituée du flot d'événements suivis de l'action "silence".

Si l'action provient du SRC, les événements correspondant à la chronique reconnue sont effacés de la mémoire : en effet cela veut dire que la chronique correspondante existe déjà, et qu'il n'est pas nécessaire de la réapprendre.

3.2.2.1.2 Evaluation de la chronique Il s'agit maintenant de voir si la chronique obtenue doit être ajoutée à la BNC ou non. Cette vérification peut donner elle-même lieu à la génération d'autres chroniques.

On a auparavant besoin de définir deux opérateurs : l'inclusion ($\subseteq$) et la différence ($\backslash$) entre deux chroniques :

Soient N et M deux entiers et soient C = $(e(v_i, t_i), a)_{i=1...N}$ et D = $(e(v_j, t_j), b)_{j=1...M}$ deux chroniques. On a, par définition :

C $\subseteq$ D $\Leftrightarrow$ $(a = b)$ et $e(v_i)_{i=1...N}$ "sous-mot" de $e(v_j)_{j=1...M}$

La vérification de l'inclusion est une opération en O(M). De plus on ne vérifie pas l'égalité des dates des événements.

Maintenant supposons qu'on dispose de :

- de deux chroniques C et D telles que C $\subseteq$ D,
- et d'une fonction heuristique h, qui à un événement e(v,t) de C associe un événement e(v,t') de D.

On a alors la définition suivante :

D $\setminus$ C = ({ e $\in$ D / $\forall$ f $\in$ C, e $\neq$ h (f) },"silence")

L'opération de différence nous permettra dans notre algorithme de créer les chroniques filtres, d'où le positionnement de l'action à "silence". La complexité du calcul de cette différence dépend essentiellement du choix de la fonction heuristique h, qui a pour vocation d'aider à trouver le sous-mot de D le plus "proche" de C en tentant de minimiser les différences d'intervalles de temps entre les événements. On peut exhiber des heuristiques relativement satisfaisantes en O(n^2), n étant le nombre d'événements de la chronique D, mais nous nous sommes contentés de choisir le sous-mot constitué des événements rencontrés pour la première fois dans D.

Voici alors l'algorithme 1 d'évaluation d'une chronique C dont l'action se nomme a :

Algorithme 1 : ÉVALUER(C)

% C est une chronique avec comme action a
début
  $Trust(C) := 1$
  pour *tout* $C' \in$ *la BC tel que action*$(C') = a$ **faire**
    switch C **do**
      case $C \subseteq C'$: exit
      case $C \supset C'$: creer-et-evaluer-filtre $C'' := C \setminus C'$
      case $C \neq C'$: continue
    end
  fin
  pour *tout* $C' \in$ *la BNC tel que action*$(C') = a$ **faire**
    switch C **do**
      case $C \subset C'$
        ajouter C a la BNC (si pas deja fait)
        $Trust(C) := \max(Trust(C), Trust(C')) + 1$
        elargir les intervalles de temps de C en considerant C'
        enlever C' de la BNC; creer-et-evaluer-filtre $C'' := C' \setminus C$
      case $C \supseteq C'$
        creer-et-evaluer-filtre $C'' := C \setminus C'$; elargir les intervalles de temps de C' en considerant C
        $Trust(C')$++
      case $C \neq C'$: continue
    end
  fin
  ajouter C a la BNC
fin

3.2.2.1.3 La création et l'élargissement des contraintes temporelles Une autre opération que l'on considérera lors de l'évaluation de la chronique est la création et l'élargissement des contraintes temporelles. On n'envisagera que la création de contraintes de plus en plus lâches entre deux événements successifs, c'est-à-dire l' "élargissement" des intervalles de façon à recouvrir la contrainte de la chronique évaluée (un instant t est considéré comme un intervalle [t,t]). Mais rien n'interdit de penser que d'autres types de contraintes temporelles (par exemple entre chaque événement et le premier événement de la chronique, ou plus généralement un ordre partiel) puissent être plus adéquats. Or apprendre un ordre partiel entre événements revient à apprendre plusieurs ordres totaux. Ainsi :

A (B et C dans n'importe quel ordre) → D

équivaut à :

A B C → D

A C B → D

Donc l'ordre partiel ne pose pas de problème tant que l'on peut apprendre des chroniques différentes ayant un même champ "action".

L'algorithme 2 calcule les nouvelles contraintes temporelles, et retourne aussi la chronique-filtre en effectuant une différence. En effet le calcul des deux se fait dans le même algorithme pour être plus économique. On remarquera qu'on met aussi à jour les intervalles entre le dernier événement et l'action.

Algorithme 2 : CALCULE-CONTRAINTES(A)-ET-$C = B \setminus A$

% A est une chronique inclue dans B

début

$i := j := 1$

$binf := bsup := 0$

tant que *($i \leq$ taille(B))* **et** *($j \leq$ taille(A))* **faire**

si $B_i = A_j$ **alors**

borninf(predecesseur(A_j)) = min (borninf(predecesseur(A_j), $binf$))

bornsup(predecesseur(A_j)) = max (bornsup(predecesseur(A_j), $bsup$))

$binf :=$ borninf(B_i)

$bsup :=$ bornsup(B_i)

$j{+}{+}$

fin

else

$binf$ += borninf(B_i)

$bsup$ += bornsup(B_i)

ajouter-fin B_i a C

end

$i{+}{+}$

fin

si *$j >$ taille(A)* **alors**

pour *$k := i$ a taille(B)* **faire**

$binf$ += B_k

$bsup$ += B_k

ajouter-fin B_k a C

fin

fin

borninf(dernier(A)) = min (borninf(dernier(A), $binf$))

bornsup(dernier(A)) = max (bornsup(dernier(A), $bsup$))

$action(C)$ = silence

fin

3.2.2.1.4 L'algorithme d'évaluation des filtres L'évaluation des filtres (chroniques qui ont pour action "silence"), ressemble à l'algorithme d'évaluation des autres chroniques, à la différence près que nous avons privilégié la création de filtres longs, plus spécifiques. En fait, à chaque cas d'inclusion détecté, nous avons gardé la chronique la plus longue (c'est l'opération inverse qui est effectuée dans le cas des autres chroniques). L'idée était d'éviter ainsi la création de filtres trop "courts" qui perdraient alors toute signification. Si cela permet d'éviter de perturber le suivi des autres chroniques, nous n'avons en contrepartie pas obtenu de résultats satisfaisants pour le moment. Voici en tout cas l'algorithme utilisé :

Algorithme 3 : EVALUER-FILTRE(C)

% C est une chronique avec comme action *silence*

début

  $Trust(C) := 1$

  si $\exists\, C' \in$ *la BC tel que (action*(C') $=$ *silence)* **et** $C \subseteq C'$ **alors** : exit

  pour *tout* $C' \in$ *la BNC tel que action*(C') $=$ *silence* **faire**

    switch C **do**

      case $C \subset C'$: exit

      case $C \supseteq C'$

        ajouter C a la BNC (si pas deja fait)

        $Trust(C) := \max(\ Trust(C), Trust(C')) + 1$

        elargir les intervalles de temps de C en considerant C'

        enlever C' de la BNC

      case $C \neq C'$: continue

    end

  fin

  ajouter C a la BNC

fin

3.2.2.1.5 Traitement des assertions Le système de reconnaissance de chroniques d'IxTeT, afin de traiter les prédicats "holds", gère pour chaque chronique une liste d'"événements interdits". Si l'un d'eux se produit , l'assertion n'a pas été respectée et la chronique n'est pas reconnue. Autrement, la chronique est reconnue.

Pendant le processus d'apprentissage, suivant que la valeur d'un domaine qui doit être protégé est modifiée ou non, l'opérateur fera deux actions différentes, dont l'une sera (dans le cas du non-respect de l'assertion) probablement le "silence", ce qui correspond à la non-reconnaissance de la chronique. Le SA générera donc deux chroniques différentes : une pour le respect d'une assertion, et l'autre pour son non-respect. Bref une chronique comportant une assertion a été apprise comme étant en fait deux chroniques, ceci évidemment si les deux cas ont été rencontrés durant la supervision.

3.2.2.1.6 Confirmation de la chronique Il s'agit donc du passage d'une chronique de la BNC à la BC. Deux possibilités sont actuellement offertes :

- la décision de la confirmation provient de l'opérateur : celui-ci peut à tout moment visualiser l'ensemble des chroniques (rangées par ordre de confiance décroissante) de la BNC, et en confirmer une par un simple clic de la souris. Il n'est pas interdit d'envisager que l'opérateur puisse, s'il le désire, modifier la chronique (par exemple modifier les contraintes temporelles) avant de la confirmer ;

- la confiance en une chronique dépasse un certain seuil à déterminer à l'avance : le passage à la BC pourrait se faire automatiquement ou être suggéré à l'opérateur.

On peut également envisager que l'opérateur puisse supprimer une chronique de la BNC, parce qu'elle correspond par exemple à une erreur de manipulation.

3.2.2.1.7 Les invariants à préserver Une des propriétés que doit respecter l'algorithme d'apprentissage est le fait qu'on ne doit pas retrouver dans la base non confirmée deux chroniques dont l'une pourrait être incluse dans l'autre. Or cette propriété est trivialement vérifiée par l'algorithme, puisque tous les tests d'inclusion possible entre chroniques sont effectués, et en cas de détection d'inclusion seule la chronique la plus petite est conservée. Toutes les autres chroniques de la base non confirmée qui la contiennent sont éliminées. Cette propriété est bien entendu conservée dans la base confirmée.

Par exemple, supposons qu'on dispose des deux chroniques suivantes :

- chronique C_1 : a b d $\rightarrow \alpha$

- chronique C_2 : a b c $\rightarrow \alpha$

Si on évalue la chronique C : a b $\rightarrow \alpha$, on a C $\subseteq C_1$ et C $\subseteq C_2$: il est donc nécessaire de ne pas s'arrêter à la première inclusion, et remplacer les chroniques C_1 et C_2 par C dans la base non confirmée.

Dans la base confirmée, une autre propriété doit être vérifiée : il ne faut pas qu'une même séquence d'événement puissent donner lieu au déclenchement de plusieurs actions. Ceci est respecté grâce au mécanisme de maintien de cohérence.

3.2.2.1.8 Le maintien de cohérence dans la base confirmée Dans le contexte de l'apprentissage automatique et plus généralement de l'acquisition de connaissance, il est nécessaire de vérifier la cohérence de la base de connaissance obtenue. Il est possible de disposer de mécanismes qui vérifient que l'ajout d'une nouvelle connaissance ne contredit pas celles déjà existant dans la base : ce sont tous les TMS (truth maintenance systems) [Kle86].

En ce qui concerne notre système, nous faisons une vérification à chaque confirmation d'une chronique. Le programme détecte si, étant donnée la suite d'événements de la chronique à confirmer, on ne declenche pas déjà une autre action. Réciproquement, on détecte si avec les suites d'événements des chroniques de la base confirmée on ne déclenche pas l'action de la nouvelle chronique. Ces deux tests reviennent à détecter d'éventuelles relations de sous-mots entre les chroniques de la base confirmée et la chronique à confirmer, indépendamment du champ "action" des chroniques en question. Quand de telles relations sont détectées, le système avertit l'opérateur, en lui signalant le nombre d'incohérences détectées, et lui propose d'annuler la confirmation, ou de l'efffectuer tout de même. En effet il est envisageable qu'un même événement puisse être attribué à plusieurs chroniques.

3.2.2.1.9 Conclusions sur la première méthode La possibilité de créer plusieurs chroniques déclenchant la même action a rendu concevable la création de filtres (plusieurs séquences peuvent engendrer l'action "silence"), la gestion simple du préordre entre les événements, et la quasi-inutilité de création d'assertions "holds". En revanche, et on le verra dans la partie consacrée aux résultats, les performances de l'algorithme se détériorent en présence d'entremêlements de séquences. D'autre part, les résultats de l'apprentissage dépendent de l'ordre de création des chroniques.

La deuxième méthode, présentée dans le paragraphe suivant, tente de pallier ces inconvénients, mais on en verra également les contreparties.

3.2.2.2 La deuxième méthode

Un des grands problèmes rencontrés lors de l'apprentissage de chroniques est l'entremêlement des événements conduisant à des actions différentes. La première méthode conduit à la création de chroniques "tronquées" puisqu'on ne s'intéresse pas aux événements survenus avant l'action précedente. Une solution serait donc de mémoriser les k derniers événements pour chaque action, indépendamment du déclenchement d'autres actions. Mais cela pose le problème suivant : du fait qu'on risque de garder des événements sans rapport avec l'action, on risque de trouver difficilement des relations d'inclusions. La solution que nous proposons est de substituer à la relation d'inclusion celle de "plus grand sous-mot commun", autrement dit une intersection entre deux chroniques qui conserve l'ordre et la fréquence d'occurence des événements. Ainsi il sera possible à chaque déclenchement d'action de généraliser la chronique correspondante en éliminant les événéments inutiles. Mais l'adoption de ce nouvel algorithme ne sera possible que si on suppose qu'on ne peut pas avoir deux suites d'événements différentes qui conduisent à une même action. Sinon, on risque d'obtenir à nouveau des chroniques tronquées. En particulier, il sera impossible de créer des chroniques-filtre avec ce procédé, sous peine d'obtenir très rapidement des chroniques vides.

Par exemple, supposons qu'on dispose des deux chroniques suivantes :

- chronique C_1 : a b c d e f g $\rightarrow \alpha$
- chronique C_2 : h b e j f w $\rightarrow \alpha$

Le plus grand sous mot commun est le plus grand C tel que C $\subseteq C_1$ et C $\subseteq C_2$: c'est donc b e f $\rightarrow \alpha$ dans ce cas.

D'autre part, ce nouveau procédé peut avoir un intérêt supplémentaire : on peut aussi considérer une action comme un événement comme les autres. Ainsi on pourra obtenir des chroniques dans lesquelles on déclenche une action non seulement en fonction des événements reçus, mais aussi des actions déclenchées précédemment. Ainsi dans les données que nous avons eu à étudier, on pourra peut-être caractériser la disparition d'une alarme à partir du diagnostic précédent qui était l'apparition de la même alarme.

3.2.2.2.1 Le calcul du plus long sous-mot commun Le calcul du plus long sous-mot commun n'est pas, contrairement aux apparences, une tâche triviale. Une mauvaise implémentation peut vite s'avérer coûteuse en temps de calcul ou volumineuse en mémoire consommée (par exemple dans le cas d'une implémentation récursive qui a le mérite d'être intuitive). Beaucoup de travaux ont été menés pour réduire ce coût, sachant qu'on peut exhiber relativement facilement un algorithme en $O(n^2)$ (le calcul du plus long sous-mot commun à plus de deux mots est NP-difficile). Nous avons choisi un algorithme classique et efficace proposé par Hunt et Szymanski [HS77]. Voici le code Smalltalk [2] correspondant :

```
longestCommonSubsequence: anOrderedCollection
  | aStart aFinish bStart bFinish matchVector bMatches thresh links |
  "This method computes the longest common subsequence in self and
anOrderedCollection.
  It uses the algorithm described in A Fast Algorithm for Computing
Longest Common Subsequences, CACM, vol.20, no.5, pp.350-353, May 1977,
with a few minor improvements to improve the speed."

  "First we prune off any common elements at the beginning or end."
  aStart := 1.
  aFinish := self size.
  bStart := 1.
  bFinish := anOrderedCollection size.
  matchVector := Array new: self size.
  [((aStart <= aFinish) and: [bStart <= bFinish])
  and: [(self at: aStart) = (anOrderedCollection at: bStart)]]
    whileTrue: [matchVector at: aStart put: bStart.
      aStart := aStart + 1.
```

2. trouvé dans les archives Smalltalk de Manchester
(URL: http://st-www.cs.uiuc.edu/archive/)

```
    bStart := bStart + 1].

"now the end"
[((aStart <= aFinish) and: [bStart <= bFinish])
and: [(self at: aFinish) = (anOrderedCollection at: bFinish)]]
  whileTrue: [matchVector at: aFinish put: bFinish.
    aFinish := aFinish - 1.
    bFinish := bFinish - 1].

"Now compute the equivalence classes of positions of elements"
bMatches := Dictionary withPositionsOf: anOrderedCollection
  inInterval: (bStart to: bFinish).

thresh := SortedCollection new sortBlock: [:x :y | x < y].
links := Array new: ((aFinish - aStart + 1) min: (bFinish - bStart + 1)).

aStart to: aFinish do: [ :i || ai |
  ai := self at: i.
 (bMatches includesKey: ai)
   ifTrue:
     [(bMatches at: ai) reverseDo:
       [ :j || k link |
         k := thresh replaceNextLargerWith: j.
         k notNil ifTrue:
           [link := LinkElement new.
            k > 1 ifTrue: [link nextLink: (links at: k - 1)].
            link value: (Array with: i with: j).
            links at: k put: link]]]].

thresh size > 0
  ifTrue:
    [| link |
      link := links at: thresh size.
```

```
[matchVector at: (link value at: 1) put: (link value at: 2).
link nextLink notNil] whileTrue: [link := link nextLink]].

^matchVector
```

Une remarque importante est que le plus long sous-mot commun à deux mots n'est pas nécessairement unique, et qu'il faut donc faire un choix. Dans le même ordre d'idée, rien n'indique que ce soit toujours le plus long sous-mot qui soit le plus pertinent. C'est ce qui fait le caractère heuristique de notre algorithme tel qu'il est exposé dans la section suivante.

3.2.2.2.2 Le nouvel algorithme d'évaluation des chroniques L'algorithme d'évaluation de chroniques présenté ci-dessous reprend en grande partie les principes du premier algorithme. Il s'agit ainsi d'un algorithme incrémental où l'on cherche à éliminer les événements non pertinents des chroniques. On calcule la plus longue sous-séquence commune (LCS) entre la séquence d'événements de la chronique à évaluer et celle de la chronique C' (unique si elle existe, c'est une propriété de l'algorithme) se trouvant dans la base non confirmée et qui déclenche la même action. Cette nouvelle sous-séquence remplace alors C' avec une confiance augmentée.

Algorithme 4 : EVALUER-2(C)

% C est une chronique qui a pour action α

début

$Trust(C) := 1$

si $\exists$ $C' \in$ *la BC tel que* $action(C') = \alpha$ **alors** exit;

si $\exists$ $C' \in$ *la BNC tel que* $action(C') = \alpha$ **alors** $C'' := LCS(C, C')$;

$action(C'') := action(C)$;

$Trust(C'') := Trust(C') + 1$

enlever C' de la BNC;

ajouter C'' a la BNC; **else** ajouter C a la UCB;

fin

Remarquons que la création et l'élargissement des contraintes temporelles pourront se faire lors du calcul du plus long sous-mot commun.

Les autres étapes de la méthode : confirmation et maintien de cohérence sont identiques à celles de la première méthode. En revanche il est clair qu'étant donné qu'on ne peut apprendre plusieurs chroniques déclenchant la même action, il sera inutile de créer et évaluer des filtres (on en aurait tout au plus un). Il sera de plus nécessaire d'avoir une méthode pour créer des assertions "holds". Une possibilité est de créer des "holds" lors de la détection d'incohérence lors de la confirmation de chronique : si l'opérateur le souhaite, le système peut effectuer une différence entre les chroniques incohérentes, et transformer les événements issus de cette différence en "holds".

3.2.3 Un exemple

Illustrons le comportement de notre système d'apprentissage avec un exemple. Nous choisissons ici le premier algorithme d'apprentissage, et nous ne tiendrons pas compte ici de la création et l'élargissement des intervalles temporels pour rester simple. Les événements seront représentés par des lettres latines, et les actions par des lettres grecques.

Supposons que nous disposons à un moment donné des bases de chroniques suivantes :

Base Non Confirmée :

1. a b c d $\rightarrow \alpha$. Confiance: 1.
2. a b c e $\rightarrow \alpha$. Confiance: 1.
3. a b c f $\rightarrow \beta$. Confiance: 2.

Base Confirmée :

1. a b c g $\rightarrow \gamma$. Confiance: 3.

Supposons alors qu'à la réception de la suite d'événements a b c, l'opérateur déclenche l'action α. La chronique a b c $\rightarrow \alpha$ est alors créée. Ensuite l'algorithme d'évaluation cherche d'abord une relation d'inclusion entre la nouvelle chronique et celle de la base confirmée. Il n'en trouve pas (rappelons que les actions doivent être égales). Elle passe alors à la base non confirmée. Il détecte d'abord une inclusion entre la nouvelle chronique et la première de la BNC. Elle remplace donc celle-ci par la nouvelle, et incrémente la confiance de la nouvelle chronique, qui passe donc à 2. On crée également une chronique-filtre, résultant de l'opération de différence. La chronique-filtre est immédiatement ajoutée à la BNC, puisqu'aucun filtre n'existe déjà dans la base. L'algorithme détecte ensuite une nouvelle relation d'inclusion entre la nouvelle chronique et la deuxième chronique de la BNC. On fait donc les mêmes opérations que précédemment. Le nouveau filtre créé est évalué à son tour, et comme il n'est pas inclu dans le filtre précédent, il est aussi ajouté à la BNC. La confiance de la nouvelle chronique est maintenant égale à 3. Comme le seuil de confirmation est égal à 3 (on peut le déduire de la confiance de la chronique se trouvant déjà dans la BC), la chronique est alors confirmée automatiquement (si bien sûr l'opérateur le désire). A ce moment-là, le système de maintien de cohérence détecte une relation de sous-mot entre la nouvelle chronique et celle de la BC (ce système ne doit pas comparer les champs “action”), et en avertit l'opérateur, qui peut décider de confirmer tout de même la chronique, ou de la rejeter. En supposant qu'il l'accepte, voici alors le nouvel état des bases de chroniques :

Base Non Confirmée :

1. d $\rightarrow$ *silence*. Confiance: 1.

2. e $\rightarrow$ *silence*. Confiance: 1.

3. a b c f $\rightarrow \beta$. Confiance: 2.

Base Confirmée :

1. a b c g $\rightarrow \gamma$. Confiance: 3.

2. a b c $\rightarrow \alpha$. Confiance: 3.

3.3 Apprentissage multi-assistants

L'idée sous-jacente est d'essayer de bénéficier de la présence de plusieurs assistants pour profiter de leur expérience acquise. Cela peut non seulement améliorer l'apprentissage, mais aussi permettre à un nouvel assistant de ne pas partir de zéro, et être opérationnel aussi vite que possible. On peut ainsi imaginer un réseau de superviseurs de réseaux appartenant à un constructeur donné, manipulant donc le même type de messages. L'installation d'un nouveau superviseur donnerait lieu à son insertion dans ce réseau, permettant ainsi la collaboration de cet assistant avec les autres. Cela dit, les comportements des différents opérateurs de supervision ne sont pas identiques. On ne peut donc pas se contenter de recopier les bases de chroniques des autres assistants. Il s'agit donc d'établir des relations de confiance avec des assistants au comportement proche, et tenir seulement compte d'avis exprimés majoritairement par les assistants dignes de confiance.

Comme nous l'avons dit précédemment, ce qui suit est grandement inspiré des travaux de Pattie Maes [Mae94] [LMM94].

3.3.1 Les protocoles utilisés

On peut découper la coopération en quatre phases bien distinctes qui correspondent à autant de protocoles simples d'interaction. Ces protocoles sont constitués d'échanges de messages au format proche de CMIS que nous avons déjà utilisés par ailleurs dans notre prototype.

- Le protocole de présentation : un nouvel assistant doit diffuser (protocole UDP) son adresse aux autres assistants. Ceux-ci mettent à jour leur liste d'assistants actifs (donc consultables à leur tour), et lui répondent en lui donnant leur adresse (nom de la machine + numéro de port). Ainsi, l'assistant n'enverra désormais ses requêtes qu'aux assistants actifs.

  Suite de messages échangés, A désignant un nouvel assistant, et $A_1 ... A_n$ les assistants déjà actifs (la syntaxe des messages sera décrite dans le chapitre suivant) :

  A à tous (broadcast) : **CREATE*darius.lune.lirmm.fr*1602** (l'assistant envoie le nom de sa machine et son no de port et se met désormais à son tour en attente de l'arrivée d'autres assistants. Cette requête ne respecte pas la syntaxe habituelle, la raison est que nous avons voulu condenser l'information en un seul message, plutôt que d'envoyer plusieurs diffusions successives, peu économiques en nombre de messages)

$A_1 ... A_n$ à A : **CREATE*1*0*** (demande de création d'un objet de type **Manager**. Le **0** désigne l'option de création d'objet en utilisant le *namebinding*. Le **1** désigne l'objet à partir duquel le nom va être déterminé)

A à A_i (pour tout i de 1 à n) : **SET*0*1.i** (A renvoie à chaque assistant le nom de l'objet qui le représente)

A_i à A : **SET*1.i*0*nomMachine:*picpoul.lirmm.fr*** puis **SET*1.i*0*port:*1604*** (A_i positionne chez A le nom de sa machine et son no de port)

A à A_i : **SET*1.i*picpoul.lirmm.fr** puis **SET*1.i*1604** (A confirme le positionnement de ces valeurs)

– Le protocole de requête : on définit un seuil dit de "consultation", inférieur au seuil de confirmation. Quand la valeur de confiance d'une chronique atteint ce seuil, l'assistant envoie une requête aux autres assistants contenant la séquence d'événements constituant la chronique. Les assistants voient s'ils peuvent déclencher une action avec cette séquence. Leur réponse consiste en trois parties :

  1. si la séquence reçue inclut ou est inclue dans la séquence déclenchante,
  2. dans quelle base (confirmée ou non confirmée) se trouve la chronique trouvée qui déclenche l'action,
  3. l'action déclenchée

  Parfois il se peut que chez un assistant, plusieurs réponses soient possibles pour une séquence donnée. L'assistant cherche d'abord une réponse qui provienne de la base confirmée, ensuite dont la séquence soit exactement égale, puis inclue, et enfin incluant.

  Suivant les réponses reçues et la confiance qu'a l'assistant en ses différents collègues, l'assistant qui a formulé la requête peut alors augmenter ou diminuer le degré de confiance de sa chronique. Cette façon de faire a le mérite d'être un moyen déterministe et relativement économique en nombre de messages échangés. Chez Pattie Maes, la requête est envoyée quand l'agent se trouve dans une situation dans laquelle elle n'a pas beaucoup confiance dans l'action à adopter. Nous verrons dans le paragraphe suivant comment se fait la prise en compte des réponses.

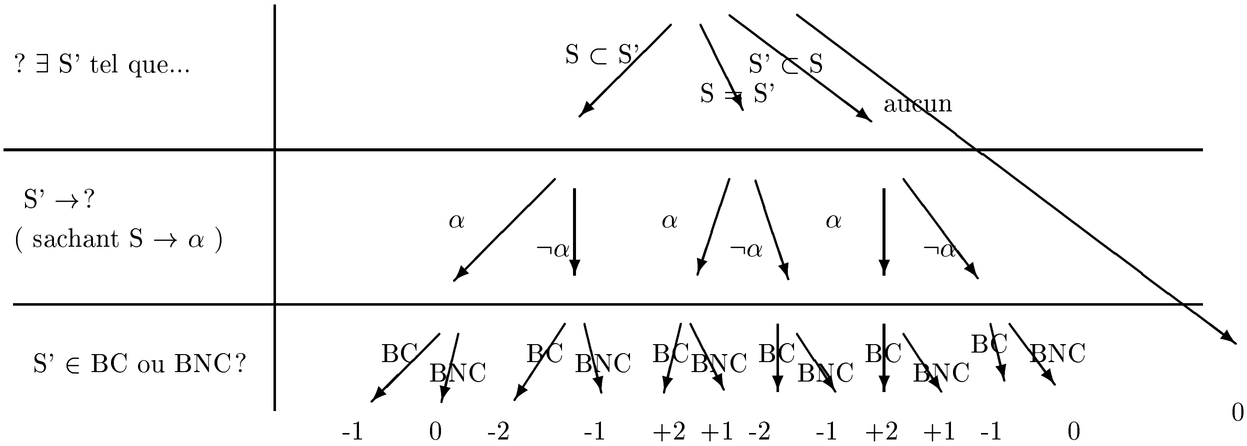
Suite de messages échangés :

A à A_i (pour tout i de 1 à n) : **ACTION*1*0*match:*C1!C2!C3*** (l'objet 1 est le module de coopération. Il va vérifier ce que l'assistant peut déclencher avec la séquence (C1, C2, C3)).

A_2 (par exemple) à A : **SET*1*B!C!<*** (la séquence est inclue dans une chronique de la base confirmée de l'assistant, et déclenche l'action C).

- Le protocole d'exploration : tous les assistants n'ont pas nécessairement le même comportement. Il ne s'agit donc pas d'accorder la même confiance à tous. Un assistant, au moment où le seuil de confirmation d'une chronique est atteint, envoie alors une requête de même nature que dans le protocole précédent, mais compare cette fois-ci la réponse avec l'action qu'il a lui-même apprise, augmente sa confiance à un assistant s'il a préconisé la même action et la diminue dans le cas contraire. Ainsi il pourra pondérer la valeur des réponses fournies lors du protocole de requête en fonction des valeurs de confiance. Nous envisageons d'utiliser également ce protocole pour diminuer le nombre de messages échangés lors du protocole de requête : l'assistant pourra limiter l'envoi de sa requête aux n assistants ayant les valeurs de confiance les plus élevées. Ainsi on peut voir apparaître des agents se "regroupant par affinité". Un problème qui peut apparaître est qu'une fois cette liste constituée, l'assistant risque de ne plus la renouveler, à moins de continuer de tester les nouveaux assistants. Or ces derniers, par définition, n'ont pas beaucoup de connaissances, et risquent en plus d'être submergés par ce type de requête alors qu'ils ne savent pas y répondre de façon pertinente. On peut alors envisager qu'un nouvel assistant ne puisse être testé qu'une fois ayant atteint un certain nombre de chroniques apprises. Il pourra alors se juger assez "expert" pour être utile aux autres assistants, et envoyer un avis de disponibilité. Chez Pattie Maes, il n'est pas précisé exactement à quel moment un assistant envoie une requête de type exploratoire.

- Le protocole de départ : il est très simple : quand un assistant doit quitter le réseau pour une raison quelconque (par exemple si son utilisateur ne désire pas qu'il coopère), il envoie une suite de messages destinés à l'enlever de la liste d'assistants actifs des autres assistants.



Arbre de notation des réponses à l'envoi d'une séquence S d'événements

FIG. 3.3 -

3.3.2 L'évaluation des réponses

Comment un assistant peut-il évaluer et tenir compte des réponses envoyées par les autres assistants ? Nous avons utilisé un système de notation des réponses qui tient compte de trois éléments :

1. suivant que la séquence est inclue, est égale ou inclut la séquence qui déclenche une chronique chez l'assistant consulté, sa réponse est encourageante ou non ;
2. suivant que l'action renvoyée est égale ou non à l'action envisagée,
3. suivant que la réponse provient de la base confirmée ou non confirmée.

L'assistant (fig. 3.3) donne alors à chaque réponse une note comprise entre -2 et +2. Il calcule ensuite la moyenne des notes. Si cette moyenne est positive, elle est arrondie et ajoutée au taux de confiance de la chronique. Comme nous choisirons en général un seuil de consultation égal au seuil de confirmation - 2, en cas de réponses très encourageante, la confiance de la chronique pourra atteindre directement le seuil de confirmation. Si en revanche la réponse est fortement négative, on peut même envisager d'éliminer la chronique de la base non confirmée.

La section suivante décrit MAGENTA, notre prototype de simulation d'une plateforme de gestion de réseau, et dans lequel a été intégré l'assistant.

Chapitre 4

MAGENTA : MAnaGEmeNT Application ou Multi-AGENT Assistant ?

MAGENTA est une plateforme développée en Smalltalk et Actalk [Bri88], initialement pour vérifier notre compréhension des normes ISO présentées au début de ce document en simulant une plateforme simplifiée de gestion de réseau. Elle était aussi destinée à la mise en place des expérimentations que nous devions mener. Or nous nous sommes rendus compte que ce prototype avait peut-être les caractéristiques nécessaires pour être une plateforme de développement de systèmes multi-agents. Nous verrons dans cette section la description de ce prototype, l'implémentation de notre assistant dans cette plateforme, mais aussi une application-jouet (l'algorithme distribué des n-reines) pour montrer l'utilisation possible de MAGENTA pour autre chose que la gestion de réseau.

4.1 La structure de MAGENTA

MAGENTA implémente le modèle gestionnaire-agent de l'OSI : il consiste en un programme Smalltalk qui peut jouer le rôle de gestionnaire ou d'agent suivant qu'il émet ou reçoit des requêtes de type CMIS, à travers des sockets TCP-IP, de la part d'autres programmes (par exemple d'autres programmes MAGENTA). Le prototype est divisé en quatre modules :

- Le gestionnaire de communication (GC), qui reçoit et envoie les requêtes CMIS. Elle implémente donc la couche de communication qui utilise l'interface socket ;

- Le gestionnaire d'objets (GO), qui traite les requêtes reçues en mettant à jour les données, qui consistent en un arbre de contenance d'objets gérés ;

- Le gestionnaire d'événements (GE), qui détermine si les modifications opérées sur les objets doivent donner lieu à l'envoi de notifications à d'éventuels gestionnaires ;

- L'application de gestion (AG), qui est spécifique au problème à traiter. Elle consiste à traiter les notifications reçues, et à composer des requêtes à destination d'éventuels agents. La composition des requêtes peut être faite par un utilisateur de MAGENTA ; dans ce cas, l'AG n'est qu'une interface utilisateur.

La figure 4.1 illustre bien qu'en fonction des événements reçus, différents protocoles d'interaction sont exécutés par les modules.

Les paragraphes suivants détaillent chacun des modules.

4.1.1 Le gestionnaire de communication

Comme nous l'avons dit précédemment, le gestionnaire de communication s'occupe de l'envoi et la réception des messages. Il s'initialise en se mettant en écoute et en chargeant un serveur de noms qui lui permet de retrouver l'adresse de ses correspondants à partir de leur nom. Grâce à ce serveur de noms, on peut également définir la portée d'éventuelles diffusions (broadcast). A la réception d'un message, suivant qu'il s'agit d'une requête, de la réponse à une requête ou d'une notification, le gestionnaire de communication fait appel dans le premier cas au gestionnaire d'objets et dans les autres cas à l'application de gestion. Dans le premier cas, il se met en attente de la réponse du gestionnaire d'objets pour la transmettre immédiatement à l'émetteur de la requête.

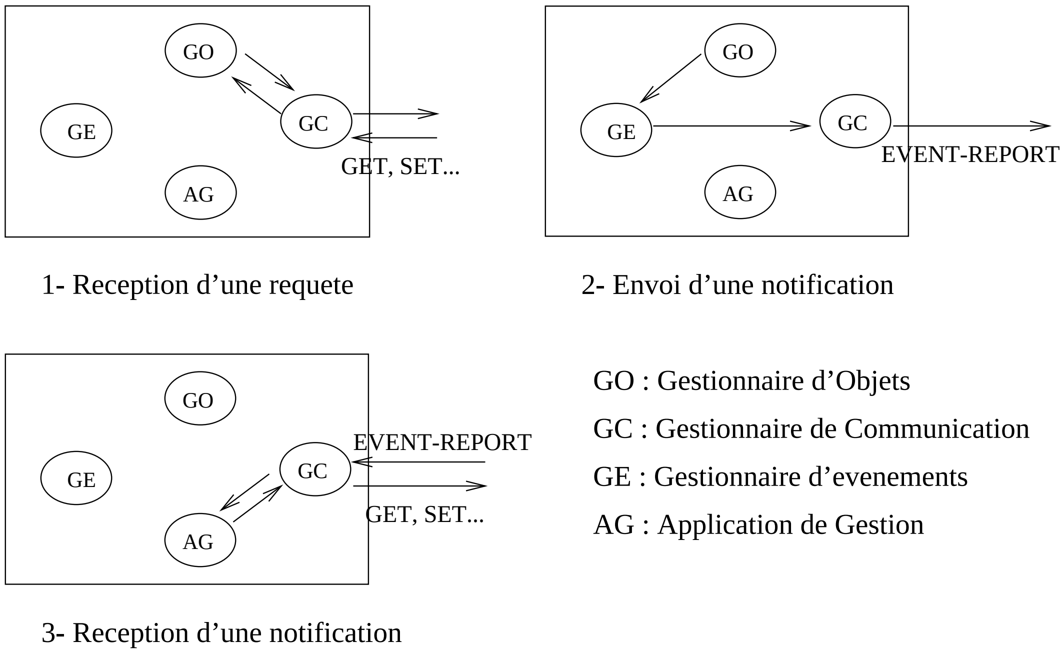


FIG. 4.1 - *La structure de MAGENTA*

4.1.2 Le gestionnaire d'objets

Il s'agit sans doute de la partie la plus intéressante du prototype, car c'est celle qui met le plus en oeuvre les normes ISO de gestion de réseau, et où on peut s'apercevoir de leur intérêt plus général pour la conception de systèmes multi-agents. Nous allons décomposer la description de ce module en deux parties :

1. l'interprétation des requêtes et leur répercussion sur la MIB (la base d'objets gérés, stockés sous la forme d'un *arbre de contenance*, et décrits sous la forme d'une hiérarchie de classes de type GDMO),

2. la description des classes elles-mêmes.

4.1.2.1 Le traitement des requêtes

Toutes les requêtes portent sur des objets "GDMO", qui correspondent dans MAGENTA à des instances de classes Smalltalk, dérivées de la classe GDMOObject créée à cet effet. Un attribut GDMO correspond évidemment à une variable d'instance, et une action à une méthode d'instance.

Les requêtes sont de 5 types : CREATE, GET, SET, ACTION et DELETE. Dans toutes les requêtes, nous avons utilisé le caractère * pour séparer les différents champs du message.

4.1.2.1.1 La création d'objets La requête CREATE est composée de deux champs : le premier peut soit indiquer le nom à attribuer à l'objet à créer (le nom absolu qui représente sa position dans l'arbre de contenance sous la forme d'une notation pointée), soit être positionné à zéro pour indiquer une attribution automatique du nom en fonction du lien de nommage. Le deuxième champ représente la classe de l'objet à instancier dans le cas du premier type d'attribution de nom, ou alors le nom de l'objet qui sera le père, dans l'arbre de contenance, de l'objet à créer dans le cas du nommage automatique. Nous avons déjà décrit la notion de lien de nommage dans le paragraphe 2.1.3.1.2, nous pensons néanmoins que décrire comment le gestionnaire d'objets effectue ce nommage permet de mieux expliquer cette notion. Voici donc ce que fait le gestionnaire d'objets lors de la réception de la requête **CREATE*0*1.0.3*** :

1. le gestionnaire d'objets obtient le nom de la classe de l'objet 1.0.3 et le lien de nommage défini pour cette classe, c'est-à-dire la classe (unique dans MAGENTA) de l'objet à créer, par exemple *Alarm*.

2. il consulte ensuite le nombre i d'objets de cette classe qu'il a déjà créés et insérés dans l'arbre de contenance comme fils de l'objet 1.0.3.

3. il incrémente ce nombre, en déduit le nom du nouvel objet (1.0.3.i+1), et le crée en l'insérant "à côté" de ses frères.

4. il renvoie le nom de cet objet au manager.

Grâce au nommage automatique, on peut ainsi notamment éviter des conflits de noms entre des objets créés par plusieurs gestionnaires. Ce système de nommage allié au stockage des objets confère également une dimension de persistance des objets dans MAGENTA dont Smalltalk est dépourvu.

Dans le véritable service CMIS, les noms des objets ne sont pas nécessairement des entiers, et on peut faire appel au lien de nommage tout en précisant le nom relatif que l'on veut donner à l'objet.

4.1.2.1.2 La consultation d'objets La requête GET comporte trois champs obligatoires : un nom d'objet, le paramètre "scope" qui détermine la portée de la requête, et enfin le nom de l'attribut que l'on veut consulter. Suivant que le "scope" vaut 0, 1 ou 2, le gestionnaire d'objet consultera respectivement l'objet seul, ses fils immédiats ou tout le sous-arbre ayant pour ancêtre commun cet objet.

Le GET comporte également un champ optionnel qui lui permet d'avoir une sémantique semblable à une requête SQL : le champ "filtre" permet de sélectionner un sous-ensemble d'objets en posant des conditions sur les attributs de ces objets (un peu comme le "WHERE" dans un "SELECT" en SQL). Par exemple :

GET*1.2.3*2*class*status!=!0*

veut dire : à quelles classes appartiennent les objets contenus dans le sous-arbre de racine 1.2.3 et dont le champ **status** est égal à 0 ? (On remarquera en passant que dans l'exemple ci-dessus, on a pu utiliser la méthode "class", qui a été héritée de la class Smalltalk "Object".) Le filtre est également utilisable par les requêtes SET et ACTION.

4.1.2.1.3 La mise à jour d'attributs et l'exécution d'actions Les requêtes SET et ACTION ont une syntaxe semblable, et en fait ont été implémentées de la même manière. En fait de la même façon que dans le GET, le gestionnaire d'objets fait appel à la méthode Smalltalk **perform** pour interpréter le champ "attribut" de la requête (ou le champ "action" pour la requête ACTION) comme une méthode adressée à l'objet désigné.

SET et ACTION ont un champ de plus que le GET, qui permet de spécifier la nouvelle valeur qu'on veut donner à l'attribut dans un SET, ou de passer des arguments pour la méthode dans un ACTION.

4.1.2.1.4 La destruction d'objets La requête DELETE comporte deux champs, l'un permettant de désigner un objet, l'autre pour préciser le scope (cf la consultation d'objets).

A l'utilisation, il s'est avéré que les requêtes implémentées étaient tout-à-fait suffisantes sémantiquement. Ainsi on a pu voir leur utilisation dans les protocoles d'interaction entre assistants (paragraphe 3.3.1). Nous n'avons donc pas vu l'intérêt dans notre cas d'utiliser des langages plus riches tels que KQML. Il est probable néanmoins que l'utilisation de "performatives" KQML tels que **subscribe** aurait limité le nombre de messages échangés.

4.1.2.2 Les classes "GDMO" implémentées

Nous allons maintenant décrire les classes "GDMO" qui sont concernées par les requêtes que nous venons de décrire.

Tout d'abord la classe *GDMOObject*, la racine de toutes les autres classes : elle détient le comportement par défaut de tous les objets "GDMO", tel que les variables à initialiser lors de la création de nouvelles instances. Cette classe à également une variable de classe **name-binding** qui permet de spécifier le lien de nommage pour chaque classe. En général le lien de nommage exprime une relation d'appartenance ou de contenance (cf fig.4.2). Bien sûr les classes qui héritent de GDMOObject peuvent spécifier leur propre lien de nommage.

Nous avons également implémenté tout un ensemble de classes d'objets gérés, dans l'esprit des normes ISO. Nous en avons utilisé quelques-unes pour notre expérimentation sur le réseau local. Ainsi la classe *Computer* comporte un attribut **ps** qui est calculé à chaque requête de consultation en utilisant la commande `ps | tail +2 | tr '\012' '*'`. Nous constatons ainsi que la MIB (la base d'informations de gestion) peut être totalement virtuelle, car les données ne sont pas nécessairement stockées.

La classe *Discriminator* implémente la fonction de gestion-système (décrit dans le paragraphe 2.1.3.1.6) de rapport d'événement. Nous le décrirons dans la partie consacrée au gestionnaire d'événements.

Enfin la classe *Reine* a été utilisée pour notre application-jouet : le problème distribué des n-reines, décrit plus loin.

En fait nous pensons que nous pourrions étendre la possibilité d'envoi de requêtes à toutes les classes Smalltalk. Ainsi, en envoyant une requête de création à la classe *Metaclass*[1], nous pourrions créer de nouvelles classes dynamiquement. On pourrait alors spécifier leur comportement en décrivant les méthodes grâce à des requêtes SET, et les exécuter avec des requêtes ACTION. L'intérêt est multiple :

1. D'ailleurs en un sens nous faisons déjà implicitement appel au niveau Méta en utilisant une variable de classe pour positionner le lien de nommage pour chaque classe

l'enrichissement de la norme GDMO avec la classe *Metaclass* permettrait alors à un gestionnaire de piloter complètement un agent en dictant, modifiant ou enrichissant son comportement. Une telle "programmation orientée message" rapprocherait également MAGENTA de plateformes d'agents mobiles telles que par exemple les "Messagers de communication" [MMTH95].

La figure 4.2 montre une partie de l'arbre d'héritage des classes "GDMO" implémentées, ainsi que certains liens de nommage entre ces classes.

4.1.3 Le gestionnaire d'événements

Le gestionnaire d'événements a comme nous l'avons dit précédemment pour but de vérifier si les objets gérés par MAGENTA doivent émettre des notifications vers d'éventuels gestionnaires. Il effectue ceci en parcourant la MIB, et en vérifiant pour chaque objet si d'éventuelles conditions d'émission d'événement propres à l'objet (dépassement d'un seuil, notification de création de l'objet, ...) sont vérifiées.[2] Dans ce cas, un message de type EVENT-REPORT est envoyé au manager. Notre syntaxe des EVENT-REPORTs est assez libre, puisqu'elle est déterminée en fonction de l'application qui doit la traiter.

De plus le gestionnaire d'événement doit être capable de filtrer les événements qu'un gestionnaire donné ne veut plus recevoir. Nous avons pour cela implémenté la fonction de gestion-système de rapport d'événements décrite dans la norme X721 [ISO90c].

La norme X721 décrit les classes *Discriminator* et *EventForwardingDiscriminator*. La classe Discriminator définit les propriétés communes à tous les filtres. La classe EventForwardingDiscriminator, qui est une sous-classe de Discriminator, est un filtre spécifique aux notifications. Il permet donc de contrôler l'envoi des notifications vers les managers. Voici une description simplifiée de ces classes:

```
discriminator MANAGED OBJECT CLASS
DERIVED FROM top
```

2. Une meilleure méthode aurait été de supprimer cette boucle infinie de parcours de l'arbre en donnant la responsabilité de l'appel du gestionnaire d'événement aux objets eux-mêmes. Ceci est possible, comme l'a suggéré Philippe Reitz, en exploitant par exemple la notion de dépendance entre objets offerte par Smalltalk.

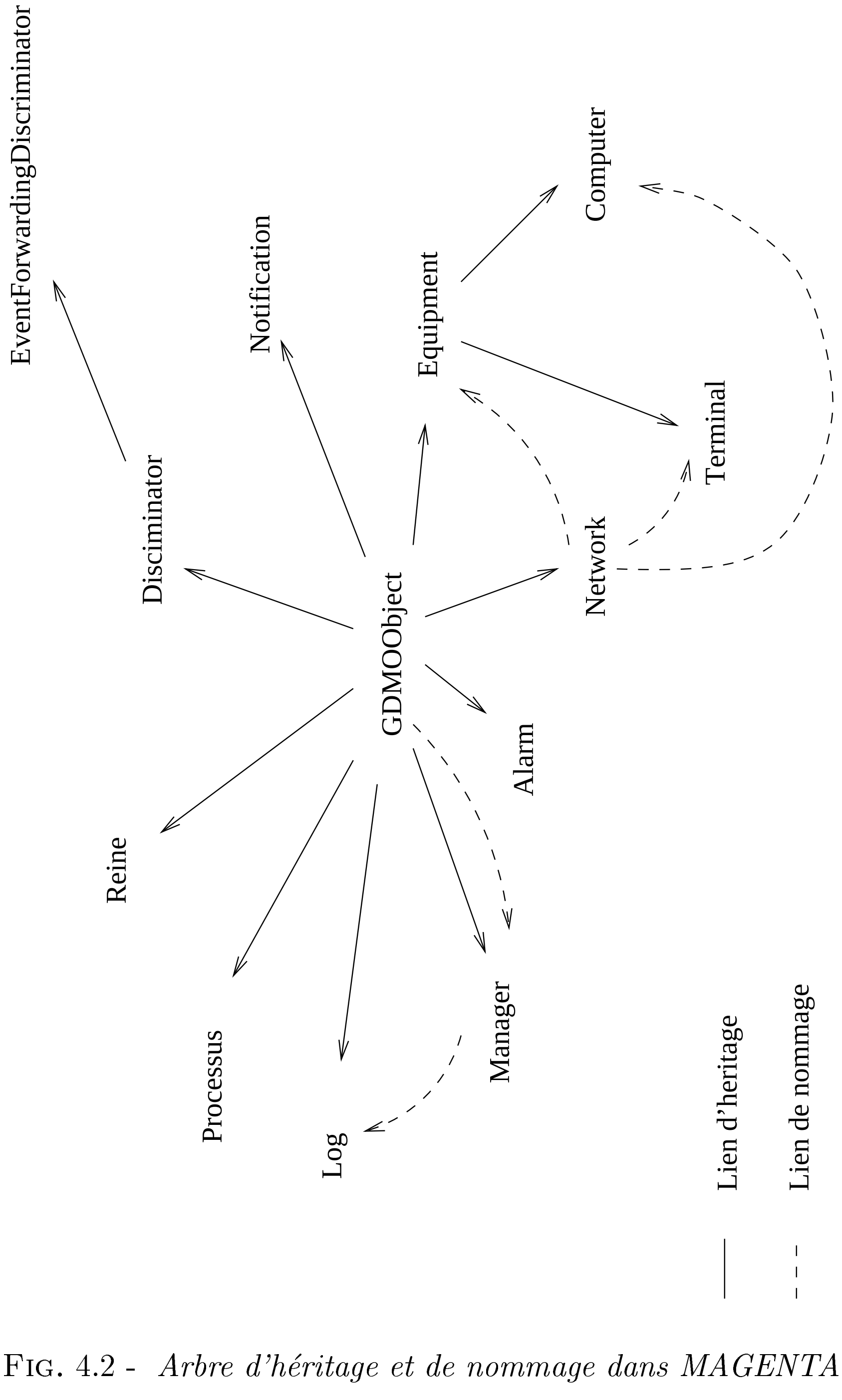


FIG. 4.2 - *Arbre d'héritage et de nommage dans MAGENTA*

```
CHARACTERIZED BY
discriminatorPackage PACKAGE
        BEHAVIOUR
        discriminatorBehaviour BEHAVIOUR
        DEFINED AS "this managed object is used to represent
        the criteria for controlling management services. "
        ATTRIBUTES
        discriminatorId GET
        NOTIFICATIONS
        stateChange
        attributeValueChange
        objectCreation
        objectDeletion


eventForwardingDiscriminator MANAGED OBJECT CLASS
DERIVED FROM discriminator
CHARACTERIZED BY
efdPackage PACKAGE
        BEHAVIOUR
        eventForwardingDiscriminatorBehaviour BEHAVIOUR
        DEFINED AS "this managed object is used to represent
        the criteria that shall be satisfied by potential
        event reports before the event report is forwarded to
        a particular destination. "
        ATTRIBUTES
activeDestination GET
```

Pour l'implémentation des filtres, nous avons donc repris les spécifications de la classe EventForwardingDiscriminator. La norme X721 ne définissant pas précisément l'utilisation des attributs de cette classe, nous les avons interprétés de façon à obtenir les comportements de filtrages qui nous ont paru intéressants :

Cette classe possède un attribut nommé **objectControlList**. Il permet de définir pour chaque classe d'objets GDMO une liste de conditions que ceux-ci doivent respecter afin d'être filtrés. Chaque condition s'applique sur un attribut de l'objet concerné. objectControlList a la forme suivante: #('classe GDMO' 'liste de condition' 'classe GDMO' 'liste de conditions'). Une condition pour la classe Processus peut être par exemple: 'name:=:1.2.3:'. De cette manière, les instances de Processus qui ont pour attribut name une valeur 1.2.3 seront filtrés.

Un second attribut de la classe EventForwardingDiscriminator est activeDestination. Il permet de spécifier l'appartenance d'un filtre à un manager particulier.

Chaque filtre étant un objet GDMO à part entière, les requêtes CMISE vont permettre de réaliser un certain nombre d'actions: création, destruction, modification d'attribut. Le fonctionnement de tels filtres est mis en oeuvre au niveau du gestionnaire d'événements. Il parcourt, comme nous l'avons dit, pendant toute la durée de vie de l'agent auquel il appartient, l'arbre de contenance. En tant qu'objets GDMO, les filtres font partie de cet arbre. Après un parcours complet, le gestionnaire a recueilli les notifications à transmettre aux managers. Il peut aussi connaître l'ensemble des filtres créés. Pour chaque manager "possédant" un filtre, il pourra appliquer à chacune des notifications à lui envoyer les conditions de filtrage. Ce traitement permettra de décider de l'envoi ou non du filtrage de la notification.

L'opérateur aura donc la possibilité de créer à distance un filtre qui lui permettra de ne recevoir que les notifications qu'il désire observer.

Un intérêt supplémentaire du gestionnaire d'événements dans le cas de notre implémentation de notre assistant à la supervision de réseau serait la possibilité de "migration des chroniques" : on peut envisager de faire effectuer la tâche de reconnaissance de certaines chroniques aux agents (au sens OSI) plutôt qu'aux managers. Les agents étant plus "proches" des événements, leur action sera moins dépendante des fluctuations de débit et de délai de transmission des messages. D'autre part on diminue ainsi la charge du réseau pusique l'information ne remontera plus jusqu'au manager. Cette "migration" peut être effectuée très simplement si on définit une classe *chronique* comme étant une sous-classe de DISCRIMINATOR. Un système de reconnaissance de chroniques pourrait faire alors partie intégrante du gestionnaire d'événements de l'agent. Une partie de la base confirmée se trouverait dans ce module, et l'ajout d'une nouvelle chronique se ferait à l'aide de requêtes CREATE et SET envoyées par le manager.

4.1.4 L'application de gestion

Nous avons bien sûr développé notre assistant comme l'application de gestion de MAGENTA. Mais auparavant nous allons présenter une application-jouet (le problème distribué des n-reines) qui illustre le fait que l'utilisation de MAGENTA n'est pas restreinte à la gestion de réseau, et que MAGENTA pourrait aussi être utilisé comme une plateforme de développement de systèmes multi-agents.

4.1.4.1 Les n-reines

MAGENTA facilite en effet l'implémentation de systèmes multi-agents en offrant des outils de communication prêts à l'emploi, un ensemble minimal d'actes de langages représentés par les primitives CMIS, et un mécanisme adéquat pour générer et interpréter ces primitives. Donc voici les étapes à suivre pour programmer chaque agent, et une application pour la programmation d'une variante de l'algorithme distribué des n-reines proposé par Ferber [Fer89] :

1. Créer les classes "GDMO" (et leurs instances) en les faisant hériter de GDMOObject. Dans notre exemple nous créons la classe "Reine", avec pour attributs ses coordonnées.

2. Déterminer les conditions d'émission de notifications pour chaque classe ; dans notre cas, une reine émet une notification à toutes les autres reines à chaque fois qu'elle se déplace.

3. Programmer le comportement que doit avoir l'application de gestion à la réception des notifications. Ici, la reine détermine à la réception de la position des autres si elle est menacée, et si oui alors elle se déplace aléatoirement.

La frontière entre agent réactif et agent cognitif est assez clair avec un tel mode de conception : dans le cas des n-reines, aucun agent n'éprouve le besoin de composer et envoyer des requêtes, chacun se contente d'émettre des notifications ou d'y réagir.

Dans le cas de la collaboration entre assistants, ceux-ci envoient des requêtes pour se consulter mutuellement, ce qui témoigne d'une *intention*.

4.1.4.2 L'assistant

Le comportement de l'assistant a été largement décrit dans le chapitre précédent, on se contentera donc ici d'en décrire l'interface (fig. 4.3) :

L'interface est composée de quatre parties :

- La fenêtre supérieure gauche affiche les événements reçus ;
- La fenêtre inférieure gauche permet l'envoi de requêtes à des agents MAGENTA ;
- La fenêtre supérieure droite affiche le contenu de la base de chroniques confirmées et celui de la base de chroniques non confirmées, et montre ainsi l'activité du système d'apprentissage. L'opérateur peut éditer ou confirmer une chronique. L'interface devrait à l'avenir être plus ergonomique, par exemple les chroniques pourraient être affichées sous forme graphique.
- La partie inférieure droite est réservée au système de reconnaissance de chronique. Il n'a pour le moment pas été intégré à l'assistant.

4.2 L'interface WWW

Dans l'esprit du "Web-based management" (gestion de réseau grâce à un browser WWW) (des exemples de tels systèmes peuvent être consultés aux URLs: http://www.anacapa.com/ et http://www.cabletron.com/webview/), nous avons également développé une interface WWW qui permet à un utilisateur distant de dialoguer avec un programme MAGENTA.

La création des pages WWW a été effectuée grâce au langage Perl qui est souvent utilisé à cette fin. Ces pages permettent de disposer des fonctionnalités minimum d'un manager : saisie et envoi de requêtes CMIS vers un agent MAGENTA, réception de notifications provenant de ce dernier. L'utilisation du langage JavaScript a permis de fournir une interface conviviale pour composer les requêtes.

Comme la création d'un manager WWW doit être effectuée de façon dynamique, il fallait prévoir un protocole d'ajout de ce manager à la liste des managers connus de l'agent. Pour cela nous avons créé une classe GDMO "Manager" dans la hiérarchie d'objets de l'agent. Cette classe a pour attributs le nom de la machine et le numero de port, et pour action la méthode "start". L'insertion d'un nouveau manager se fait donc à l'aide des requêtes suivantes :

```
CREATE*1.2.3*Manager
SET*1.2.3*0*hostName:*darius*
SET*1.2.3*0*port:*1602*
ACTION*1.2.3*0*start*
```

Evidemment la création et l'envoi de ces requêtes sont transparents à l'opérateur, mais permettent à celui-ci de consulter les notifications reçues en cliquant sur le lien hypertexte "messages from MAGENTA". En fait ces notifications ont été stockées dans une instance de la classe log propre au manager, et la consultation des notifications a pour effet l'envoi d'une requête GET pour récupérer le contenu du log, ainsi que l'envoi d'un SET pour remettre le contenu de cette instance à zéro. Ainsi, en cliquant sur le bouton "update", on peut obtenir la liste des notifications reçues depuis la dernière consultation. La raison d'une telle implémentation est que l'affichage dynamique de ces notifications ne semble pas possible si on respecte la norme HTML. Pour y remédier il faudrait implémenter certaines parties de l'interface en Java pour garder une connexion active entre le browser du client et l'application qui se déroule chez le serveur. On pourrait alors représenter graphiquement les notifications reçues, par exemple pour afficher la charge du réseau géré.

4.2.1 Que penser de MAGENTA ?

Nous n'avons pas pour ambition de promouvoir MAGENTA comme plateforme de développement de systèmes multi-agents : ce n'était pas le but que nous recherchions en l'implémentant, et bien que les critères pour attribuer un tel qualificatif ne soient pas encore fixés et soient encore du domaine de la recherche [WJ95], il paraît clair qu'il manque à MAGENTA une surcouche de type AGENT0 (ou tout autre langage manipulant des concepts cognitifs). Toutefois, nous pensons que beaucoup des caractéristiques de MAGENTA pourraient être réutilisées dans l'élaboration d'une véritable plateforme :

- Partir de la mise en communication de machines virtuelles existantes : ainsi nous avons profité de tous les mécanismes que Smalltalk mettait à notre disposition, et nous avons pu utiliser une syntaxe déjà connue. On pourrait de même envisager des systèmes semblables à base de lisp ou prolog ([Rao96] va d'ailleurs d'une certaine façon dans ce sens). Dans notre cas, le choix de Smalltalk nous a permis d'adopter sans aménagement particulier les normes GDMO et CMISE. La pauvreté en terme de langage de représentation de connaissances (si tant est que GDMO et plus généralement le paradigme objet puissent être considérés comme pauvres, et la supériorité d'Ontolingua reste à démontrer) est compensée par l'aspect opérationnel et dénué d'ambiguïté de Smalltalk.

- Dans le même ordre d'idée, nous avons vu que l'introduction d'une classe Metaclass dans notre implémentation de GDMO (ou plus simplement l'extension de la gestion distante des objets à toutes les classes Smalltalk) permet des manipulations puissantes telles que la reprogrammation d'un agent par un autre (bien entendu des gardes-fous doivent être envisagés).

- MAGENTA fournit par ailleurs des primitives simples pour permettre la communication entre agents à travers le réseau, et est un système ouvert qui prend en compte l'apparition et la disparition d'agents, le tout dans l'esprit de DPSK [CJS93].

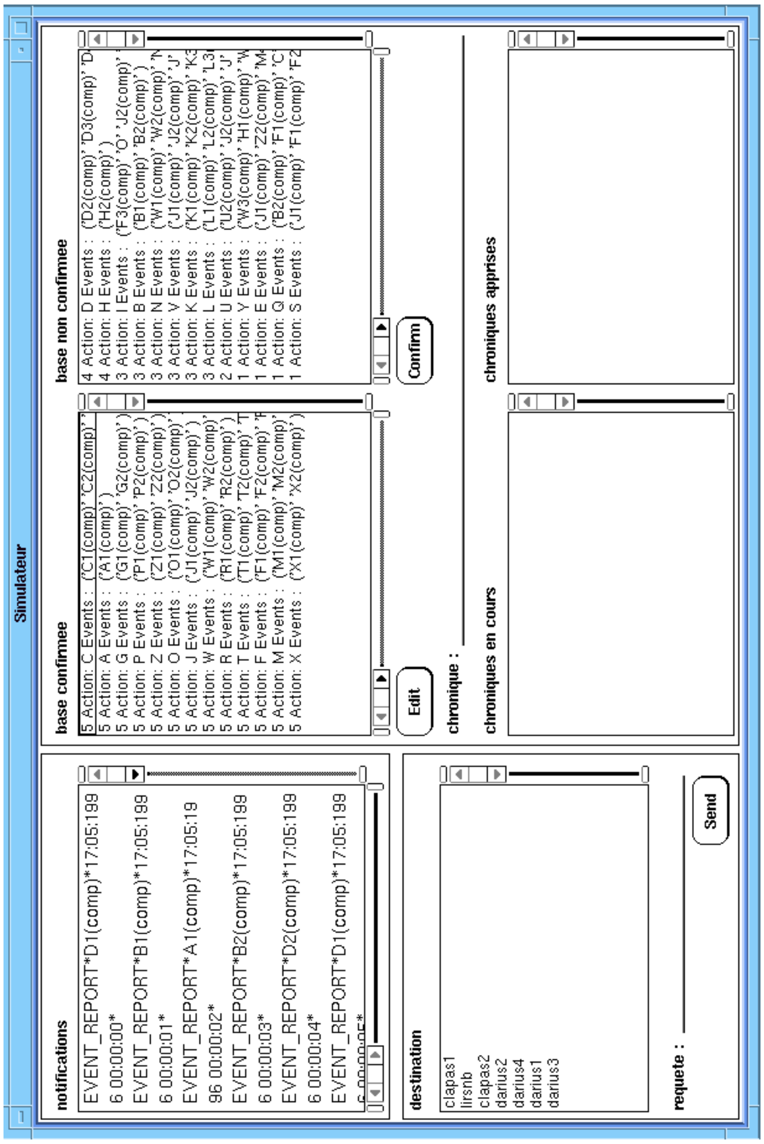


FIG. 4.3 - *L'interface de l'assistant*

Chapitre 5

L'expérimentation

L'expérimentation a consisté à placer MAGENTA dans plusieurs situations où il nous a semblé intéressant d'évaluer les possibilités d'introduire l'assistance à la supervision ou de tester la valeur et la généricité de nos algorithmes d'apprentissage. Rappelons que MAGENTA peut être utilisé aussi bien comme agent et comme gestionnaire (au sens OSI des termes).

5.1 Le réseau local

Il s'agissait de simuler l'administration système des machines du réseau d'enseignement. Pour cela, nous avons lancé un processus MAGENTA en tant qu'agent chargé de récolter des données systèmes provenant de différents processus démons qui testent différentes variations de ressources physiques ou logiques telles que la création et la destruction de processus, d'apparition ou la disparition de machines, l'arrivée ou le départ d'un utilisateur. Ces démons communiquent d'ailleurs ces informations à l'agent à l'aide de notifications de type CMIS de création ou de disparition d'objets, avec pour noms d'objets des identifiants de ces ressources (pid, adresse internet de la machine,...). Au niveau de l'agent nous avons dû créer des classes GDMO correspondant à ces ressources, et associer à chaque classe des conditions d'émission de notification vers un manager MAGENTA. Ces conditions sont pour le moment simplistes : on se contente de notifier la création et la disparition d'objets. Les EVENT-REPORTs correspondants sont donc envoyés vers le manager MAGENTA. La partie

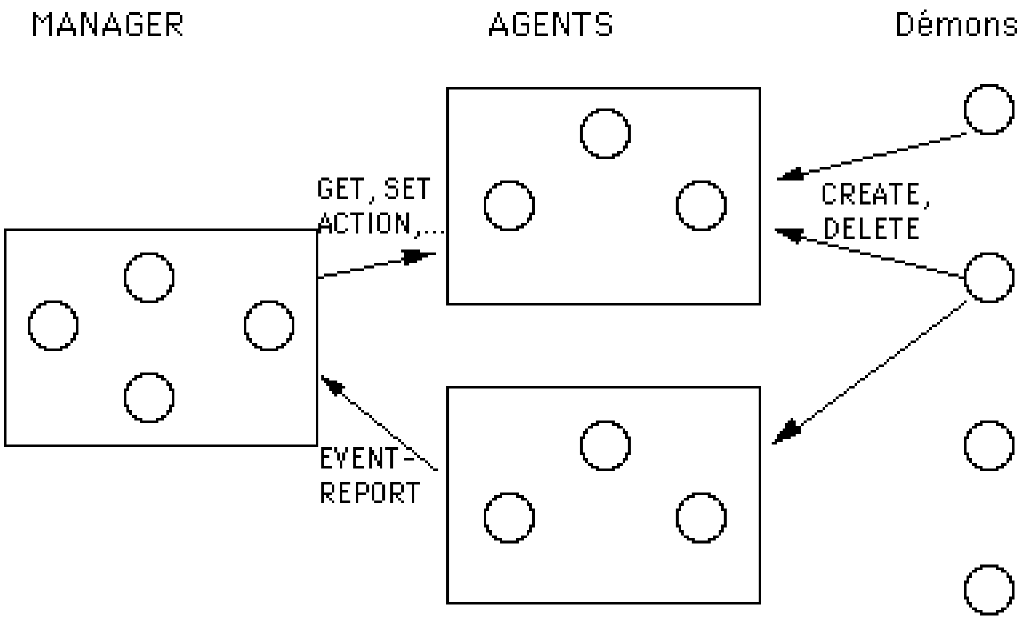


FIG. 5.1 - *Les interactions dans le réseau local*

"application de gestion" de ce manager (qui nous le rappelons fait la spécificité d'un manager MAGENTA par rapport à un simple agent MAGENTA) est l'interface avec l'utilisateur, qui lui permet de visualiser les notifications reçues, et de composer éventuellement des requêtes CMIS sur les objets de la MIB de l'agent MAGENTA. Parmi les requêtes utiles qu'un opérateur peut composer, nous avons identifié la création de filtres, qui permettent de spécifier les conditions d'émission de notifications et de limiter ainsi le nombre de messages envoyés par l'agent. Nous avons vu dans le chapitre précédent l'implémentation de la classe DISCRIMINATOR qui permet la création de tels filtres.

Nous espérons utiliser notre assistant à la supervision pour apprendre le comportement de l'opérateur, et en particulier les cas de création de filtres par celui-ci. Or pour espérer apprendre autre chose que des chroniques triviales, il faudrait une analyse assez fine des besoins et du comportement d'un opérateur, donc un expert en la matière. En attendant une telle coopération, nous nous sommes repliés sur d'autres données, qui sont présentées dans les paragraphes suivants.

Le schéma 5.1 récapitule la mise en place de l'expérimentation au niveau du réseau local.

5.2 Transpac

Il s'agit en fait de fichiers de données générés par le fonctionnement journalier d'un système expert, CHRONOS [Fer], utilisé par un superviseur du réseau Transpac. Ces fichiers sont composés de deux types d'informations :

– les entrées, qui correspondent à des messages émis par des composants du réseau Transpac (commutateurs, liaisons, stations de gestion, rôles,...) ou des logiciels qui les simulent, à destination du superviseur ;

– les sorties, qui correspondent à des diagnostics déclenchés par le système expert à la suite d'inférence de règles de productions déclenchées par la réception des messages en entrée.

Certaines de ces règles correspondent à la reconnaissance de séquences typiques qui caractérisent une panne ou un dysfonctionnement d'un composant du réseau. Notre objectif est d'apprendre et de reconnaître certaines de ces séquences typiques à l'aide de notre algorithme d'apprentissage de chroniques. Il est clair qu'on ne peut pas prétendre retrouver exactement les règles du système expert, ne serait-ce que parce que les chroniques ne peuvent correspondre une à une à des règles CHRONOS, et d'autre part certaines de ces règles ne se seront pas déclenchées suffisamment souvent pendant une journée pour qu'on puisse tirer une généralisation significative de ces règles.

Pour interfacer ces données avec notre algorithme d'apprentissage qui constitue cette fois-ci la partie "application" d'un manager MAGENTA, nous avons simulé l'existence d'un agent qui lit les données et les transforme en EVENT-REPORT s'il s'agit d'entrées, ou en ACTION, s'il s'agit de sorties. Cette lecture de fichier nécessitant un travail de reconnaissance et de mise en forme de chaînes de caractère, nous avons implémenté cet agent dans le langage Perl pour des raisons de simplicité d'utilisation.

Nous avons restreint l'information à extraire du fichier au nom et à la date du message, ainsi que le nom de l'équipement à l'origine du message.

Nous avons lancé notre algorithme sur des données correspondant à une journée d'exécution. Pour ne pas saturer la file d'attente de demandes de connexion vers le manager, nous avons cependant dû temporiser une seconde entre la lecture de chaque ligne du fichier, ce qui est de toute façon largement plus rapide que le débit réel des entrées dont le respect aurait évidemment ralenti considérablement l'expérimentation !

Voici les chroniques générées par notre assistant avec le premier algorithme d'apprentissage et avec un seuil de confirmation (constaté optimal) de 4. Chaque chronique est précédée de sa valeur de confiance "Trust". Nous avons supprimé les chroniques qui avaient une confiance égale à un, ce qui voulaient dire que les actions correspondantes n'avaient été déclenchées qu'une seule fois ; or il y en a beaucoup et en général elles ne sont pas encore assez intéressantes car comportant beaucoup trop d'événements "parasites".

```
Base confirmee (seuil de confirmation automatique: 4)

4 Chronique Suppression(t2) {
         Event(Type:(<>p128(nlg),=p128(nlg)),t0)
         Event(Type:(<>p009(cm),=p009(cm)),t1)
1 <= t1 - t0 <= 139
0 <= t2 - t1 <= 0
}

4 Chronique Suppression(t1) {
         Event(Type:(<>p002(cm),=p002(cm)),t0)
0 <= t1 - t0 <= 33
}

4 Chronique Panne(t3) {
         Event(Type:(<>p089(nst),=p089(nst)),t0)
         Event(Type:(<>de-rol(nrol),=de-rol(nrol)),t1)
         Event(Type:(<>de-stn(nst),=de-stn(nst)),t2)
7 <= t1 - t0 <= 68
2 <= t2 - t1 <= 10
0 <= t3 - t2 <= 0
```

```
}

4 Chronique Panne(t2) {
        Event(Type:(<>p403(nla),=p403(nla)),t0)
        Event(Type:(<>p089(nst),=p089(nst)),t1)
3 <= t1 - t0 <= 32
0 <= t2 - t1 <= 38
}

4 Chronique Recuperer(t1) {
        Event(Type:(<>p082(nst),=p082(nst)),t0)
0 <= t1 - t0 <= 0
}

4 Chronique Pb-rechargement(t1) {
        Event(Type:(<>p501(cm),=p501(cm)),t0)
0 <= t1 - t0 <= 0
}

4 Chronique Probleme(t1) {
        Event(Type:(<>p026(nrol),=p026(nrol)),t0)
0 <= t1 - t0 <= 0
}

4 Chronique La-station(t1) {
        Event(Type:(<>p089(nst),=p089(nst)),t0)
-424 <= t1 - t0 <= 220
}

----------------------------------------------------------

Base non confirmee
```

```
3 Chronique Arret-du-role(t2) {
         Event(Type:(<>p089(nrol),=p089(nrol)),t0)
         Event(Type:(<>p128(nlg),=p128(nlg)),t1)
0 <= t1 - t0 <= 26
0 <= t2 - t1 <= 0
}

2 Chronique silence(t12) {
         Event(Type:(<>de-stn(nst),=de-stn(nst)),t0)
         Event(Type:(<>de-stn(nst),=de-stn(nst)),t1)
         Event(Type:(<>p089(nst),=p089(nst)),t2)
         Event(Type:(<>p089(nst),=p089(nst)),t3)
         Event(Type:(<>p089(nst),=p089(nst)),t4)
         Event(Type:(<>p089(nrol),=p089(nrol)),t5)
         Event(Type:(<>p128(nlg),=p128(nlg)),t6)
         Event(Type:(<>p403(nla),=p403(nla)),t7)
         Event(Type:(<>p128(nlg),=p128(nlg)),t8)
         Event(Type:(<>de-rol(nrol),=de-rol(nrol)),t9)
         Event(Type:(<>p089(nrol),=p089(nrol)),t10)
         Event(Type:(<>p089(nst),=p089(nst)),t11)
2 <= t1 - t0 <= 2
14 <= t2 - t1 <= 14
12 <= t3 - t2 <= 12
2 <= t4 - t3 <= 2
21 <= t5 - t4 <= 21
9 <= t6 - t5 <= 9
8 <= t7 - t6 <= 8
0 <= t8 - t7 <= 0
20 <= t9 - t8 <= 20
4 <= t10 - t9 <= 4
2 <= t11 - t10 <= 2
0 <= t12 - t11 <= 0
}
```

```
2 Chronique silence(t10) {
         Event(Type:(<>p128(nlg),=p128(nlg)),t0)
         Event(Type:(<>p128(nlg),=p128(nlg)),t1)
         Event(Type:(<>p128(nlg),=p128(nlg)),t2)
         Event(Type:(<>p089(nrol),=p089(nrol)),t3)
         Event(Type:(<>de-stn(nst),=de-stn(nst)),t4)
         Event(Type:(<>p128(nlg),=p128(nlg)),t5)
         Event(Type:(<>p090(nst),=p090(nst)),t6)
         Event(Type:(<>p128(nlg),=p128(nlg)),t7)
         Event(Type:(<>p128(nlg),=p128(nlg)),t8)
         Event(Type:(<>p259(cm),=p259(cm)),t9)
72 <= t1 - t0 <= 72
84 <= t2 - t1 <= 84
92 <= t3 - t2 <= 92
123 <= t4 - t3 <= 123
0 <= t5 - t4 <= 0
17 <= t6 - t5 <= 17
0 <= t7 - t6 <= 0
0 <= t8 - t7 <= 0
299 <= t9 - t8 <= 299
-440 <= t10 - t9 <= -440
}

2 Chronique Arret-de-l''UE(t2) {
         Event(Type:(<>p089(nrol),=p089(nrol)),t0)
         Event(Type:(<>p128(nlg),=p128(nlg)),t1)
36 <= t1 - t0 <= 81
-81 <= t2 - t1 <= -36
}

2 Chronique La-ligne(t6) {
         Event(Type:(<>p128(nlg),=p128(nlg)),t0)
```

```
        Event(Type:(<>p128(nlg),=p128(nlg)),t1)
        Event(Type:(<>p128(nlg),=p128(nlg)),t2)
        Event(Type:(<>p128(nlg),=p128(nlg)),t3)
        Event(Type:(<>p128(nlg),=p128(nlg)),t4)
        Event(Type:(<>p128(nlg),=p128(nlg)),t5)
0 <= t1 - t0 <= 62
0 <= t2 - t1 <= 5140
0 <= t3 - t2 <= 153
0 <= t4 - t3 <= 341
78 <= t5 - t4 <= 549
0 <= t6 - t5 <= 208
}

2 Chronique silence(t15) {
        Event(Type:(<>p259(cm),=p259(cm)),t0)
        Event(Type:(<>p275(cm),=p275(cm)),t1)
        Event(Type:(<>p259(cm),=p259(cm)),t2)
        Event(Type:(<>p275(cm),=p275(cm)),t3)
        Event(Type:(<>p259(cm),=p259(cm)),t4)
        Event(Type:(<>p275(cm),=p275(cm)),t5)
        Event(Type:(<>p259(cm),=p259(cm)),t6)
        Event(Type:(<>p275(cm),=p275(cm)),t7)
        Event(Type:(<>p259(cm),=p259(cm)),t8)
        Event(Type:(<>p275(cm),=p275(cm)),t9)
        Event(Type:(<>p259(cm),=p259(cm)),t10)
        Event(Type:(<>p275(cm),=p275(cm)),t11)
        Event(Type:(<>p259(cm),=p259(cm)),t12)
        Event(Type:(<>p275(cm),=p275(cm)),t13)
        Event(Type:(<>p276(cm),=p276(cm)),t14)
11 <= t1 - t0 <= 11
23 <= t2 - t1 <= 23
0 <= t3 - t2 <= 0
17 <= t4 - t3 <= 17
```

```
0 <= t5 - t4 <= 0
11 <= t6 - t5 <= 11
45 <= t7 - t6 <= 45
21 <= t8 - t7 <= 21
70 <= t9 - t8 <= 70
12 <= t10 - t9 <= 12
42 <= t11 - t10 <= 42
13 <= t12 - t11 <= 13
11 <= t13 - t12 <= 11
5 <= t14 - t13 <= 5
76 <= t15 - t14 <= 76
}

2 Chronique Apparition-alarme(t2) {
         Event(Type:(<>p128(nlg),=p128(nlg)),t0)
         Event(Type:(<>p271(cm),=p271(cm)),t1)
1159 <= t1 - t0 <= 1928
0 <= t2 - t1 <= 0
}

2 Chronique Disparition-alarme+Apparition-alarme(t2) {
         Event(Type:(<>p287(cm),=p287(cm)),t0)
         Event(Type:(<>p271(cm),=p271(cm)),t1)
789 <= t1 - t0 <= 865
-865 <= t2 - t1 <= -789
}
```

Nous ne sommes pas familiers avec la sémantique de ces messages. Néanmoins la comparaison avec les séquences typiques dont nous disposions nous permet de dire qu'une bonne partie des chroniques de la base confirmée, notamment pour les chroniques qui ont pour action: *Probleme*, *Recuperer* ou *Suppression*, est juste, mais certaines autres sont tronquées. Ceci est évidemment due à l'entremêlement des événements. D'autres chroniques sont valables mais n'atteignent pas le seuil de confirmation.

Le seuil de confirmation doit être convenablement choisi : en effet s'il est trop grand, les chroniques resteront trop longtemps dans la base non confirmée, ce qui non seulement retarde leur utilisation, mais aussi les expose à des cas d'entremêlement de pannes. En revanche, un seuil trop faible ne discrimine pas suffisamment les événements, et les chroniques risquent de contenir des événements non pertinents. Le seuil de 4 est celui qui a correspondu en pratique au meilleur compromis. Néanmoins certaines chroniques au taux de confiance égal à trois paraissent justes.

En revanche les chroniques "silence" générées sont décevantes, et il faudrait trouver d'autres algorithmes pour les générer correctement.

Certaines contraintes de temps sont négatives : ceci signifie probablement que le premier des deux événements incriminés est superflu. Une heuristique serait d'éliminer directement de tels événements. Une autre raison provient de la façon dont les actions sont datées dans ces données. En fait les dates ne correspondent pas à la date de déclenchement de l'action, mais à la date probable de l'apparition du problème. Ceci explique également que souvent l'intervalle entre le dernier événement de la chronique et l'action soit nul.

Quand deux actions avaient lieu successivement, nous les avons fusionnées, au lieu de créer une chronique "vide" pour la deuxième action. Nous aurions également pu créer deux chroniques, avec les mêmes événements conduisant à deux actions différentes. Ceci serait déjà un pas vers l'utilisation du second algorithme d'apprentissage qui tient mieux compte de l'entremêlement. Nous n'avons cependant pas longtemps utilisé ce second algorithme avec ces données, car du fait que des actions comme "suppression" ou "panne" peuvent être déclenchées par des suites d'événements différentes, nous n'obtenions pas des chroniques satisfaisantes. Le second algorithme d'apprentissage a été testé avec d'autres données que nous présentons dans la suite de ce document.

Cela dit, nous n'avons pas pour objectif de réussir un apprentissage parfait du comportement du système expert CHRONOS, car non seulement il s'agirait d'une tâche très difficile, mais en plus nous ne voulons pas développer un outil ad hoc. Quitte à perdre en efficacité, nous préférons privilégier la généricité. Les deux paragraphes suivants montrent que notre algorithme a pu être utilisé dans des cas assez différents, et pas nécessairement en rapport avec la supervision de réseau.

5.3 Apprentissage sur des traces générées par un robot

Dans le cadre de son stage de DEA, Emmanuel Castro a utilisé notre algorithme (et sa traduction du Smalltalk en C par des étudiants en Maîtrise supervisés par Luc Yriarte) pour détecter les cas de freinage d'urgence chez le robot mobile RAT expérimenté au LIRMM. Ces données nous ont paru très intéressantes, car y apparaissait une nette distinction entre événements et action. De ce fait notre algorithme s'est révélé beaucoup mieux adapté que d'autres algorithmes testés sur ces mêmes données. Pour ces données, c'est le premier algorithme d'apprentissage qui a été utilisé car il s'agissait de n'apprendre que des chroniques déclenchant la même action. Les événements constituant la trace indiquent en général un changement d'allure ou de direction. Voici alors le contenu de la base confirmée, avec un seuil de confirmation de 4 :

```
4 Chronique URGENCE_AA(t4) {
         Event(Type:(<>SUPERURGENT_Start(RAT),=SUPERURGENT_Start(RAT)),t0)
         Event(Type:(<>MOYEN_Start(RAT),=MOYEN_Start(RAT)),t1)
         Event(Type:(<>SUPERURGENT_End(RAT),=SUPERURGENT_End(RAT)),t2)
         Event(Type:(<>MOYEN_End(RAT),=MOYEN_End(RAT)),t3)
0 <= t1 - t0 <= 0
50 <= t2 - t1 <= 1200
50 <= t3 - t2 <= 300
100 <= t4 - t3 <= 350
}

4 Chronique URGENCE_AA(t2) {
         Event(Type:(<>GAUCHE_End(RAT),=GAUCHE_End(RAT)),t0)
```

```
        Event(Type:(<>DROITE_Start(RAT),=DROITE_Start(RAT)),t1)
0 <= t1 - t0 <= 0
70 <= t2 - t1 <= 820
}

4 Chronique URGENCE_AA(t5) {
        Event(Type:(<>SUPERURGENT_End(RAT),=SUPERURGENT_End(RAT)),t0)
        Event(Type:(<>MOYEN_Start(RAT),=MOYEN_Start(RAT)),t1)
        Event(Type:(<>LENT_End(RAT),=LENT_End(RAT)),t2)
        Event(Type:(<>RAPIDE_Start(RAT),=RAPIDE_Start(RAT)),t3)
        Event(Type:(<>MOYEN_End(RAT),=MOYEN_End(RAT)),t4)
0 <= t1 - t0 <= 0
150 <= t2 - t1 <= 150
0 <= t3 - t2 <= 0
150 <= t4 - t3 <= 150
100 <= t5 - t4 <= 250
}

4 Chronique URGENCE_AA(t2) {
        Event(Type:(<>GAUCHE_Start(RAT),=GAUCHE_Start(RAT)),t0)
        Event(Type:(<>DROITE_End(RAT),=DROITE_End(RAT)),t1)
0 <= t1 - t0 <= 0
20 <= t2 - t1 <= 521
}
```

Ces chroniques ont toutes été jugées pertinentes par Emmanuel Castro. Elles constituent un sous-ensemble de tout ce qui pourrait être appris (il n'est a priori pas possible de quantifier cet ensemble, puisqu'il s'agit justement de comprendre le comportment du robot !).

5.4 Apprentissage du comportement d'un programmeur Smalltalk

Dans le cadre de son stage de DEA, Jean-David Ruvini a également utilisé plusieurs algorithmes d'apprentissage pour détecter des régularités dans le comportement d'un programmeur Smalltalk. Le but est, toujours dans le même esprit que les agents d'interface de Pattie Maes, d'automatiser ces régularités pour supprimer certaines tâches fastidieuses que le programmeur a à effectuer. La distinction entre événements et actions étant moins nette dans ces données, notre algorithme semblait au départ moins adapté. Or il s'est finalement comporté de façon semblable aux autres algorithmes expérimentés. L'algorithme a notamment détecté le fait que lors d'un bug, le programmeur avait à chaque fois besoin de déplacer et d'agrandir la fenêtre de débuggage.

5.5 Données pour tester la gestion de l'entremêlement

Pour évaluer le second algorithme, nous avons travaillé sur des données fournies par Gilles Deflandre. Ces données sont créées par un générateur de données aléatoires où l'on peut paramétrer le degré d'entremêlement des chroniques à retrouver. Dans ces données, l'action est représentée par une lettre de l'alphabet. Les événements conduisant à cette action sont alors représentés par cette même lettre suivie d'un index qui désigne leur rang dans la séquence. Les chroniques ont des "longueurs" (nombre d'événements) variables. Les dates dans ces données ne sont pas significatives : on ne créera donc pas de contraintes temporelles.

Nous avons effectué les tests en faisant varier trois paramètres :

- le seuil de confirmation des chroniques : plus ce seuil est haut, plus normalement la qualité des chroniques apprises est meilleure, puisqu'on laisse plus de possibilités à la chronique de se "débarrasser" des événements parasites. En contrepartie on ralentit l'apprentissage, et on s'expose également à l'éventualité de tronquer la chronique ;

- la taille du buffer : plus cette taille est grande, plus on diminue le risque de rater les premiers événements déclenchant la chronique. En contrepartie, on augmente le temps de calcul du plus long sous-mot commun, et on augmente aussi le risque d'un mauvais choix de ce sous-mot ;

- le traitement à effectuer lors de la détection d'incohérences (constat de relations d'inclusions entre sous-séquences conduisant à des actions différentes). Trois choix sont possibles :

  - On confirme tout de même la chronique, en admettant la possibilité qu'une même séquence puisse déclencher plusieurs actions ;
  - On garde la chronique dans la BNC, en supposant que l'incohérence est due à l'existence d'événements parasites, et qu'elle disparaîtra bientôt ;
  - On élimine la chronique, parce qu'elle est de mauvaise qualité.

  Normalement la décision doit être prise par l'opérateur au cas par cas. Nous avons constaté qu'avec les données dont nous disposions, le troisième choix était le plus pertinent en général. Nous avons donc décidé de nous limiter à ce choix par la suite pour diminuer l'explosion combinatoire du nombre de tests.

Voici le fichier de résultats obtenu avec les meilleurs réglages :

```
Assistant : 1
Donnees : fichier test-lirmm1.tst
buffer : 34
confirmation : 5
incoherence : elimination

base confirmee : 13 chroniques
                  10 entieres
                  0 avec des evenements en trop
                  3 avec des evenements en moins
```

```
listeChroniquesConfirmees :

5 Action: C Events :  ('C1' 'C2' 'C3' )
5 Action: A Events :  ('A1' )
5 Action: G Events :  ('G1' 'G2' 'G3' )
5 Action: P Events :  ('P1' 'P2' )
5 Action: Z Events :  ('Z1' 'Z2' )
5 Action: O Events :  ('O1' 'O2' )
5 Action: J Events :  ('J1' 'J2' )
5 Action: W Events :  ('W1' ''W2' )
5 Action: R Events :  ('R1' 'R2' )
5 Action: T Events :  ('T1' 'T2' 'T3' 'T4' )
5 Action: F Events :  ('F1' 'F2' 'F3' 'F4' )
5 Action: M Events :  ('M1' 'M2' 'M3' 'M4' 'M5' )
5 Action: E Events :  ('C'' 'E5' )

listeChroniquesNonConfirmees :

4 Action: D Events :  ('D1' 'D2' 'D3' 'D4' )
4 Action: H Events :  ('H2' )
3 Action: I Events :  ('O3' 'D2' 'X' 'J1' 'F1' 'F2' 'F3'
'O' 'J2' 'J' 'J' 'F4' 'A1' 'A' 'I1' 'I1' 'D3' 'D4' )
3 Action: B Events :  ('B1' 'B2' )
3 Action: N Events :  ('Z2' 'Z' 'W1' 'W2' 'N3' 'N4' )
3 Action: K Events :  ('E' 'K1' 'K2' 'K3')
3 Action: L Events :  ('L1' 'L2' 'L3' 'L4' )
3 Action: V Events :  ('X1' 'X2' 'V1' 'V2' 'X' 'V3' 'V4' )
2 Action: U Events :  ('V3' 'V1' 'J1' 'Y1' 'Y' 'U1' 'U2'
'J2' 'J' 'X' 'U3' 'P1' 'V2' 'U1' 'U2' 'V3' 'V4' 'U4' 'V4'
'V' 'P2' 'U3' 'P' 'V' 'D1' 'U5' 'D2' 'Z1' )
1 Action: Q Events :  ('C3' 'G' 'A1' 'A' 'H2' 'B1' 'B2'
'F1' 'C' 'M1' 'E1' 'E2' 'B' 'M2' 'F2' 'F3' 'E3' 'F4' 'M3'
'Q1' 'M4' 'M5' 'M' 'M1' 'M2' 'H' 'F' 'Z1' 'E4' 'M3'
```

```
'J1' 'Z2' 'M4' 'E5' )
1 Action: Y Events :  ('B2' 'N3' 'W1' 'B' 'N4' 'N' 'W3'
'H1' 'W2' 'W' 'C1' 'H2' 'C2' 'C3' 'C' 'W3' 'W' 'G1' 'H'
'X1' 'R1' 'G2' 'G3' 'R2' 'R' 'G' 'X2' 'G' 'V1' 'V2' 'V3'
'V1' 'J1' 'Y1' )
1 Action: S Events :  ('J1' 'X2' 'J2' 'O3' 'D2' 'X' 'J1'
'F1' 'F2' 'F3' 'O' 'J2' 'J' 'J' 'F4' 'A1' 'A' 'I1' 'I1'
'D3' 'D4' 'I' 'I' 'D' 'F' 'Z1' 'J1' 'S1' 'I1' 'S2' 'S3'
'Z2' 'J2' 'J' )
```

Ces résultats indiquent que, pour la configuration optimale, le système obtient parfaitement 10 des 13 chroniques qui se sont répétées au moins 4 fois, ce qui est un comportement très honorable.

Les courbes suivantes montrent, pour un seuil de confirmation donné, l'évolution de la qualité des chroniques apprises en fonction de la taille du buffer. Nous considérons qu'il vaut mieux apprendre des chroniques avec des événements en trop que des chroniques tronquées.

Ces courbes confirment qu'après avoir atteint une valeur optimale, la qualité de l'apprentissage décroît à nouveau, et que donc la taille du buffer ou le seuil de confirmation ne peuvent pas être choisis aussi grands que l'on veut. D'autre part on remarque que la taille optimale du buffer est indépendante de la valeur du seuil de confirmation. Des expériences menées avec d'autres fichiers de données confirment ces conclusions.

Les résultats obtenus avec le second algorithme semblent encourageants. Cependant il doit être possible de les améliorer, par exemple en essayant de limiter le caractère heuristique du calcul du plus long sous-mot commun. Une solution serait par exemple de ne pas se contenter de garder seulement le plus long sous-mot commun, mais les n plus longs, sauf au moment de la confirmation de la chronique, où on ne prendrait plus que le plus long.

La section suivante présente notre expérimentation de l'apprentissage multi-assistant, dont on espère qu'il accélèrera l'apprentissage, et même qu'il contribuera à la diminution des chroniques tronquées.

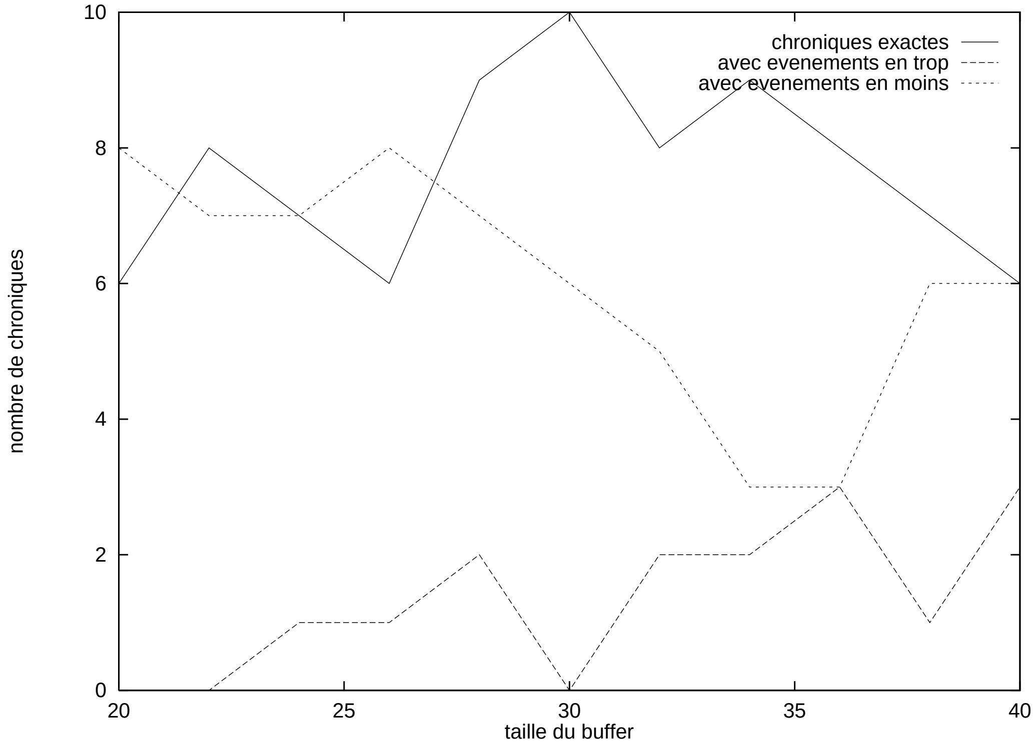


FIG. 5.2 - *Chroniques apprises : seuil de confirmation = 4*

5.6 Apprentissage collaboratif

Pour tester l'apprentissage multi-assistants, nous avons utilisé les mêmes données que celles pour l'apprentissage en présence d'entremêlement. Trois types d'expérimentation sont menées :

- données égales pour chaque assistant ;
- données différentes mais les chroniques à apprendre sont les mêmes ;

  (Pour ces deux cas, il n'est donc pas nécessaire d'utiliser le protocole d'exploration)
- données différentes et chroniques des différentes pour chaque assistant (cas le plus réaliste).

Il y a une véritable explosion combinatoire du nombre de tests à mener en fonction du nombre de paramètres à faire varier. Nous n'avons donc pas pour l'instant mené

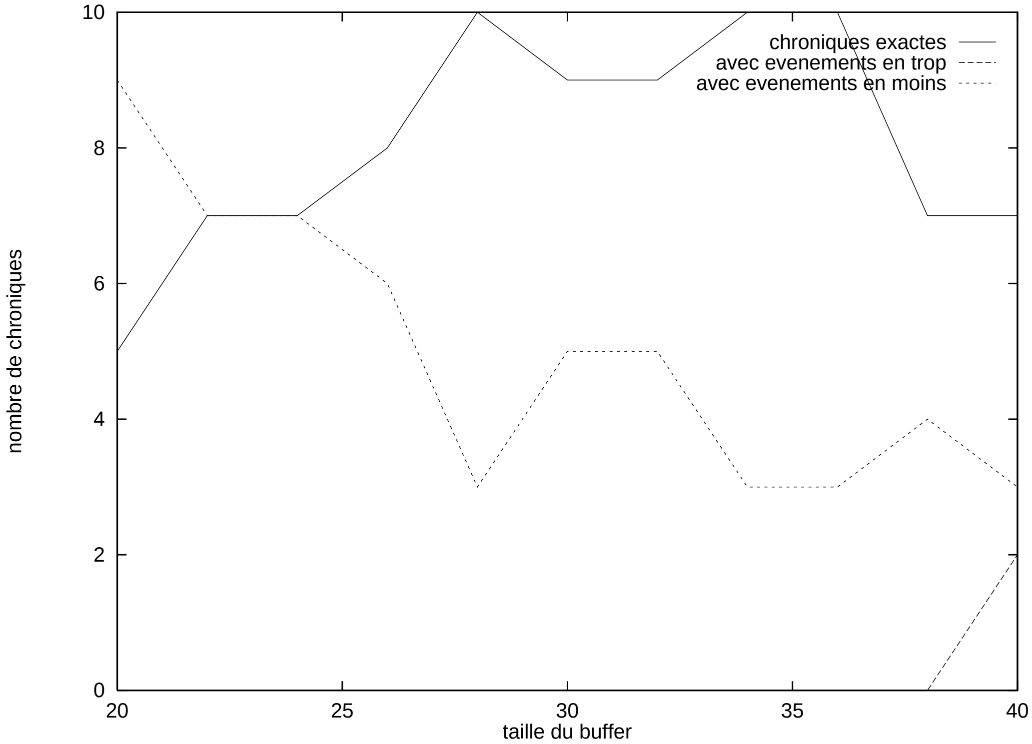


FIG. 5.3 - *Chroniques apprises : seuil de confirmation = 5*

ces expérimentations à terme. Nous avons néanmoins constaté pour le moment que, dans les deux premiers cas, la collaboration accélère effectivement l'apprentissage, ce qui confirme les résultas obtenus par Maes. D'autre part, à la fin de la réception des données, on constate que certaines chroniques qui n'étaient pas présentes dans la base confirmée dans l'apprentissage solitaire le sont dans l'apprentissage collaboratif, grâce à la méthode d'évaluation des réponses des autres assistants qui permet d'augmenter la confiance dans les chroniques.

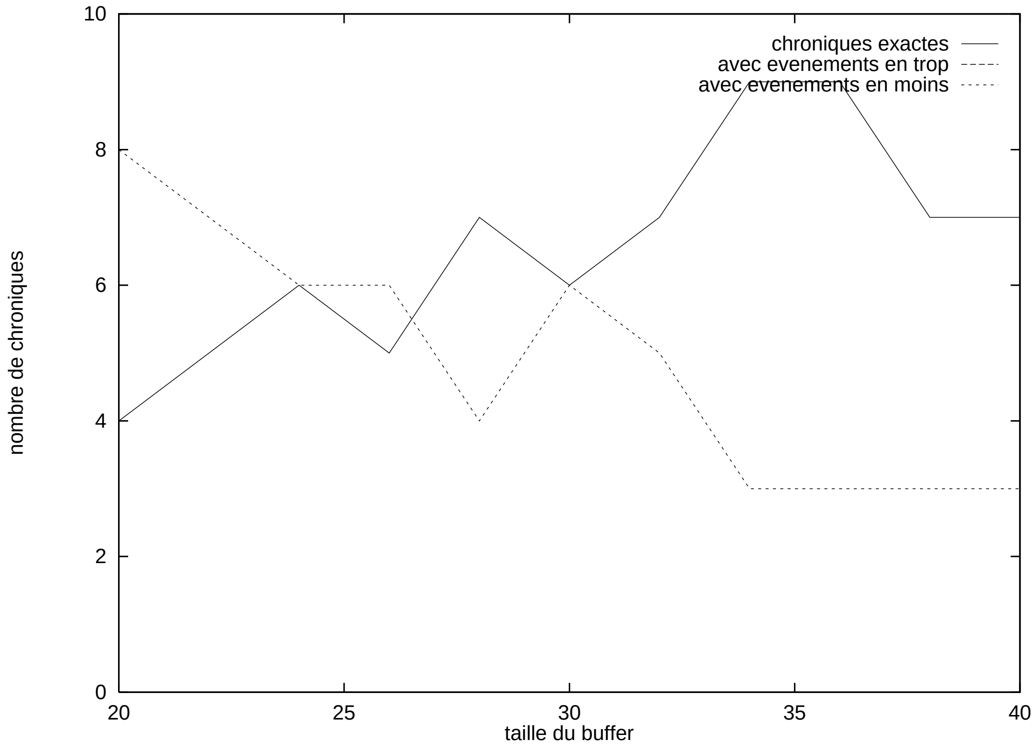


FIG. 5.4 - *Chroniques apprises : seuil de confirmation = 6*

Chapitre 6

Conclusion et perspectives

Nous avons présenté dans cette thèse un système d'assistance à la supervision de réseau à base d'agents d'interfaces qui observent le comportement des opérateurs de supervision de réseau pour l'apprendre, et qui collaborent entre eux pour améliorer cet apprentissage. Deux algorithmes originaux d'apprentissage ont été présentés, ainsi que les protocoles d'interaction nécessaires pour permettre la collaboration.

L'assistant a été intégré dans un prototype de simulation d'une plateforme de gestion de réseau, MAGENTA, qui a présenté des caractéristiques intéressantes qui pourraient être utilisées dans la perspective de l'élaboration d'une plateforme de développement de systèmes multi-agents.

Les expérimentations menées avec MAGENTA pour valider les algorithmes ont donné dans la plupart des cas des résultats satisfaisants, à condition d'effectuer le bon choix d'algorithme en fonction du contexte lié aux données à traiter (simultanéité de pannes, multiplicité des causes d'une même panne).

L'utilisation d'agents d'interface en supervision de réseau est une voie nouvelle et prometteuse. Bien sûr, les techniques utilisées devront être améliorées et adaptées au cas par cas. Cela dit les algorithmes d'apprentissage présentés dans ce document semblent constituer un bon point de départ par leur généricité qui leur a permis d'être expérimentés dans des domaines très différents sans aucune adaptation particulière. Parmi les améliorations possibles des algorithmes, citons par exemple l'apprentissage d'un ordre partiel plutôt qu'un ordre total entre événements dans le cas du second algorithme d'apprentissage : on pourrait par exemple remplacer le calcul du plus long sous-mot commun par un calcul d'intersection ensembliste (les éventuels doublons devront être indexés pour éviter toute ambiguïté, ce qui est déjà une hypothèse forte en soi). On garderait (en élargissant bien sûr les intervalles de temps) alors à tout moment toutes les relations de précédence entre événements qui ne seraient pas contredites. Une autre amélioration serait un passage au "premier ordre" pour augmenter l'expressivité des chroniques : on pourrait ainsi raisonner par exemple sur les types de composants qui ont émis tel ou tel événement.

Il serait maintenant nécessaire de poursuivre les tests, et notamment de faire des expérimentations en "grandeur nature", par exemple sur un réseau de logiciels de supervision appartenant à un même fabricant, ce qui garantirait une certaine normalisation des messages traités par les différents assistants, et permettrait donc une collaboration entre eux.

D'autre part, MAGENTA a mis en lumière certains concepts présents dans les normes de gestion de réseau et qui pourraient être utilisés dans les systèmes multi-agents, ou encore dans le domaine de la programmation orientée objet. Ces aspects ont été déjà discutés dans le chapitre consacré à MAGENTA.

L'utilisation de protocoles d'interaction pour favoriser la coopération entre agents a trouvé dans notre assistance collaborative une bonne application, et a montré leur importance. Nous pensons qu'il faut aller plus loin dans ce domaine, en concevant par exemple des bibliothèques de protocoles, et aussi un "méta-protocole", qui permettrait aux assistants de connaître, choisir et utiliser leurs protocoles communs.

Nous pensons enfin que le domaine des agents d'interface est maintenant suffisamment mûr pour que l'on puisse envisager des plateformes de génération d'agents d'interface. Elles consisteraient en des bibliothèques d'algorithmes de suivi et d'apprentissage, et le choix d'un formalisme donné déterminerait les algorithmes à utiliser.

Bibliographie

[Ang] D. Angluin. Inference of reversible languages. In *Journal of the Association for Computing Machinery, 29*, pages 741–765.

[ARP92] ARPEGE. *Gestion de Réseaux - Concepts et Outils.* CNET-MASSON. Masson, 1992.

[Bou92] Thierry Bouron. *Structures de communication et d'organisation pour la coopération dans un univers multi-agent.* Thèse d'université, Université de Paris VI, 1992.

[Bri88] J.P. Briot. From objects to actors : Study of a limited symbiosis in smalltalk-80. Rapport de Recherche 88-58 RXF, LITP, Université Paris VI, Septembre 1988.

[CC93] Jean Pierre Claudé and Michelle Claudé. *Gestion des Réseaux Informatiques.* Réseaux et Systèmes. Eyrolles, Septembre 1993.

[cdpiP95] collectif du projet inter PRC. Gestion de l'évolutif et de l'incertain dans une base de connaissances. In *Actes des JN du PRC-GDR Intelligence Artificielle*, Nancy, 1995. Teknea.

[CJS93] Enric Cardozo and Yves Demazeau Jaime Sichman. Using the active object model to implement multi-agent systems. In *Proceedings of the IEEE Conference TAI '93 on Tools with AI*, Boston, Novembre 1993.

[CL90] Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. In *Artificial Intelligence*, 1990.

[CL95] Philip R. Cohen and Hector J. Levesque. Communicative actions for artificial agents. In *ICMAS 95 Proceedings*, pages 65–72, 1995.

[DBK94] Yves Demazeau, Olivier Boissier, and Jean Luc Koning. Using interaction protocols to control vision systems. In *International Conference on Systems, Man and Cybernetics*, San Antonio, Octobre 1994. IEEE.

[Dij] E.W. Dijkstra. A note on two problems in connection with graphs. In *Numer. Math, vol.1, pp. 269-271, Oct. 1959.*

[Dou94] Christophe Dousson. *Suivi d'évolutions et reconnaissance de chroniques.* Thèse d'université, Université Paul Sabatier, Toulouse, LAAS, Septembre 1994.

[EDQ96] Babak Esfandiari, Gilles Deflandre, and Joël Quinqueton. An interface agent for network supervision. *Intelligent Agents for Telecom Applications (à paraître)*, 1996.

[EDQD96] Babak Esfandiari, Gilles Deflandre, Joël Quinqueton, and Christophe Dony. Agent-oriented techniques for network supervision. *Annals of Telecommunications*, pages 521–529, 1996.

[EN96] Babak Esfandiari and Richard Nock. Chronicle learning for assistance to a network supervision operator. Research Report 96028, Lirmm, Janvier 1996.

[EPQD96] Babak Esfandiari, Michel Plu, Joël Quinqueton, and Gilles Deflandre. A multi-agent perspective for assistance to a network supervision operator. In *MAAMAW poster session*, 1996.

[EQKD95] Babak Esfandiari, Joël Quinqueton, Jean-Luc Koning, and Yves Demazeau. Systèmes multi-agents et gestion de réseaux. In *Actes des JN du PRC-GDR Intelligence Artificielle*, pages 317–330, Nancy, 1995. Teknea. rr LIRMM 94077.

[EQT95] Babak Esfandiari, Joël Quinqueton, and Richard Terrat. Designing a multi-agents environment by using network management concepts and standards. In *Use of Object-Oriented Technology for Network Design and Management*, Aarhus, DK, Août 1995. ECOOP'95 Workshop 6.

[EW94] Oren Etzioni and Daniel Weld. A softbot-based interface to the internet. *Communications of the ACM*, 37(7):72–76, July 1994.

[Fer] Albert Feroldi. Supervision du réseau transpac : le système chronos. In *rapport technique CNET*.

[Fer89] J. Ferber. *Objets et agents : une étude des structures de représentation et de communication en IA*. Etat, Univ. de Paris VI, Juin 1989.

[Gai91] D. Gaiti. *L'utilisation des techniques de l'intelligence artificielle pour la gestion des réseaux*. Thèse d'université, Université Paris 6, 1991.

[Gai92] Dominique Gaiti. Architecture intelligente pour la gestion des réseaux. Rapport de recherche, MASI-IBP, Paris, 1992.

[Gan] J. G. Ganascia. L'hypothèse du "knowledge level" : Théorie et pratique.

[GH92] Francisco J. Garijo and Donn Hoffman. A multi-agent architecture for operation and maintenance of telecommunication networks. In Jean Paul Haton, editor, *Proceedings of the 12th Int. Avignon Conference*, pages 427–436. EC2 and AFIA, EC2, Paris, Juin 1992.

[Gha94] Malik Ghallab. Past and future chronicles for supervision and planning. In Jean Paul Haton, editor, *Proceedings of the 14th Int. Avignon Conference*, pages 23–34. EC2 and AFIA, EC2, Paris, Juin 1994.

[GK94] M. R. Genesereth and S. P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):100–105, Juillet 1994.

[GP92] Dominique Gaiti and G. Pujolle. *L'intelligence dans les Réseaux*. Eyrolles, Octobre 1992.

[Gru93] T.R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. In *KSL 93-4, Knowledge System Lab*, 1993.

[HS77] J.W. Hunt and T.G. Szymanski. A fast algorithm for computing longest common subsequences. In *Commu. ACM, 20, 5*, pages 350–353, 1977.

[ISO90a] ISO/DIS 9595. *Information Technology - Open Systems Interconnection - Common Management Information Service Definition*, Janvier 1990. 2nd DP N3070.

[ISO90b] ISO/DP 10165-4. *Information Technology - Structure of Management Information - Part 4. Guidelines for the Definition of Managed Objects*, Juin 1990.

[ISO90c] ISO/IEC 10165-2. *Information Technology - Open Systems Interconnection - Systems Management - Part 2 : Definition of Management Function*, Juin 1990.

[ISO95] ISO/DP 10165-1. *Recommendation X.725 : Information Technology - Structure of Management Information - Part 7 : General Relationship Model*, Juin 1995.

[KDEQ95] Jean-Luc Koning, Yves Demazeau, Babak Esfandiari, and Joël Quinqueton. Quelques perspectives d'utilisation des langages et protocoles d'interaction dans le contexte des télécommunications. In *Actes des J. Francophones IAD et SMA*, Chambéry, Mars 1995. AFCET-AFIA.

[Kle86] J. De Kleer. An assumption-based tms. In *Artificial Intelligence 28*, pages 127–162, 1986.

[KLPV87a] M. Kearn, M. Li, L. Pitt, and L.G. Valiant. On the learnability of boolean formulae. In *ACM Symp. on Theory of Computing*, pages 285–295, Mai 1987.

[KLPV87b] M. Kearn, M. Li, L. Pitt, and L.G. Valiant. Recent results on boolean concept learning. In *IWML 87*, pages 337–352, Juin 1987.

[KMP77] D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern-matching in strings. In *SIAM J. on Computing*, pages 323–350, 1977.

[LC92] C. Lecerf and D. Chaumel. *Les normes de gestion de réseau à l'ISO*. CNET-MASSON. Masson, 1992.

[LMM94] Y. Lashkari, M. Metral, and P. Maes. Collaborative interface agents. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*. AAAI-Press, 1994.

[Mae94] Pattie Maes. Learning interface agents. In *Proceedings of the 1994 International Symposium on Next Generation Computer Interfaces*, 1994.

[MK93] Pattie Maes and Robyn Kozierok. Learning interface agents. In *Proceedings of the 11th Nat Conf on Artificial Intelligence*. AAAI, MIT-Press/AAAI-Press, 1993.

[MMTH95] G. Di Marzo, M. Muhugusa, C. F. Tschudin, and J. Harms. The messenger paradigm and its impact on distributed systems. In *ICC'95 International Workshop on Intelligent Computer Communication*, pages 79–94. Ioan Alfred Letia, Claus Unger, Editors, Juin 1995.

[Puj] Guy Pujolle. In *Les Réseaux*. Eyrolles.

[Rao96] Anand Rao. Agentspeak(l): Bdi agents speak out in a logical computable language. In *Proceedings of the 7th MAAMAW workshop*, pages 42–55. Springer, Janvier 1996.

[RQ93] Philippe Reitz and Joël Quinqueton. Mosca : un environnement d'apprentissage multi-agents. In *Ecole sur l'Apprentissage Automatique, PRC-GDR Intelligence Artificielle*. Inria, Sophia Antipolis, Saint Raphaël, France, Mars 1993.

[RRCS88] S. Rabie, A. Rau-Chaplin, and T. Shibahara. Dad: a real-time expert system for monitoring of data packet networks. *IEEE Network Magazine*, 2(5):29–34, Septembre 1988.

[SARJ94] Didier Siboni, Anasser Ag-Rhissa, and Shamliang Jiang. Une fonction intelligente de gestion globale des incidents pour un hyperviseur de réseaux hétérogènes. In Jean Paul Haton, editor, *Proceedings of the 14th*

Int. Avignon Conference, pages 379–387. EC2 and AFIA, EC2, Paris, Juin 1994.

[SH93] J.C. Schlimmer and L.A. Hermens. Software agents: Completing patterns and constructing user interfaces. *Journal of Applied Intelligence Research*, 1:61–89, 1993.

[Sho90] Yoav Shoham. Agent oriented programming. Research Report STAN-CS-90-1335, Stanford University, Octobre 1990.

[Smi80] R. Smith. The contract net protocol: High level communication and control in distributed problem solver. *IEEE Transactions on Computers*, 29, 1980.

[Tan91] A. Tannenbaum. In *Réseaux: Architectures, Protocoles, Applications*. InterEditions, 1991.

[Val84] L.G. Valiant. A theory of the learnable. In *Com. of the ACM*, pages 1134–1142. AAAI, 1984.

[WJ95] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 2(10), 1995.

[ZTD88] W. Zhan, S. Thanawastien, and L. Delcambre. Simnetman: an expert workstation for designing rule-based network management system. *IEEE Network Magazine*, 2(5):35–42, Septembre 1988.